Novel Orientation Representations for Deep Learning-Based Instance Segmentation

DANIEL KLUVANEC

Novel Orientation Representations for Deep Learning-Based Instance Segmentation

Daniel Kluvanec

A Thesis presented for the degree of
Doctor of Philosophy

Department of Computer Science
Durham University
United Kingdom
September 2024

Abstract

This thesis is rooted in Computer Science and Deep Learning, with the primary application being in the interpretation of 3D seismic tomographic images, particularly the identification of geological faults therein. Deep Leaning methods can aid and automate parts of the seismic interpretation process. The outcome of the work in this thesis is a method that can predict the location of faults and separate them out into non-intersecting fault segments. These segments can then be selectively visualised in 3D. The method is powered by a convolutional neural network that predicts the location and orientation of faults, which are separated into segments using a post-processing algorithm that can run on the GPU and is scaleable to large seismic volumes. The means for the neural network to predict orientation is the biggest scientific contribution of this thesis, with four novel 3D representations. These representations are negation-invariant and continuous ($f(\mathbf{v}) = f(-\mathbf{v})$). The work in this thesis also has relevance in other fields. A 2D counterpart to the task of fault separation is the separation of overlapping chromosomes for karyotyping. My work proposes a method for using orientation to separate chromosome instances and uncovers issues with existing approaches in the field, namely the use of semantic segmentation directly and issues with an existing dataset.

Declaration

The work in this thesis is based on research carried out at the Department of Computer Science, Durham University, United Kingdom. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

First, I would like to thank Noura Al Moubayed, my primary academic supervisor. You have been there for me from the very start to the finish, always keeping my best interest at heart and guiding me through the process of doing research and a PhD.

My brother, Tom Wood, you have been more influential in my life than you can imagine. Not only have you stepped up to the task whenever I needed some help, getting me to keep going when the going was tough, helped me to reflect on my challenges, or reviewed my writing when I needed another pair of eyes. You are the person with whom I can honestly be open about anything and are the single biggest influence on my personal development. It means a lot.

I would like to thank Ken McCaffrey and Thomas Phillips, my academic supervisors from the Geosciences, for your guidance through my work and your support with everything I needed to do research on seismic imaging.

I would like to thank all my colleagues and members of Noura's research group. You are the research community that I am proud to be part of, people with whom I can discuss ideas and who drive me forward intellectually.

I would like to thank Durham University, GeoTeric, and ERDF for providing me with the opportunity and funding to do a PhD.

Lastly, I would like to thank all my friends and family, who are very dear to me. You have been a meaningful part of my life and therefore very much part of this thesis as well.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This PhD thesis is primarily rooted in the fields of Computer Science and Deep Learning, contributing to both theoretical advancements and practical applications. The theoretical contributions are twofold. First, it introduces novel representations of orientation, specifically designed to be effectively predicted by Deep Learning models in both 2D and 3D contexts. Second, it explores how Instance Segmentation can be performed by leveraging the predicted orientations, building on the foundation of semantic segmentation. These theoretical contributions are applied to two distinct fields. The first application is in the medical domain, where the focus is on segmenting overlapping chromosomes for karyotyping, using two-dimensional images. The second application lies within geoscience, where the goal is to analyse three-dimensional seismic tomographic images to identify faults.

A cornerstone of this thesis is the concept of orientation - how it can be predicted and how its predictions can be effectively utilized? For the purpose of this work, orientation is defined as the set of all lines passing through the origin. This concept is analogous to direction expressed as the set of all unit-vectors, but without the sense of the vector. In other words, any vector $\mathbf{v}$ and its negative $-\mathbf{v}$ have the same orientation. To assign orientation to various real world objects, we can look at it

as the orientation of the tangent line going across a chromosome in 2D, or as the orientation of a line that is normal to a fault plane in 3D.

When predicting the orientation of an object using a deep learning model, a key challenge arises: how to represent this orientation in a form that the model can predict and that we can interpret? This is not a trivial task because the representation must adhere to two crucial principles to be effective: uniqueness and continuity. This means that each orientation is represented using a unique vector of values, and similar orientations should be represented by vectors with similar values. Achieving these principles is particularly challenging in three dimensions. I have designed orientation representations that satisfy these criteria and have empirically evaluated their effectiveness. The 2D theory and applications are discussed in Chapter 4, while the 3D aspects are covered in Chapter 5.

Instance Segmentation algorithms typically rely on region proposals in the form of bounding boxes to in the process of discerning the different object instances. However, bounding boxes provide less useful information in scenarios involving high occlusion, such as overlapping chromosomes, very thin structures like fault planes, or structures with poorly defined boundaries, such as individual fault segments. Instead, I explore the use of orientation as the key factor for distinguishing individual instances. The approach leverages the fact two lines or planes must have different orientations to intersect, and that spatially disconnected regions cannot belong to the same instance. The algorithms I developed operate as post-processing steps, built on top of the predictions provided by the deep learning model, and rely solely on spatially local operations. This design allows for easy parallelisation on GPUs and facilitates the tiling of large images or volumes for analysis.

The first application explored in this thesis focuses on karyotyping chromosomes. My work in this area was published in the ICANN conference in a paper and presentation titled "Using Orientation to Distinguish Overlapping Chromosomes" [7]. In karyotyping, many images of chromosomes from various cells are taken in search of one where none of the chromosomes overlap. This process could be streamlined with tools capable of accurately segmenting even overlapping chromosomes. In this thesis, I explore the use of Deep Learning models for chromosome instance segmentation,

crucially examining an existing approach in the field and its associated dataset. I further elaborate on their methods and expand their dataset to further explore the limitations of the approach. This allows for a comprehensive comparison with my own method, which leverages predicted chromosome orientation and an orientation-based instance segmentation algorithm. Additionally, this application serves as a test bed for evaluating the effectiveness of various 2D orientation representations.

The second application, which contributes more substantially to this thesis, focuses on identifying and separating faults in geological seismic tomographic images of the Earth's subsurface. This work resulted in a publication in IEEE TGRS titled "Negation Invariant Representations of 3-D Vectors for Deep Learning Models Applied to Fault Geometry Mapping in 3-D Seismic Reflection Data" [5]. Interpreting seismic images and especially volumes is a challenging and labor-intensive task. Recently, Deep Learning has made significant progress in developing tools that can aid in this process, including the identification of faults locations through semantic segmentation. However, effectively visualising the resulting fault mesh can be difficult due to occlusion. Instance segmentation, on the other hand, allows for the selective visualisation of different fault segments. In this thesis, I explore deep learning model architectures designed to predict both faults and their orientations, while also using these models to evaluate the effectiveness of orientation representations in 3D. These models are trained and quantitatively evaluated on a synthetic dataset of seismic volumes. They are then deployed and qualitatively evaluated on a real seismic survey. The findings are compared against existing interpretations of the faults from the scientific literature on this survey.

1.1 Thesis Structure

This thesis is divided into chapters based on research topics, rather than application domains and published papers. The work on chromosomes and on seismic interpretation is distributed throughout the chapters of the thesis. Chapters 1, 2, and 3 serve as foundational chapters that introduce the background and concepts used throughout the rest of the thesis. Chapters 4, 5, and 6 are the content chapters that

consist of my research contributions, including methods, experiments, and results. Finally, Chapter 7 is the discussion which summarises and concludes the thesis.

The following is a brief overview of the chapters in this thesis:

Chapter 1 - Introduction: The remainder of this chapter will introduce some of the topics in this thesis in more detail.

Chapter 2 - Background: This chapter covers the principles of Deep Learning used throughout the thesis, as well as relevant model architectures. It also covers the relevant background about seismic imaging and relevant related work therein.

Chapter 3 - Datasets: The various datasets and their associated tasks that are used throughout the thesis are described in this chapter.

Chapter 4 - 2D Orientation Representation: This chapter focuses on the methods for representing orientation in 2D, as well as relevant experiments that empirically verify the representations. The experiments mainly depend on the chromosome datasets and tasks.

Chapter 5 - 3D Orientation Representation: This chapter contains my biggest scientific contribution and the 4 representations of orientation in 3D. It explains the theory behind the representations, as well as the experiments on the toy dataset and synthetic seismic data.

Chapter 6 - Instance Segmentation: This chapter looks at the deployment of models that were trained on synthetic data to predict orientation. On the chromosome task, it covers the issues with existing approaches and proposes the post-processing algorithm that is based on orientation. For seismic interpretation, the chapter covers the post-processing algorithm for separating fault segments, as well as an examination of the performance of the model on the real seismic data on the Laminaria volume.

Chapter 7 - Discussion and Conclusion: The thesis is concluded with a summary of my work, a discussion of my contributions, its limitations, other work that did not make it into this thesis, and directions for further research.

1.2 Seismic Interpretation

Seismic imaging, also known as tomography, is a powerful geophysical method that is used to create images of subsurface structures and geological features by analysing reflected and refracted seismic waves. These waves are generated by sources such as vibroseis trucks or explosives and recorded by a network of geophones. Subsurface features, such as rock layers and fluid reservoirs, can reflect seismic waves, creating distinct patterns or horizons in the seismic data. These reflections can then be interpreted by geologists to uncover the structure and properties of the sub-surface.

In the past, seismic imaging was usually performed using 2D images, which provided limited information about subsurface structures. However, with the advent of 3D seismic imaging in the 1980s, seismic surveys now create detailed volumes of subsurface data. Modern seismic surveys can cover thousands of square kilometres, and the collected data can require terabytes of storage. The resolution of seismic surveys can range from several metres to hundreds of metres, depending on the size and complexity of the survey, as well as the type of features that are being imaged. Seismic images are collected by moving the source of the seismic waves and the series of geophones listening for the reflections. Moving them in a line will result in 2D seismic images, whereas movement in a grid-like pattern yields 3D seismic volumes.

Interpreting seismic images can be a difficult and complex task, even for experts in the field. Seismic data is often noisy and incomplete, and the subsurface structures that are imaged can be complex and difficult to resolve. As a result, there is often a significant amount of uncertainty in the interpretations of the seismic images by experts. This uncertainty can arise from a variety of sources, including limitations in the data acquisition process, errors in data processing, and ambiguities in the interpretation of subsurface structures. This leaves a level of uncertainty in the interpretations of the seismic surveys, which can be both qualitative and quantita-

tive. Examples of qualitative uncertainty are the precise location of the identified faults and the detail of the resolution to which the locations are annotated. Qualitative uncertainty presents itself as uncertainty about what structures are observed in the first place and what geological processes caused them in the past. This uncertainty leads to questions about whether an observed structure is a fault in the first place, what type of fault it is, which fault happened first and what other geological processes happened to explain what we observe in the seismic surveys. Section 2.2.1 contains a more detailed examination of uncertainty in seismic interpretation.

The high level of uncertainty in the experts' interpretations highlights the difficulty of the task. Furthermore, with ever-increasing size and resolution of seismic surveys, more time and effort is required to produce seismic interpretations with adequate detail and precision. The only way this process remains feasible is thanks to technological advances that aid with seismic interpretations. Not only are computers and software used to visualise the 3D seismic volumes, but certain structures and features can automatically be highlighted within the seismic volumes using filters, machine learning, and Deep Learning methods.

Due to the difficulty of the tasks, automated interpretation techniques cannot be fully trusted. They serve as a tool for interpreters to use, performing a first-pass to bring attention to potential features, and to aid with the precision of drawing in boundaries of the features in 3D. Deep Learning promises to improve the performance of such tools to the point where they can be trusted, however, it faces some difficult challenges. Deep Learning is data-driven, requiring a lot of data to train from. However, any uncertainty and inaccuracy in the data on which the models are trained will boil down into the performance of the models. In addition, there are very few annotated and interpreted datasets to train from, and available interpretations usually selectively focus on features of interest, making the data incomplete. Using synthetic data addresses these issues, however, it brings challenges of its own.

By modelling geological processes, we can create synthetic subsurface structures and their corresponding synthetic seismic volumes. The advantage of such data is that we have perfectly accurate ground truth labels about what features are present and their exact locations. However, we are limited by how realistic the modelled

features can be. In practise, real data is much more complex than synthetic data that I had access to.

My work focuses on the interpretation of faults in seismic volumes. Faults are fractures in the earth's crust where movement has occurred. They are formed when tectonic plates move and shift, causing stress to build up within the Earth's crust. When this stress becomes too great, it is released in the form of an earthquake and causes the displacement of rock layers on either side of a fault plane. Therefore, the fault plane appears as a discontinuity or break in the subsurface layers or horizons in the seismic images. In oil and gas exploration, faults can act as barriers to the flow of hydrocarbons, and knowing their location can help identify potential reservoirs. Furthermore, faults can also be associated with geological hazards, such as earthquakes and landslides, so predicting their location is important to assess and mitigate these risks.

1.3 Seismic Data

The primary challenge that limits the capabilities of deep learning based seismic interpretation is the lack of available data. To train deep learning models, we must have annotations that are used as ground truth labels, and they must be in a standardised form that the model can predict. There is an abundance of unlabelled real seismic surveys, however, it is incredibly difficult to collect annotations that can be used to train deep learning models on the data. Seismic surveys and their interpretations are generally qualitative, with descriptions and diagrams explaining the processes that occurred. Figures displaying the locations of faults are often made appropriate for human readability and understanding, rather than complete annotations that could be used for training models. Even if the geologists made annotations or simulations as part of their study that would be appropriate for training deep learning models, they are not commonly published. Moreover, geologists commonly focus on very specific regions or features within their interpretations, which makes it difficult to create fully annotated volumes.

Furthermore, the number of faults in a volume can be very large and varying

in clarity, with both primary and secondary faults, large and small faults, or faults in areas with a lot of noise in the images, such as at higher depths, that are more difficult to determine. As such, it is very difficult to set a standard for what faults to annotate and what to ignore, however, for training deep learning models, we would need all the data to adhere to the same standard. Also, we would want all of the faults in a volume to be annotated, rather than just a subset of faults of interest. However, it is very time consuming to annotate faults in 3D. Even if geologists were to annotate the faults, there is a lot of uncertainty in the annotations. Due to all of these issues, there are no reliable annotations of seismic images that we can use to train the models. Instead, the use of synthetic data to train models has proven to be an effective alternative. In fact, I have heard of reports from within industry where they tried to gather a private dataset of geologists' annotations, which only led to poorer results than when training on synthetic data.

I was kindly provided with synthetic data from my industry sponsor, GeoTeric. The dataset consists of small volumes with up to three faults present in each volume. There are three major differences between the synthetic data and the real data: First, the synthetic data looks very clean. It does not have the same kinds of noise present that we would expect in a real seismic survey, especially as the depth increases. Second, the faults modelled in the synthetic data are completely flat, with no curvature. Fault geometries present in the real world can be much more complex. Third, the faults modelled are all fully independent. There are no primary and secondary faults. The fault networks in real life are much more complex and varied. Section 3.2.3 gives a more detailed description of the data.

The advantages of the synthetic data is the accuracy of the labels and the abundance of it. We know for certain that each label is correct, that every fault in the volume is labelled. We can also create as much synthetic data as necessary, although there is only so much fundamental variety within the synthetic data. The synthetic data perfectly encapsulates the principle of what faults are and what they look like within a seismic image. This is especially true at local scale, if we don't take the fault geometries into account.

Convolutional deep learning models proved to be very effective in learning the

relevant information from synthetic data in a way that translates to real seismic data. Convolutional networks are explained in detail in Section 2.1.4, but fundamentally they perform calculations in a pixel (or voxel in 3D) space, where the output for any pixel is conditioned only on the local neighbourhood of pixels in the previous layer of the model. This also achieves a degree of translational invariance, where the model's output is not affected if the image is translated/moved. This makes the architecture very efficient at interpreting vision. At the same time, this bias toward local information is a major downside of convolutional models, where they struggle to take long distance relations in the images into account. The attention layers in transformer-based architectures gives models the ability to focus its attention on any part of the image without the bias towards the local neighbourhood, which has made them a popular architecture in the recent years, albeit a very computationally expensive architecture.

Due to the nature of the synthetic data, the downside of the convolutions is actually an advantage. The synthetic data does not model the fault geometry well. We therefore do not want the model to take the broader fault geometry into account, rather we want it to focus on the local features that define a fault, i.e. the discontinuity in the seismic reflections in the seismic image. Additionally, the efficiency of convolutional networks is very important, since we are working with 3D data, which very quickly becomes very memory intensive within deep learning models. These are also reasons why I did not focus on transformer-based architectures, in addition to reports from industry that confirmed these theoretical suspicions.

This necessity to focus on local features of the synthetic images is also problematic for the self-supervised learning approach discussed in Section 7.3, where we ask the model to regenerate the input image by filling out sections within it that we masked. This forces the model to look at the geometry of the fault to extrapolate where it is most likely to continue. By training on the synthetic data, the model gets better at identifying the faults in the synthetic images, but struggles to generalise to the real seismic volumes that have much more complex fault geometries.

1.4 Network Output Representations

A fundamental challenge present in this field is the ambiguity in the outputs. There are many types of concepts that we might want models to predict that rely on the principle of consistency rather than specific labels. For instance, we might want to identify individual faults in an image, where we do not need to label each fault specifically with its unique ID, but only need to distinguish that it is distinct from another. We could start labeling the faults as the first, second, third, etc., but it does not matter specifically which fault gets the label of "first." This type of ambiguity requires careful consideration when designing neural networks to ensure that they can consistently differentiate between distinct instances without relying on arbitrary labels.

Neural networks are deterministic in nature. Given the same inputs, they will predict the same outputs. As such, they predict fixed outputs for any given input, making it difficult for the models to work with concepts such as ambiguous outputs, where multiple outputs might be correct, such as labelling the faults as (first, second) or (second, first). To address issues like this, clever outputs must be selected that avoid ambiguity and are unique, while providing the relevant information. For example, bounding boxes can be used for object detection, where each instance of an object in an image is labelled with a bounding box. If two bounding boxes overlap too much we assume that they represent the same object and select the one with the higher confidence score.

A substantial part of my work with chromosome separation consists of the criticism of existing techniques and publications that encounter this precise problem of ambiguous outputs. The models in question perform a semantic segmentation with class labels of "the first chromosome" and "the second chromosome" in the image, however, their labels could also be switched.

Another important principle in the design of the output representations is the continuity of the representations. Neural networks are continuous, meaning that a small change in the input can only ever create a small change in the output. The models are non-linear, meaning that the changes to the output could be disproportionately large in comparison to the changes in the input, but they are fundamentally

continuous. As such, it is important to make output representations continuous in a way, where changing the property they are representing by a small amount will not suddenly change the representation's value. This is surprisingly difficult for the 3D representations of orientation.

Since orientation is similar to a direction vector, only with negation invariance ($-\mathbf{v} = \mathbf{v}$), the choice of the $+/-$ sign complicates things. As we smoothly vary the orientation, after moving by 180° we arrive at the same orientation, while our vector points in the opposite direction. To achieve a unique representation for any given orientation, we must at some point switch the sign of the vector and multiply it by -1. This introduces a point of discontinuity that is problematic for the models. Instead, finding a clever representation that does not suffer from such discontinuities is the subject of Chapter 4 for 2D orientation, and Chapter 5 for 3D.

1.5 Motivation

This thesis explore the application of Deep Learning in two distinct domains: chromosome segmentation and fault identification in seismic images/volumes. The key question remains: Why is it relevant to investigate Deep Learning approaches in these fields?

Chromosome segmentation, while comparatively simpler, still presents challenges, especially when dealing with overlapping chromosomes. The task involves analyzing 2D images where chromosomes are clearly visible and exhibit high contrast against the background, making it more approachable with classical image processing techniques. However, segmenting overlapping chromosomes is more complex and remains an open problem despite the existence of classical image processing methods in scientific literature. This justifies the exploration of Deep Learning as a means to advance performance in this area. Moreover, existing Deep Learning approaches have shown promise, yet the problem remains unsolved, highlighting the need for further investigation. This topic is further explored in Section 6.1. Additionally, karyotyping chromosomes serves as an effective test bed for developing and evaluating 2D orientation representations for neural networks. These representations are

interesting in their own right and have the potential for broader application across various fields.

Fault identification in seismic images is an exceptionally challenging task. Geologists often disagree on whether faults are present in specific locations, and the process of interpreting and precisely annotating faults is labor-intensive. Various tools and filters exist to enhance the visualization of features in seismic images, aiding geologists in identifying areas of interest. These are further discussed in Section 2.2.2. Deep Learning has made significant strides in this field, with numerous studies focusing on the automatic identification of faults and their locations. This research has seen industry adoption, with tools integrated into visualization software packages and offered as on-demand services, such as those provided by GeoTeric. The demand for better tools is matched by advances in Deep Learning algorithms, driving active research in this area. However, performing Instance Segmentation on faults is a relatively unexplored area, despite its clear applications in fault visualization. Furthermore, predicting the orientation of faults offers valuable information for analysis, motivating and inspiring the research presented in this thesis.

1.6 Research Questions and Findings

The following are the three primary research question that this thesis answers, together with a summary of the relevant findings:

How can Deep Learning models be used to predict orientation? I propose four novel representations of orientation in 3D, all of which are effective, with the Projection-Doubleangle representation performing the best in my experiments. The Doubleangle representation can be used for 2D tasks.

Can predicting orientation on top of a semantic segmentation be used for better performing instance segmentation of overlapping chromosomes? Yes, I demonstrated how orientation can be predicted using a deep learning model for chromosomes and developed an algorithm that further separates the overlapping chromosome instances.

Can prediction orientation on top of a semantic segmentation be used for better performing instance segmentation of fault segments in seismic geological surveys? Yes, I demonstrated how the orientation of faults can be predicted using my proposed 3D orientation representations using a deep learning model and explored various model architectures to do so. I developed an algorithm that separates fault into fault segments based on their predicted orientation and location, which can be individually visualised. The approach can be deployed effectively to large volumes and parallelised on GPUs.

CHAPTER 2

Background

2.1 Deep Learning

Deep Learning is a field of Machine Learning, where we train deep artificial neural networks to find patterns in large amounts of data. Artificial Neural Networks are inspired by our own brains, mimicking the function of neurons and their synapses. A biological neuron accumulates membrane potential on the synapses of its dendrites until a threshold is reached and an action potential is triggered. This will cause the neuron to release neurotransmitters in the synapses on its axon, which will in turn stimulate the dendrites of the postsynaptic neurons. Depending on the type of neurotransmitter used in a synapse, this will have an excitatory or inhibitory effect on the postsynaptic neuron. After a neuron has fired, it must recover its ions, making it harder to fire again in quick succession. The rate of the neuron's activation is what we model in artificial neural networks, which makes the calculations computationally simpler, as opposed to simulating individual neuron firings. We model the neuron's behaviour (firing rate) as a a sum of all the contributions on its dendrites' synapses, each of which is a weight (positive for excitation or negative inhibition) multiplied by the firing rate of the presynaptic neuron connected to it. Additionally, we also

have a bias term added to the sum. Overall, this operation is a weighted sum of the input neurons' firing rates plus a bias expressed as: $\mathbf{x} \cdot \mathbf{w} + b$, where $\mathbf{x}$ is the input activation, $\mathbf{w}$ the weight and b the bias. Finally, we must apply an activation function to this weighted sum to transform it into the firing rate value. This operation also leads to the terminology of pre-activation for the weighted sum and post-activation or activation for the value returned by the activation function. Historically, to model the biology, the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ was popular as an activation function, however, the Rectified Linear Unit (ReLU) $f(x) = \max(0, x)$ and its successors, such as GeLU, have become more common since. This behaviour of a single artificial neuron is also shown in Figure 2.1.

To create a neural network, we connect multiple neurons together. Neurons are arranged in layers, with the simplest architecture connecting every neuron in one layer to every neuron in the next, forming a so called fully connected layer or dense layer. The operation for such a layer with n inputs and m neurons is $\mathit{activation}(W \cdot x + b)$, where W is matrix of weights of shape $m \times n$, x the input vector of shape n, and b the bias vector of shape m. Refer to Figure 2.2 for a diagram. We would refer to m as the width of the layer.

To create a deep neural network, we connect multiple layers of neurons together, with the number of layers being called the depth of the network. See Figure 2.3 for an example. Provided that the layers use a non-linear activation function, deeper networks can model increasingly complex functions. However, making networks deep also causes issues with learning. Most importantly, the deeper the network the more the learning signal gets diluted as it backpropagates through the network and the early layers become prone to the vanishing gradient problem. Many breakthroughs in Deep Learning help to address this issue, at least in part amongst having other benefits. Examples include the use of ReLU as the activation function [8], the use of batch normalisation [1], specialised architectures such as convolutional neural networks [9], skip connections such as in a ResNet [10] and others. Let us first have a look at backpropagation and how we train neural networks, before looking at some of the aforementioned techniques in more detail.

Most commonly, neural networks are trained in a supervised learning context,

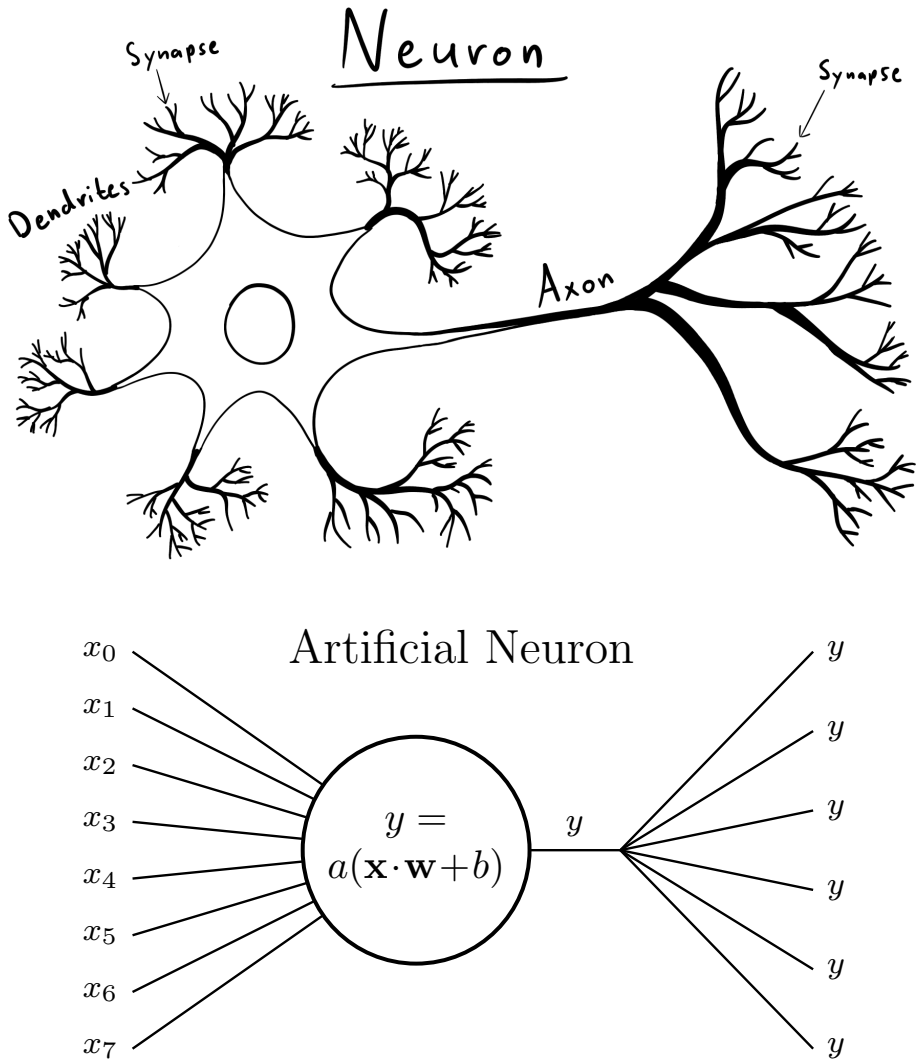


Figure 2.1: A diagram of a biological neuron and its artificial counterpart. The biological neuron works by firing at different intervals depending on the accumulated signals on its dendrites, which is a binary signal, while the artificial neuron models something akin to the rate of firing, a floating point value. Note that every synapse on a dendrite has a weight associated with it. In the artificial neuron, $\mathbf{x}$ is the input vector and y the output of the neuron, $\mathbf{w}$ the vector of weights, b the bias and a the activation function.

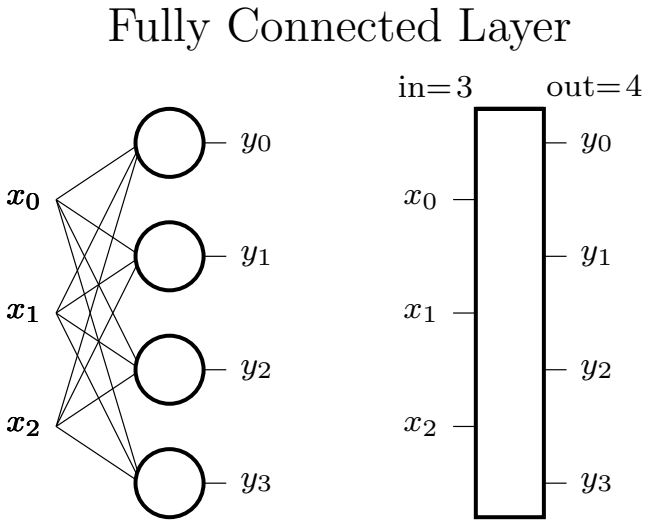


Figure 2.2: A diagram of a fully connected layer in an artificial neural network, drawn in two different ways. The layer has an input width of 3 and an output width of 4.

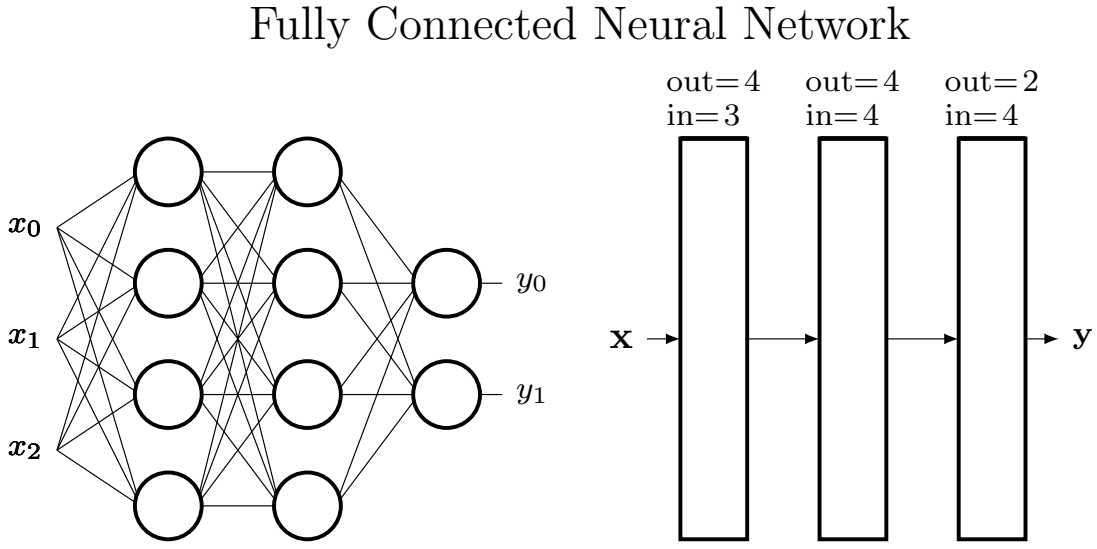


Figure 2.3: A diagram of a fully connected neural network, drawn in two different ways. The network is 3 layers deep, has an input width of 3, hidden widths of 4, 4 and an output width of 2 features.

where we train them on a dataset of input and label pairs. The model is deployed on an input making a prediction. This prediction is compared to the label using a loss function that calculates the loss or error. The network's weights are then updated in a way that decreases this loss on this particular data example. This is done by calculating the derivative of the loss with respect to every weight in the network through a process called backpropagation [11]. We then use gradient descent based optimisation algorithms to update the weights in the network. This process is repeated with different data examples from the dataset, hoping that the model will generalise to all the data in the dataset and domain at large.

When performing gradient descent, each step is conditioned on the data example that the gradients were calculated from. These gradients could point in opposing direction for different data. It is beneficial to average the gradients over multiple data points and therefore reducing the variation in the gradient directions, which also allows us to perform model updates in the direction of the resulting gradient. We call the number of data points fed through the network for a single update the batch size of the corresponding batch of data.

When evaluating a model's performance, we must be careful with the data that the metrics are gathered from. Commonly, a dataset is split into three subsets: training, validation, and testing subsets. The training set is used to update the model's parameters (using gradient descent). The validation and testing sets are strictly used for evaluation only, so that we can be certain that the model generalises to the task and these unseen datasets, rather than, for example, merely memorising the specific examples in the training set (overfitting). The validation set is used to make decisions about the training strategy, the model used, or any other hyperparameters. Lastly, we use the testing set to evaluate the final model performance, which must not affect any decisions that we make.

When evaluating the performance of a model, there are two commonly discerned cases, called underfitting and overfitting. Underfitting occurs when the model is unable to represent the data. It manifests itself as low performance in both the validation and training sets. This is often a symptom of error in the training algorithms used, or an indication that the model is too small and simple for the task at hand.

Overfitting occurs when the model fails to generalise by modelling the training data too closely, such as memorising the examples in the training data. It manifests itself as a decrease in validation performance while training performance is increasing. Overfitting can be addressed by increasing the size of the training dataset, simplifying the model, early stopping, or a variety of regularisation techniques.

With the basics of neural networks explained, we can now move to more specific network components and architectures that are relevant to my design of neural networks in this thesis.

2.1.1 Normalisation

When providing inputs into a neural network, it is important that the values are normalised [12]. This is especially true when feeding values on different scales to the network. If a network is fed an input feature close to 1×10^{-3} and another close to 1×10^{3}, then given a random initialisation of weights in the network, the latter feature will contribute much more to the prediction, and subsequently affect the learning process of the model undesirably. Commonly, values in a dataset are normalised to the standard normal distribution, with a mean of 0 and a standard deviation of 1. Additionally, many properties of neural networks are studied on the basis of normalised values. For example, optimisation algorithms and their default hyperparameters are often chosen to work best on normalised data.

A big breakthrough in deep learning came with the introduction of Batch Normalisation [1], which normalises hidden activation values within the neural network for better optimisation and learning throughout the network, not just at its inputs. This is because even when we start with normalised data, its distribution can change as the data moves through the network. Batch Normalisation looks at all the hidden activations in a specific layer and across the whole batch dimension, and normalises them to a target distribution. The layer gathers statistics of how it normalises the data during training, which it then uses on deployment even with examples that do not have batches.

There are also alternatives to batch normalisation. Notably, for use on image datasets with Convolutional Networks, Instance Normalisation [2], Layer Normali-

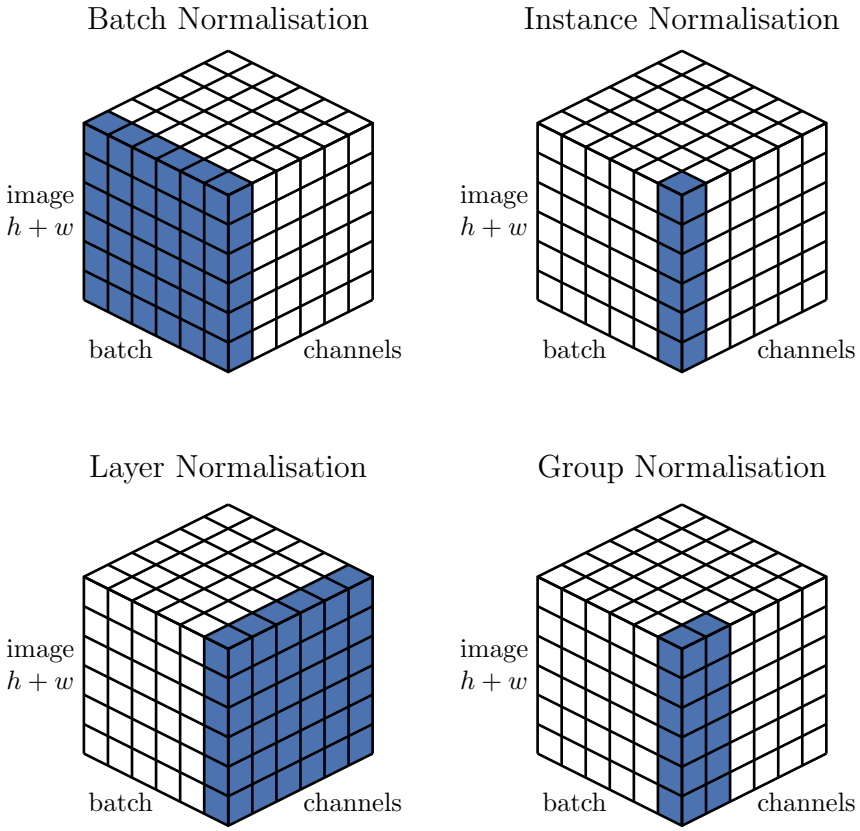


Figure 2.4: Visualisation of different normalisation techniques in convolutional neural networks: Batch Normalisation [1], Instance Normalisation [2], Layer Normalisation [3], Group Normalisation [4]. The area in blue shows over what part of the tensor the normalisation is calculated. The whole tensor is normalised in chunks of the same shape as the one shown in blue.

sation [3], and Group Normalisation [4] are common candidates. Their functionality is visualised in Figure 2.4, where they normalise over different parts of the tensor.

2.1.2 Regularisation

Regularisation techniques can be used to fight overfitting. One of the simplest techniques is early stopping, where training of the model is stopped after validation performance stops improving and the model that achieved the best validation performance during training is selected.

Increasing the dataset size is perhaps one of the best ways to fight overfitting. Overfitting implies that the model is capable of modelling more complex relations than is appropriate for the dataset. Since it is not always easy to get more data, data

augmentation techniques can be used to achieve a similar purpose. For example, mirroring an image can, in certain contexts, yield an equivalently valid datapoint. Crucially, we must never treat different augmentations of the same datapoint as independent. They must always belong to the same training/validation/testing subset. This is a principle I explore further in Section 6.1.1.

Dropout [13] is another very effective and popular technique. During training, it erases random connections in the network, temporarily setting their weights to zero and not updating them. This encourages the network to distribute its computation and to not rely on specific connection. This increase in independence between neurons helps to fight overfitting.

Batch Normalisation can also be considered as a regularisation technique, since its normalisation has the effect of reducing the network's dependence on specific weights and activations that would otherwise have higher values.

Weight decay, also known as L1 or L2 normalisation is a technique, where the magnitude of the weights is added as a penalty to the loss function. This encourages the gradient descent updates to reduce the magnitude of the weights. L1 regularisation uses the absolute values as the penalty, while L2 regularisation uses the squares of the weights. L1 normalisation commonly has the effect of reducing some weights to zero, which can then be used to prune the network. L2 normalisation generally encourages all weights to be smaller.

2.1.3 Tasks and Loss Functions

The choice of loss function is inherently tied to the task that the model is trained on. Let us look at the commonly used loss functions for regression and classification tasks in a supervised learning context. Note that the activation function used in the last layer of the network is considered as part of the loss function in these definitions.

When performing a regression, we get the network to predict a continuous output value and compare it to the value in the label. Depending on the range of the predicted values, we might want to use a sigmoid activation function, which will bound the output between 0 and 1, however, it will never be able to reach either extreme. It may make sense when predicting probability values or other principles

that asymptotically approach 0 and 1. Similarly, the tanh function can be used when the bounds are -1 and 1. A linear activation may be the most appropriate for many variables that are expected to scale linearly. Similarly, we could apply the exponential function or a log as the activation if we expect the property to scale appropriately.

Next, let us talk about the different loss functions for a regression. The most common is the mean squared error (MSE), defined as follows in equation 2.1:

$$\text{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - y_i)^2, \tag{2.1}$$

where $\mathbf{x}$ is the prediction, $\mathbf{y}$ the label, and n the number of data points compared. MSE has the effect of heavily penalising outliers, while only slightly penalising near misses. This has the effect of creating blurry predictions, akin to Gaussian noise, where values are rarely perfectly accurate but often in the general vicinity.

Mean absolute error (MAE), a.k.a. L1 loss, is defined as follows in equation 2.2:

$$\text{MAE}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=0}^{n-1} |x_i - y_i|, \tag{2.2}$$

where $\mathbf{x}$ is the prediction, $\mathbf{y}$ the label, and n the number of data points compared. Unlike MSE, MAE does not penalise outliers harshly and allows occasional big errors more readily than MSE.

Next, let us move onto classification tasks. These are single-class classification tasks and multi-class classification tasks. In single-class classification the probabilities across all classes must add up to 1. This allows us to use the softmax activation function defined as follows in equation 2.3:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=0}^{n-1} e^{x_j}}. \tag{2.3}$$

For a multi-class classification, any example can belong to multiple classes. We therefore have to calculate the probability for each class independently. We therefore use the sigmoid activation function to predict the probability for each class

asymptotically bound between 0 and 1. Sigmoid is defined as follows in equation 2.4:

$$\text{sigmoid}(\mathbf{x})_i = \frac{1}{1 + e^{-x_i}}. \tag{2.4}$$

Let us talk about the loss functions used for classification. Most commonly, we use a Negative Log Loss (NLL), which in conjunction with the various activation functions gives rise to different variants of the Cross Entropy Loss (CE). NLL is defined as follows in equation 2.5:

$$\text{NLL}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=0}^{n-1} \begin{cases} -\ln(x_i) & \text{if } y = 1 \\ -\ln(1 - x_i) & \text{if } y = 0 \end{cases}, \tag{2.5}$$

which is visualised in Figure 2.5. This loss is large when the label is misclassified with a large confidence, while levelling out the close we get to the correct prediction. Note the use of the negative sign, which makes the values of the log positive, with lower values that are closer to zero being desirable. This makes it appropriate as a loss or error function that we try to minimise.

We can equivalently express NLL without using a piecewise equation as follows in equation 2.6. This form is also commonly referred to as the Binary Cross Entropy Loss:

$$\text{NLL}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=0}^{n-1} -[y_i \cdot \ln(x_i) + (1 - y_i) \cdot \ln(1 - x_i)]. \tag{2.6}$$

When expressed in this form, NLL can also be calculated for labels that are not only $y = 0$ or $y = 1$, but any continuous value between 0 and 1. In this way, a weighted average is calculated between the two losses such that the lowest loss is found at $x = y$. Note that the loss will never be zero in these cases, even when $x = y$, which is why the loss is not used with such values frequently. However, I found it to be effective in my experiments. The loss is visualised for a selection of target values in Figure 2.6. The types of problems that use such labels are known as classification tasks with continuous labels, fuzzy labels, or partial class assignment. Despite being

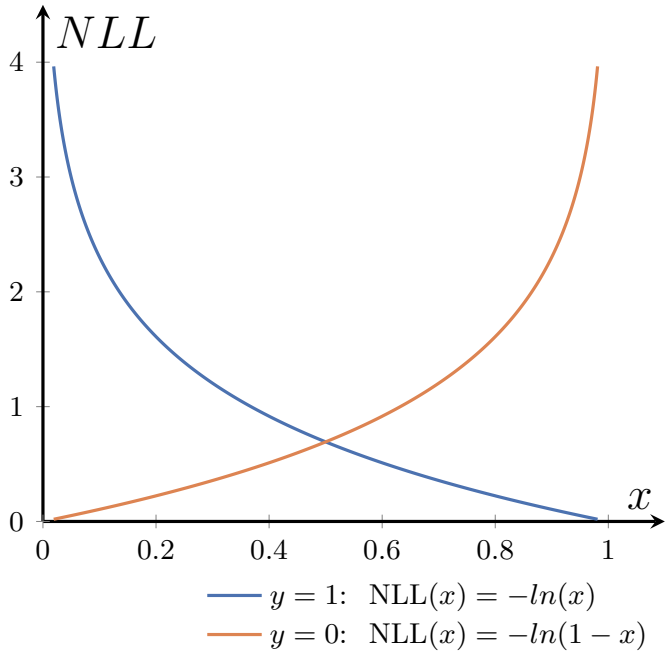


Figure 2.5: Graph showing the curve of the negative log loss. The blue curve shows the loss if the label $y = 1$ and the orange curve shows the loss if $y = 0$.

classification tasks, these models often use the MSE or MAE losses.

2.1.4 Convolutional Networks

There are numerous properties of images that we can exploit when designing Deep Learning models that work with image data. Most importantly, the pixels in an image are laid out spatially, with nearby pixels being more closely related to each other than pixels that are far away. Additionally, a degree of translation invariance is present in images, since shifting an image by a few pixels does not completely change the contents of the image. If we use fully connected networks on images, every pixel is compared to every other pixel with equal importance, losing any notion of local connectivity. Additionally, shifting the image would cause completely different weights to be used for each pixel, which could vastly change the prediction of the model.

Convolutional networks are built around the convolution operation, which is an operation that applies a stencil to the local neighbourhood in an image as seen in Figure 2.7. Each output pixel is calculated from the local neighbourhood of the

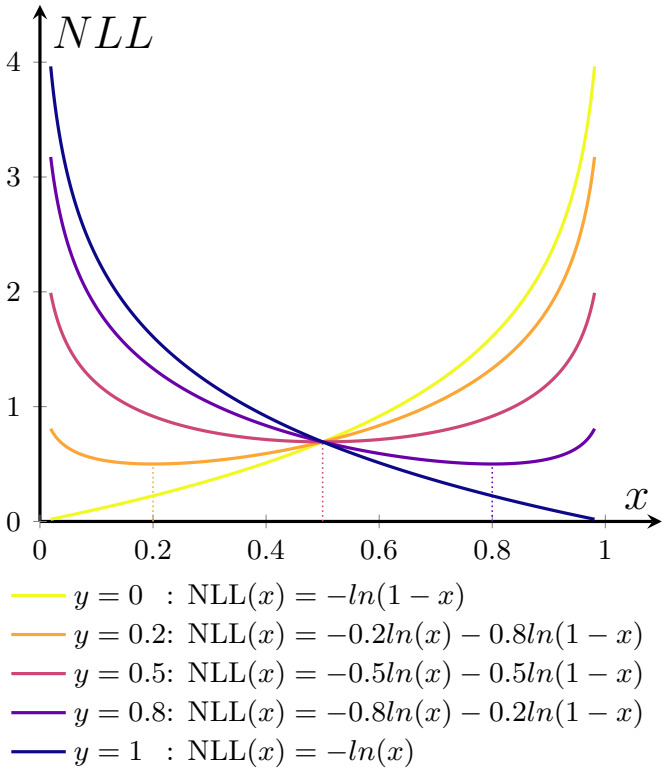


Figure 2.6: Graph showing the curve of the negative log loss, also known as binary cross entropy, for target values y that are not only $y = 0$ and $y = 1$.

respective pixel in the input image. Crucially, the same weight matrix called the kernel is used for each output pixel, with only the relative position between the input and output pixel being important. The size of the kernel, such as a 3×3 kernel, determines the receptive field of the operation, which is the area in the input that affected a particular output pixel.

Notably, the size of the output image is reduced with a convolution, because there aren't further pixels near the boundary to be multiplied by the kernel. This can be inconvenient. Instead, we could pad the image with additional values to keep the image sizes the same, most commonly zero values.

By itself, the convolution operation can only input and output a single greyscale image, i.e. an image with 1 channel. If we wanted to apply a convolution to an RGB image with 3 channels, we could apply a separate convolution to each image and sum their results (before applying the activation function). Furthermore, if we wanted to output more than 1 channel, we could repeat the process multiple times with different kernels. Together we call this a convolutional layer. This operation

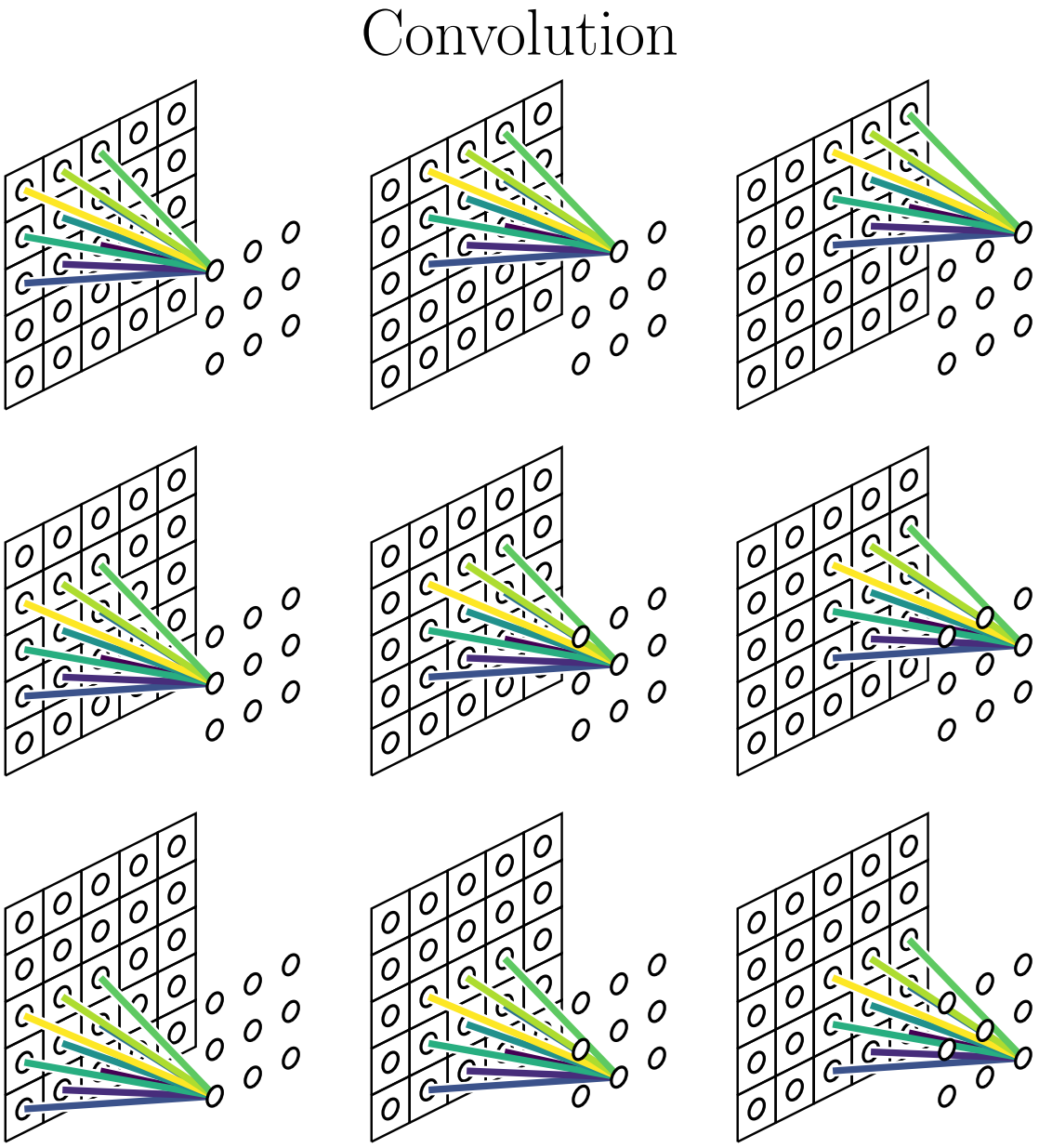


Figure 2.7: A diagram of a 3x3 convolution operation inside a convolutional network layer. With a 3x3 kernel, each output neuron is conditioned on the inputs in a local neighbourhood of size 3x3. The weights used for each neuron are the same, i.e. the blue and the orange lines have the same respective weights, which is a 3x3 matrix of weights known as the kernel.

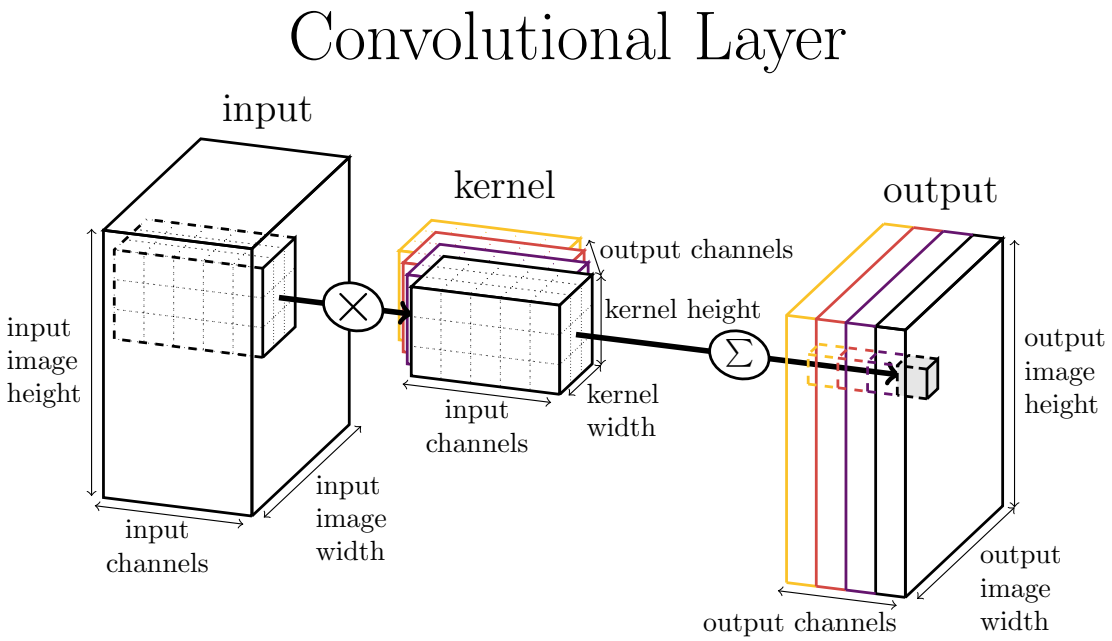


Figure 2.8: Visualisation of a convolutional layer with a 3×3 kernel.

if visualised in Figure 2.8, when expressed in terms of multi-dimensional tensor operations. Note that this convolutional layer behaves like a fully connected layer within the "input channels" and "output channels" dimensions, where each output channel is connected to each input channel.

There are two additional properties of a convolutional layer that can be modified. The first is the stride. This refers to how many pixels we move the kernel across the image. This value can be specified independently for each axis, height and width. With a stride bigger than one, the resolution of the output image will be lower than that of the input image. The second property is the dilation of the convolutional kernel, also known as an atrous convolution. The dilation specifies how spread out the pixels are in the local neighbourhood of the input image that are multiplied with the kernel. With a dilation bigger than one, the pixels will not be adjacent, but rather spread out by the dilation factor.

Convolutional layers can be applied in sequence, commonly with normalisation layers between them, to form deep convolutional networks. An important principle to keep in mind when designing convolutional networks is the receptive field of the whole network. It tells us the size of the local neighbourhood on the inputs that affected a single output pixel. Notably, with convolutional networks, pixels near the

middle of the receptive field tend to have a larger contributions than near the edge. To increase the receptive field of a convolutional network, it is common to use a series of downsampling and upsampling operations that change the resolution of the image, which is the basis behind the UNet architecture [14].

Convolutional networks are most commonly used with two types of tasks on images. Either predicting features for the whole image, or predicting features localised in space (on a per-pixel basis). The loss functions used are commonly the same, only the loss is calculated repeatedly for each output pixel in the latter case.

3D Convolutions

So far we talked about 2D Convolutional networks that work on images with height and width components. The same principles can be applied to 3D images with x, y and z components, by adding an extra spacial dimension to all the tensors. Examples of data that these 3D convolutional networks can be used on are MRI scans, or tomographic images. Notably, convolutional layers don't have many parameters compared to their fully connected counterparts. However, they have a big memory footprint of the images being passed through them. This becomes much worse in 3D convolutions than in 2D, putting different constraints on the 3D convolutional networks' design, such as their widths, depths and resolutions of volumes used.

2.1.5 Semantic Segmentation

Semantic Segmentation is an imaging task, where each pixel of the image is classified into a distinct semantic category. The goal is to identify objects within the image, as well as their precise boundaries. In the simplest form, if we have k available semantic classes, the model predicts an image with k channels. A softmax activation function is used across the channel dimension to get probabilities of the pixel belonging to each semantic class and the network is trained using a Log-Loss on top of the softmax layer. Together these are known as the Cross Entropy Loss.

Within Deep Learning, the field started with fully convolutional networks. The next significant improvement came with the use of down and up-sampling together with skip connections in the form of the UNet architecture [14]. The UNet was

originally designed for biomedical image segmentation, but has been widely adopted across various fields and its principles remained as a structure for many following architecture. Most of the model architectures I explore in this thesis fall within the UNet structure.

Improvements that followed are dilated or atrous convolutions. These can serve as an alternative method of capturing features at various resolutions and are used by the DeepLab family of models, such as DeepLabv3+ [15].

The principles of the style of skip-connections in a ResNet architecture [10] can also be used within architectures used for semantic segmentation. They allow the models to be much deeper than before, and proved to be an effective approach.

Attention is another principle that has been incorporated into segmentation models. It allows models to focus on relevant regions of an image, with architectures such as The Attention U-Net [16].

More recently, transformer-based architectures have become more popular for semantic segmentation. Architectures such as the Vision Transformer (ViT) [17] and the Swin Transformer [18] demonstrate that even for segmentation, attention is all you need. However, as it turns out, these attention based architectures are a double edged sword for the type of synthetic data that I use. It may exhasterbate the differences between the synthetic and real data and may not necessarily lead to improvements on the model's generalised performance on the real data. See Chapter 7 for a more detailed discussion.

2.1.6 Instance Segmentation

Instance segmentation is a task that is more complex than semantic segmentation. It similarly classifies pixels in an image, however, rather than just determining semantic categories, it also distinguishes between multiple instances of the same object.

Early approaches often relied on a combination of traditional computer vision and object detection methods. One of the pioneering methods was the Region-based Convolutional Neural Network (R-CNN) [19], which used a two-step process: first generating region proposals as bounding boxes and then classifying and segmenting each region independently. However, these models were computationally expensive

and required significant post-processing to refine the proposed segmentation.

The Mask R-CNN architecture [20] was a significant breakthrough in instance segmentation. It uses a small convolutional network to propose a binary mask for each instance's region proposal bounding box. This approach set the standard for instance segmentation and remains as one of the most widely used architectures in the field.

Following Mark R-CNN, various alterations and improvements have been proposed to enhance the accuracy, efficiency and flexibility of the model. One such advancement is the Path Aggregation Network (PANet) [21], which improves feature fusion across different levels of the feature pyramid and better captures multi-scale information.

Another notable development is in the YOLACT model (You Only Look At Coefficients) [22], which reduces computational complexity and allows for real-time instance segmentation. It does this by predicting a set of prototype masks independently from object instances. YOLACT only predicts a set of coefficients for each instance, which are linearly combined with the prototype masks, saving much computation.

The SOLO (Segmenting Object by Locations) [23] uses a fixed grid instead of predicting bounding boxes for each instances.

These Deep Learning techniques for instance segmentation fundamentally rely on the principle of object localisation to distinguish instances. Commonly, its region proposals take the form of bounding boxes, and are problematic for instances of objects that are highly occluded. However, it is even more problematic for very narrow objects that may have large bounding boxes, such as geological fault planes. A bounding box over a fault plane would not offer much information about it. This justifies my exploration of the use of orientation for separating the instances of overlapping chromosomes or faults.

2.2 Seismic Imaging

2.2.1 Uncertainty in Geoscience

One of the main aspects of Geoscience is the analysis of subsurface geology. In order to create geological framework models, experts in the field have to rely on limited information, as the subsurface cannot be accessed directly without digging into it, such as with tunnels or wells. One major tool at their disposal is seismic tomography, which is capable of providing information about large areas and depths of the subsurface at relatively high resolution. Nevertheless, seismic reflections are only an indirect way of observing the subsurface, the interpretation of which is a difficult and time-consuming task. In addition, linking seismic interpretations with actual geology is challenging, where drilling data must be used for calibration. Therefore, one cannot understate the difficulty and importance of the geologists' task in uncovering the geology. In fact, the entire discipline of geology has been recognised as a field in which uncertainty is the norm rather than a special case by Robert Frodeman [24], a science historian and philosopher.

Uncertainty can be broadly separated into two kinds: objective quantitative uncertainty, such as errors in readings and measurements, and subjective qualitative uncertainty, where human judgement and decisions are involved. Walker et al. [25] argue that individually addressing these types of uncertainty is important. Within geoscience objective uncertainty is generally addressed, however, subjective uncertainty only has a handful of studies. Moreover, this uncertainty is rarely even acknowledged within geoscience, with a lack of means to communicate such uncertainty, argues Clare Bond [26].

Bond et al. [27] set out to identify the extent of uncertainty in geological interpretations without relying on the subjective measure of one's confidence in a solution. They designed a synthetic seismic image by forward modelling. In this way, they knew the exact underlying geological processes that formed the final model and the seismic image. Then 412 geoscientists were assigned to interpret the image. Only 10% of the interpretations were correct, which increases to 21% partially correct interpretations. In other words, 79% of the participants failed to identify the correct

geological process and the most common interpretation was incorrect.

Bond et al. further analysed a subset of their results from [27] in [28], identifying the aspects that made experts effective at interpreting the seismic image. Out of the 445 geologists, they analysed the interpretations of 184 self-proclaimed experts in structural geology. Of this subset of geologists, 35% correctly identified the inversion, compared to the overall proportion of 21% from all participants. Out of all the explanatory variables, they found the following two to have the most significant influence on correctly identifying the inversion: having a Master's or a PhD, as well as making use of evolutionary and geometric feasibility. Interestingly, having a Master's or PhD degree mattered, whereas technical specialism, years of experience, and most known tectonic setting were insignificant. As far as interpretation techniques go, the 18 out of 184 participants who forward-modelled their interpretations to verify their feasibility were almost three times more likely to find the correct interpretation. Furthermore, the interpretation and annotation of horizons was also significant, although less important.

Rankey and Mitchell [29] devised an experiment with six interpreters who identified the same seismic volume. They were provided with a known lower horizon together with the data from two wells. Subsequently, six wells were given to them, together with literature on the geology of the reservoir. Following the additional information, four of the six made changes to their interpretation. Despite the task being described as easy by the interpreters, their interpretations differed in certain places, and they applied different amounts of smoothing. Quantitatively, their volumetric measurements differed by up to 24%.

Polson and Curtis [30] studied how the confidence of a team of four experts in the hydrocarbon industry changed during a group elicitation session. Before the session, the confidence of the experts varied significantly. Their opinions changed significantly throughout the meeting, despite not receiving any new information about the task. After finding a consensus, their confidence agreed much more. Nevertheless, the self-reported best-guess probability of being correct after the consensus was reported as low as 50%, with 85% for the location of a fault. However, the experts' confidence in this interpretation dropped after the meeting, where one expert

started disagreeing with the consensus from 10 minutes ago. This behaviour was later analysed by Andrew Curtis [31], where he warns that a consensus of multiple experts might not reflect the true probabilities of being correct.

Rowbotham et al. [32] make the case for interpreters to define multiple possible scenarios as opposed to predicting a single most likely interpretation with uncertainty bounds. They analyse the cognitive biases that people are prone to when interpreting and demonstrate them in action in three separate case studies. The problematic behaviours are as follows: The need to get the 'right' answer, being instinct driven, box ticking, focusing on the wrong thing, herding, overconfidence, anchoring on a best technical case, and the lack of understanding of impact.

These levels of uncertainty in geological interpretations show how difficult it is to interpret geology. This highlights a few important aspects that need to be addressed when trying to automate the interpretation process. First, the importance of interpretability in any automated solution. The model needs to be able to justify its interpretation so that geologists can verify its reasoning. Furthermore, outputting multiple feasible interpretations and their respective confidence would also be helpful. Second, with such a high amount of uncertainty, it is difficult to gather data on which a machine learning model can be trained effectively. A machine learning model can only be as good as the dataset on which it is trained. Relying on human interpretations of real data is dangerous, and augmenting the process with synthetic data formed by forward modelling might be appropriate, even though it poses additional challenges.

2.2.2 Automatic Interpretation

Computers have always been an integral tool in the analysis of seismic images, with technology becoming more important as the seismic surveys increased in size and resolution. Modern seismic surveys are now able to cover areas of more than 10000km^2 represented as terabytes of data. As a result software for interacting with this data has become important. These tools not only help visualise the the seismic data and provide interfaces for doing the interpretation, but also aid in the interpretation process. Broadly speaking, the tools for aiding with interpretation

can be separated into three categories: attributes, picking tools and automated interpretation. A review of such techniques was performed by Wang et al. [33].

Attributes are a group of techniques that highlight certain features in the seismic images. They can be thought of as visual processing filters. A good attribute is sensitive to geological features directly or it highlights the depositional or structural environment from which the geological features can be inferred. Seismic attributes are however still highly reliant on a person to perform the interpretation, they merely aid in the visualisation of the data. Chopra and Marfurt [34] performed a thorough analysis on the importance of seismic attributes from a historical perspective. For the purpose of fault detection, there are several commonly used attributes, such as coherence [35], curvature [36], similarity [37], entropy [38] or flexure [39].

The second category of tools are designed to help a person interact with the software and express their own interpretation. These tools often require the user to input some seed points, which the algorithm then connects, or grows out along the feature boundary. Examples of such tools for fault detection are ant tracking by Pedersen et al. [40], Cohen et al.'s 3D fault surfaces extraction algorithm [38], or GeoTeric's proprietary Adaptive Fault Tracking [41]. Pavlidis [42] proposed a method to extract one pixel wide fault surfaces from larger fault regions and Hale's method [43] constructs a surface of quadrilaterals to define a fault surface from quadrilaterals. Active Contour Algorithms [44] can also be used to create surfaces from fault likelihood data, or any other geobody likelyhood. These methods can be applied to seismic attributes as well in order to perform a fully automated interpretation, however these interpretations generally suffer from either a low precision or a low recall. A human interpreter is ultimately still needed to create the interpretation.

The emergence of Deep Neural Networks has significantly improved the ability of tools to perform seismic interpretations directly and detect faults [45]. Deep Learning is capable of discovering intricate patterns in large data sets by performing processing in multiple layers with an increasing level of abstraction with each layer [46]. This property of Deep Neural Networks has been compared to human intuition, however due to its high level of abstraction the models suffer from a lack

of interpretability [46]. They are black boxes. Nevertheless, Deep Learning models have revolutionised a number of fields including image processing. For the tasks of image recognition or object detection, Convolutional Neural Networks [47] have become the leading architecture [48]. These models have also made it into the field of seismic interpretation with a number of approaches that attempt to automate the interpretation process.

Convolutional Neural Networks work by repeatedly applying convolutional filters, which identify features of increasing complexity in an image. The first filters detect lines and curves, which feed into middle layers that detect more complex textures, until eventually features such as eyes or a nose can be detected in a facial recognition setting. The relative positions of the eyes and a nose could then be used to identify a person in the last layers of the network. A major advantage for image processing is that when compared to standard deep neural networks, convolutional networks are position invariant. This means that if a convolutional network learns to identify a face in one corner of the picture it will be able to identify it anywhere in the image. A standard neural network would treat different parts of the image independently and need to learn to identify the same features separately for every location in the image.

Convolutional Neural Networks have also made big impact within seismic interpretation. They have been successfully applied to task such as facies detection [49–53], salt body detection [54–57], inversion [58–61], horizon detection [62–65] or fault detection [54–57, 66–77]. This also demonstrates the versatility of convolutional networks and their ability to distinguish a wide variety of features given the correct dataset and approach. Despite their success, the aforementioned models are far from perfect and still require a lot of research to get anywhere close to fully automating the interpretation process. The general formulation of the task for all the aforementioned approaches is called semantic segmentation.

Semantic segmentation is defined as a task where for every pixel/voxel in the input the network will output a category that the pixel belongs to. Depending on the labels in the dataset the network will learn to identify different features, even with the same network architecture. The semantic segmentation can also be

expanded such that instead of categorising every pixel into distinct categories the network will output a set of parameters for each pixel. This allows a single network to identify multiple types of features simultaneously. The outputs of such a network can be treated as a custom filter that highlights certain features, similarly to any image processing filter. The output may then further be modified in post processing to extract meaningful information. In comparison to image processing filters or attributes applied to seismic images, the neural network is able to provide better results. If good enough, the network outputs could be treated as an automatic interpretation, instead of just another tool for a human interpreter to use.

2.2.3 Synthetic Data

One difficulty with applying neural networks to the tasks of seismic interpretation is that neural networks require large amounts of data to learn. This data has to be annotated in order to train the neural network in a supervised way. However, gathering large amounts of already interpreted seismic data is difficult. As described in the uncertainty Section 2.2.1, even experienced interpreters often disagree on the interpretations, especially when working with large amounts of 3D data that humans struggle to effectively work with. One of the advantages that algorithms have over humans is their ability to effectively work in any number of spacial dimensions. A second issue with human interpreted datasets is their fidelity. Not only can interpreters disagree about the exact position of a feature, such as a fault, but the drawn in feature will often be drawn with varying amounts of smoothing or resolution. Together with the interpretation being a very laborious process, real datasets are scarce and of varying quality. Since a neural network can only be as good as the data it is trained on [78], researchers have turned their attention to synthetic datasets. A common phrase used in the deep learning field signifying this principle is "garbage in, garbage out".

Synthetic datasets are produced by forward modelling and creating new geology that is different from what can be seen in the world but made to resemble the real data. Such synthetic datasets have the major advantage of having perfect labels, since the process of how exactly the synthetic image was created is known.

Synthetic data is therefore very consistent and the datasets can be made arbitrarily large. Their downside is that the synthetic images might not perfectly resemble the real images, nor do they capture all the possible geology that can be encountered in seismic surveys as they are restricted to assumptions about how geological processes work. Nevertheless, networks trained on synthetic data are capable of transferring to real data [66, 69, 75, 79] and the advantages of synthetic datasets seemingly outweigh their downside and make them more popular data type. When training a network on synthetic data and then deploying on real, a decrease in the accuracy of the models does occur. The field of Deep Domain Adaptation is focused on minimising said performance decrease and making neural networks work effectively with multiple data domains.

The common principle behind all of domain adaptation is that the models are treating different domains differently, yet not independently. They transfer learn from other domains to ultimately become more effective within its target domain. Within the sub-field of homogeneous domain adaptation, as described by Wang and Deng [80], the domains share the same feature space, which is the case for real and synthetic seismic data. The predominant way the domain adaptation works in this case is that two domains are mapped onto a shared representation. How this is achieved varies depending on the nature of the labels, with a number of approaches that exist. In the seismic data case, the real data, which is the target domain, is unsupervised. For this homogeneous unsupervised case Wang and Deng [80] categorised the existing approaches into the following: Discrepancy-based, Adversarial-based and Reconstruction-based.

With discrepancy-based approaches, the model is trying to make the shared representation similar across all the domains. Statistic criterion attempts to directly compare the shared representation of the data from various domains and apply a loss that will encourage the representation to be similar. These can be maximum mean discrepancy [81], correlation alignment [82], Kullback Leiber divergence [83] and H divergence [84]. Alternatively, architecture criterions can be used, where the architecture of the network is directly adjusted to aid with the transfer learning, such as batch normalization [85] or dropout [86, 87].

Adversarial-based approaches leverage a domain discriminator that is trained to distinguish between the various domains that the representation stemmed from, and then is adversarially used to minimise the distance between representations. These can be generator based [88, 89] or non-generator based [90, 91].

Reconstruction-based approaches reconstruct the source data again from the representation in the original or the other domains that it was trained on. These can be encoder-decoder based models, such as stacked autoencoders, or adverserially trained models, such as Dual GAN [92], Cycle GAN [93], or Disco GAN [94].

Specifically for the task of semantic segmentation there have been a few proposed models targeted at homogeneous unsupervised domain adaptation. Contributions that focus on self driving car applications are of particular significance to our work. There is a lot of focus on the semantic segmentation of a car's dashboard camera, with numerous real datasets as well as video game datasets available. These datasets are in principle very similar to synthetic and real seismic data, since the features are extremely similar across both domains, with the main difference being the textures and noise. There are, however, notable differences between the dashboard camera and seismic data, which mean that not all approaches will translate from one data domain to the other. Notably, dashboard camera data can exploit properties like the shape of objects, or the position of the sky and ground in the image and leverage these when designing the domain adaptation, such as the solution proposed by Hoffman et al [95] or Chen et al. [96].

Hoffman et al. [95] first introduced a model that uses adversarial-based training to perform domain alignment on fully convolutional networks. They make use of two principles, firstly minimising the global distribution of the position of the class labels in the images. This principle is not useful for seismic images, since we cannot rely on the global positioning of faults to be similarly distributed in the synthetic data and real data. Secondly, they perform a category specific adaptation which takes the distribution of pixels of a single category into account and trains both domains to output similar distributions. This second loss doesn't translate well to interpreting seismic images, because of the average shape of a feature such as a fault doesn't bare much significance.

Sankaranarayanan et al. [97] propose a method on the Cityscapes dataset. They treat the two domains of data in the same way, however they encourage the representation in a specific hidden layer of the network to be similar for both data domains. They do this by reconstructing the source images from the hidden layer by a generator network, which is trained using an adversarial discriminator. This method improved the performance of the model by 23% on the intersection over union metric.

A similar method is proposed by Huang et al. [88]. Their proposed solution makes use of supervised domain adaptation, with labels for both real and synthetic data domains, however they make use of unsupervised domain adaptation principles that still apply to seismic data. Their models utilises a reconstruction based adversarial model, which maps both data domains onto a shared representation Z. This representation Z is then used to recreate the image in both the original data domain and in the opposite domain. Adversarial discriminators are then used to ensure the recreated images look like the original data and so that the discriminator cannot tell which data domain the image originated from. This encourages the shared representation Z to also be independent from the original data domain.

Choi et al. [98] take a different approach to unsupervised domain adaptation, where they create an augmented synthetic dataset that looks similar to the real dataset, by applying a style transfer network from the synthetic to the real data. This method shows very promising results on the dashcam data, however it might struggle on seismic data, since there is nothing in the style transfer network that enforces that the position of the features in the augmented image remains the same as the source image. The neural network architecture ensures that the features are mostly stationary, which is sufficient for photographic images, however in seismic images small shifts in the image can have a major effect on the interpretation of the image and the features within it. Despite the augmented image looking like a proper seismic image, it might no longer match up with the labelled interpretation.

For the specific case of unsupervised homogeneous domain adaptation that is effective on seismic synthetic and real data, there are few proposed architectures. There is space for further research that combines the principles from a number of

principles with the seismic data in mind. Specifically I expect a combination of a statistic criterion, an adversarial approach and a reconstruction based approach to be the most promising on seismic data, which I combine in the GAN architecture in Section 7.3. However, this approach still suffers from fundamental issues, which led me to abandon the avenue of research. I later experimented with self-supervised learning as a reconstruction based approach with limited success.

CHAPTER 3

Datasets

Throughout my research, a consistent element is the use of various datasets, which play a central role across multiple chapters. These datasets comprise input data and, where applicable, corresponding labels. The nature of each dataset directly influences the specific task for which Deep Learning models are adapted and trained. With my biggest contributions being in representing and predicting orientation, most of my datasets address this task. They therefore need to have annotations of the orientation of features in their images. However, very few real datasets exist that have such annotations, which is why I rely on synthetic images. The first level of datasets that I use are made of binary images, consisting of either lines, curves, or planes in 3D, which I refer to as dummy datasets. These are fully synthetic datasets that give me the most accuracy in their labels and are the simplest. The second level of datasets are synthetic datasets that are made to resemble real images as best as they can, while still providing labels and orientations. These include the chromosome dataset and the synthetic seismic data. Lastly, real unlabelled datasets are what the models are ultimately meant to be deployed on, even though they cannot be trained on them because of the lack of labels. I also use these datasets in the Instance Separation Chapter 6, which uses the predicted values of the orientation

Figure 3.1: Example input images from the 2D Dummy Dataset.

applied to the real data to separate the different instances of faults or chromosomes. Since this is a hand-engineered algorithm as opposed to machine learning, it does not require labels, making the real dataset most appropriate.

3.1 2D Datasets

One of the first simplifications for predicting orientation was to address the 2D case first, before moving on 3D which is needed for the seismic interpretation. Predicting orientation in 2D shares many challenges with 3D orientation while reducing the complexity of the representation, making it easier to find or generate datasets for the task, and making the training and evaluation of the approach faster since smaller models are sufficient for the task.

3.1.1 Dummy 2D Datasets

My early experiments for predicting orientation were using a fully synthetic black and white dataset. Although this datasets was completely superseded with the chromosome dataset, it served an important role in figuring out how to approach the task. Examples of the dataset can be seen in Figure 3.1.

The input images in the dataset are greyscale and are drawn as white lines of various thicknesses on a black background. Each image contains one, two, or three

lines. The can be straight lines, Bezier curves, or custom random curves that have a randomly changing curvature. The images are annotated with the segmentation of each individual line and the orientation defined as a tangent vector for each pixel on the lines.

The primary use of this dataset is to evaluate the 2D orientation representations for pixels that belong to a single line. Points where lines intersect are annotated with multiple orientations at the same time. I discuss the idea of predicting the multiple orientations in these pixels simultaneously in Section 4.3, but ended up abandoning the project. Lastly, this dataset could also be used for the separation of the individual line instances. I explore these ideas in Chapter 6, but only deploy them on the chromosome dataset that superseded this dummy dataset.

3.1.2 Chromosome Datasets

Moving on from the dummy dataset, I was searching for a similar dataset to the dummy dataset with overlapping instances of objects that are flat. A two dimensional orientation must be sufficient to describe their orientation. Unlike the dummy dataset, this dataset should be made of real images that require the network to perform a semantic segmentation to distinguish the objects from the background. I was inspired by an assignment I had during my undergraduate study in which we had to segment and separate instances of worms using simple image processing techniques and without deep learning. However, I was not able to find the dataset, nor any related fields of work that may be interested in such algorithms. Instead I turned to chromosomes.

When karyotyping, chromosomes are segmented and arranged by size from a microscope picture of the chromosomes with various dyes. Frequently, chromosomes overlap in these images. Instead of segmenting the overlapping chromosomes, the experiment is repeated until a slide is found with chromosomes that do not overlap. If an algorithm could segment even the overlapping chromosomes, it would speed up the process of karyotyping.

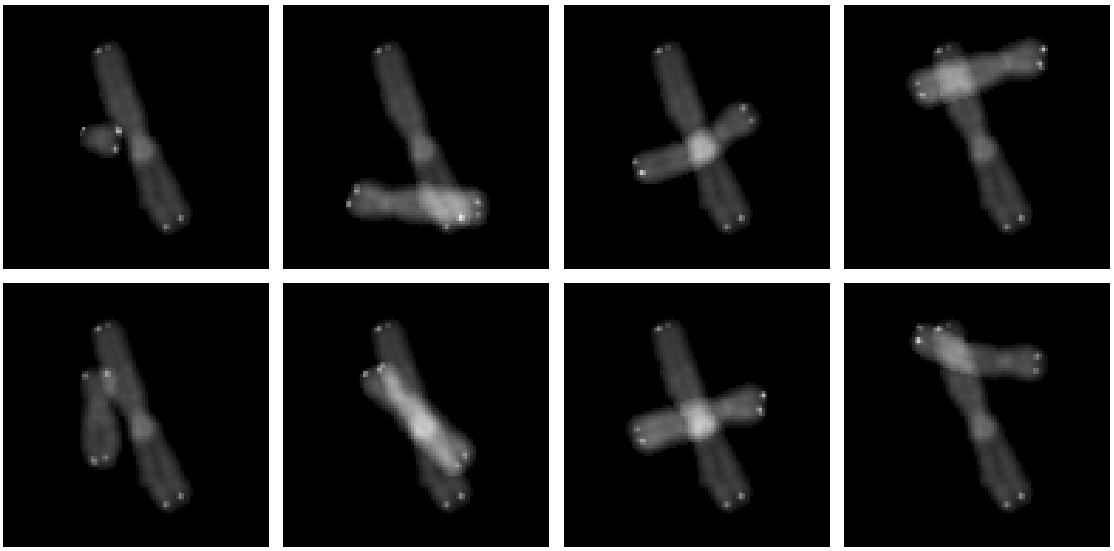


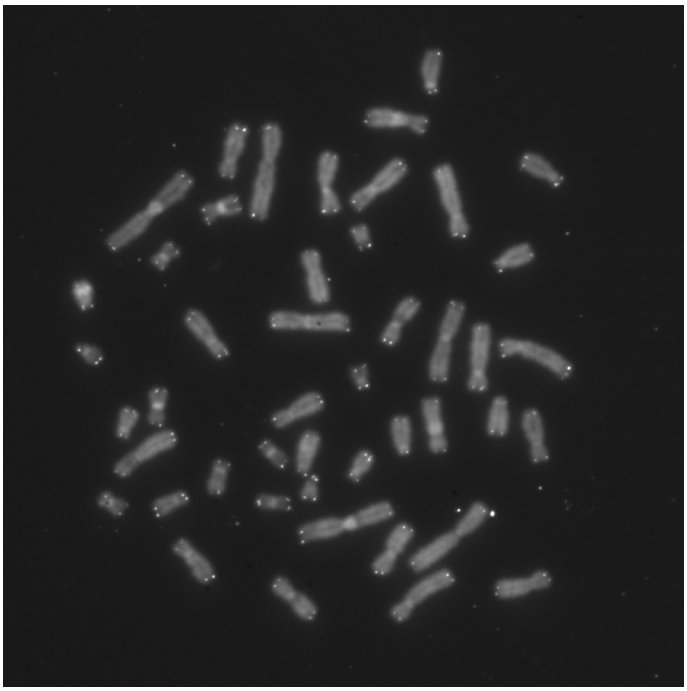
Figure 3.2: Example images from Pommier's dataset of synthetically overlapping chromosomes. A specific subset of images was selected that all share one chromosomes in the same orientation and position.

Pommier's Synthetic Dataset

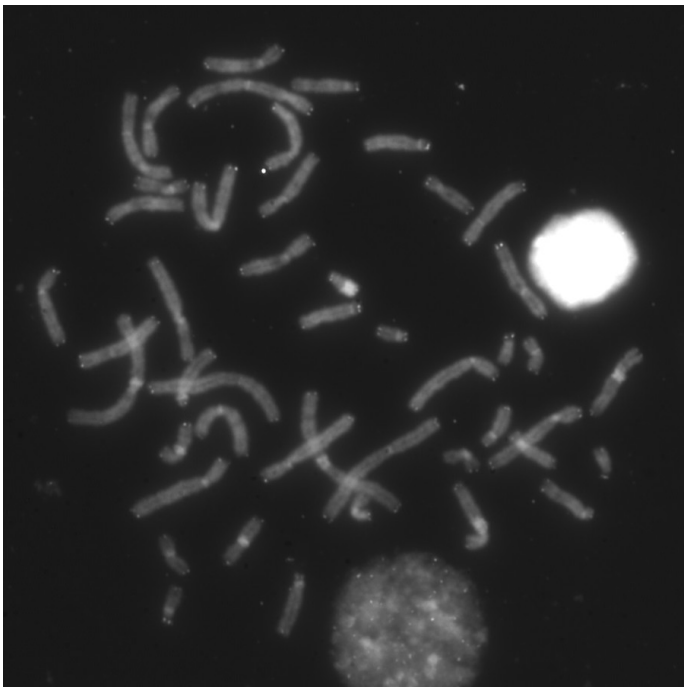
Jean-Patrick Pommier published a synthetic dataset of 13434 images of overlapping chromosomes [1], with examples visualised in Figure 3.2. Pommier also published the code used to create the synthetic images in a blog post where he also explains the process. The dataset is created from a single image of a human metaphase stained using 4',6-diamidino-2-phenylindole together with a Cy3 fluorescent telomeric probe [99], which is visualised in Figure 3.3. The image contains 46 individual chromosomes that do not overlap or touch. The synthetic dataset is created by taking pairs of chromosomes, rotating, translating, and averaging their pixel values. The original dataset is annotated with a segmentation of the chromosomes with categories: ch1, ch2, overlap and background. Although the dataset does not contain information about the orientation, these labels can be generated by manually annotating the 46 source images and adjusting the synthetic image generation code accordingly.

However, as presented by Pommier, this dataset suffers from one major flaw. Since all of the images in the dataset were created from the same 46 individual

[1] Pommier, J.P. (2016) Overlapping chromosomes dataset
`https://www.kaggle.com/jeanpat/overlapping-chromosomes`
`https://github.com/jeanpat/DeepFISH`



(a) The image of the slide that Pommier's dataset is made from.



(b) The image of a different slide also published by Pommier. This slide is not used in Pommier's dataset, but is used in mine.

Figure 3.3: Raw images of chromosome slides. Unlike Pommier, my approach also considers slides that contain overlapping chromosomes. Only the non-overlapping subset of chromosomes is used to generate the synthetic dataset, while the overlapping ones are kept for the real dataset used for evaluation.

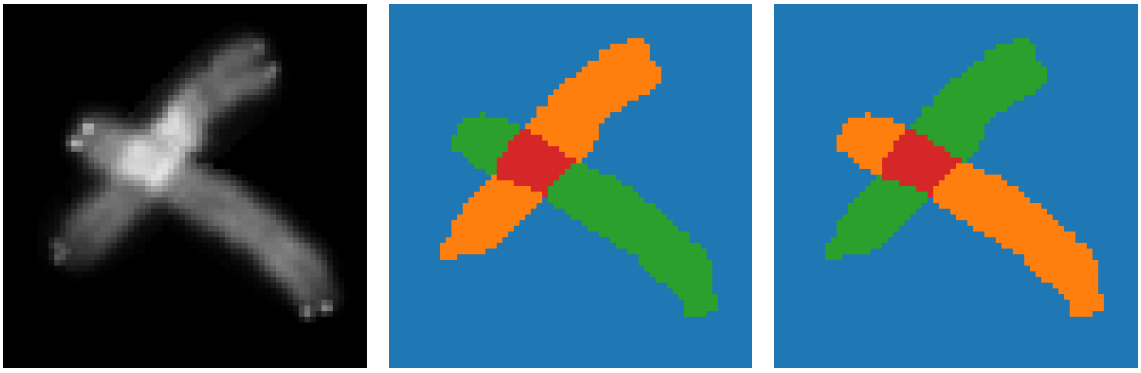


Figure 3.4: An example of an image from Pommier's dataset, with two viable annotations. It is unclear which should be correct.

chromosome images, they are not independent. This means that we cannot separate them into train/validation/test sets, since these subsets must be independent from each other. This is shown in Figure 3.2, since the selected synthetic images all share the same chromosome in the same orientation and position. However, Pommier's dataset has separated the images into train/validation/test sets, which is how the dataset was used in some related work. I explore the issues with the approach used in related work in Chapter 6 and how it was undetected because of the train/validation/test sets not being independent. In summary, ch1 and ch2 cannot be distinguished using a semantic segmentation approach, since they are semantically identical. They are both chromosomes, and are merely two different instances of the same semantic class. It is unclear which class should be assigned to which chromosome in any image, as visualised in Figure 3.4.

My Synthetic Chromosome Dataset

To address the issue with Pommier's dataset, I modified it. Pommier also published all 15 images of chromosome slides, where he only used a single image for his dataset, because some chromosomes overlap in the other slides. An example can be seen in Figure 3.3. The chromosomes that do not overlap are used to create synthetic images in a process similar to what Pommier used, while the overlapping chromosomes are saved for a small validation/test set of real overlapping chromosome images. The 15 slides contain a total of 620 non-overlapping chromosomes and 30 images of overlapping chromosome pairs or larger chromosome clusters. To separate the

images into train/validation/test sets, the chromosome source images are separated according to the slide they were found on. In other words, chromosomes from the same slide will only ever appear in a single subset. Pairs of chromosomes from a given subset are then randomly sampled, the chromosomes randomly rotated and translated, and their pixel values averaged to create the final synthetic images. This process is performed at runtime to save the memory requirement of the dataset while also allowing any continuous rotation and translation values.

This dataset is therefore formed out of two parts. The synthetic images and the real overlapping images. Each of them are annotated in different ways and used for different tasks/evaluations.

The pixels in the synthetic images are annotated in the same way as Pommier's dataset, with class labels of: ch1, ch2, overlap and background. Additionally, the pixels in the ch1 or ch2 category are annotated with the orientation of the chromosome expressed as the tangent vector to the chromosome. The tangent vector was determined as the tangent to a centre-line drawn through the chromosomes, with every pixel on the chromosome annotated based on the closest point on the chromosome's centre-line. Note that the sign of the tangent vector is selected arbitrarily. There are also labels for the two orientations in the overlapping areas, one for each chromosome, however, I ended up abandoning my attempts at predicting these values.

The real images of overlapping chromosomes are annotated in two different ways, depending on what approach is used to separate the overlapping chromosomes. If the algorithm is capable of separating any arbitrary number of chromosomes in a cluster, then the labels depict each separate chromosome in a cluster. If the algorithm only distinguishes pairs of chromosomes, then clusters are separated into multiple images of adjacent or overlapping pairs of chromosomes. The images are cropped around the pair of chromosomes, however, other chromosomes from the cluster may be visible in the image. These images are annotated in a fashion similar to the synthetic dataset with classes: ch1, ch2, overlap and background. I never annotated these real image datasets with the orientations of the chromosomes.

3.2 3D Datasets

Moving from 2D orientation representations to 3D orientation representations required similar datasets, but in 3D. The selection of datasets is inspired by seismic imaging datasets with fault annotations. As such, the 3D datasets are made from 3D volumes made of voxels also known as regular rectilinear grids.

3.2.1 Dummy 3D Dataset

Similarly to the 2D case, my earliest experiments in 3D were with synthetic greyscale images with a black background and flat planes drawn in white. The orientation of the plane is defined by its normal vector, the sign of which is selected arbitrarily. A normal vector $\mathbf{n}$ and $-\mathbf{n}$ result in a plane with the same orientation, hence the need for negation invariant orientation representations. The planes passed through a randomly selected point in the volume with a randomly selected orientation, and multiple planes could be present in each volume of size $96 \times 96 \times 96$ voxels. Any voxel with a centre point within 3 units (distance between voxels) to the plane is deemed to be on the plane and is drawn in white. These voxels are also annotated with the normal vector of the plane. It is possible to get the annotations for every plane independently, where a voxel may lie on multiple planes in areas of intersection, and have multiple normal vectors in these points. In practise, however, I never used these annotations, since I never moved onto 3D representations with the multi-orientation representations discussed in Section 4.3.1. Instead the labels are simplified to a semantic segmentation with the classes: background, unique plane, intersection. The voxels on a unique plane are labelled with the orientation of the plane as its normal vector and left as arbitrary values for points on the background or the intersections.

Armed with the experience from the 2D case, I didn't spend a lot of time experimenting on this dataset. I knew that I only relied on the synthetic chromosome dataset for evaluating the 2D representations in the end, since it offered similar insight to the utility of the representations but in an context more similar to the intended use of the representations. Similarly in 3D, I quickly moved to the syn-

thetic seismic dataset in Section 3.2.3. However, there are aspects that cannot be easily evaluated using the synthetic seismic dataset, which led me to design a second dummy 3D dataset.

3.2.2 Toy 3D Dataset

The representations for 3D orientation suffer from issues that are only present for certain orientations. It is therefore important to evaluate the representations for every possible orientation. This is not possible with the synthetic seismic dataset, because it can only model faults that fall within a subset of all the possible orientations. For example, a perfectly horizontal fault with a dip of 0° cannot be modelled effectively. Additionally, rather than averaging over the whole dataset of random orientations, systematically evaluating the representations for all the orientations brings additional insight into the shortcomings of the representations. Lastly, training a model on the synthetic seismic data takes a lot of time. Having a simpler dataset made from smaller volumes that requires a smaller network to evaluate the orientation representations allows for much quicker evaluation. This toy dataset is designed to address all of these points.

The dataset is made of volumes of size $8 \times 8 \times 8$ voxels, with binary values of 0 for the background and 1 for a flat plane going through the centre with a specified orientation. Any voxel with its centre within 0.6 units of the plane is defined as a voxel on the plane. An example volume from this dataset can be seen in Figure 3.5

This dataset is annotated with only a single value for the orientation of the plane, defined as the normal vector to the plane with an arbitrary sign. This means that the predicted features (orientation) are per-image, rather than per-pixel. Although this makes the dataset's task somewhat different from the seismic interpretation, requiring different and much smaller networks, it allows to evaluate the orientation representations very effectively.

Note that the dataset is not actually a collection of volumes, but rather a method for generating the volumes at runtime. This allows for the generation of a volume for any continuous value of the orientation, which allows us to evaluate the representations for selected specific orientations.

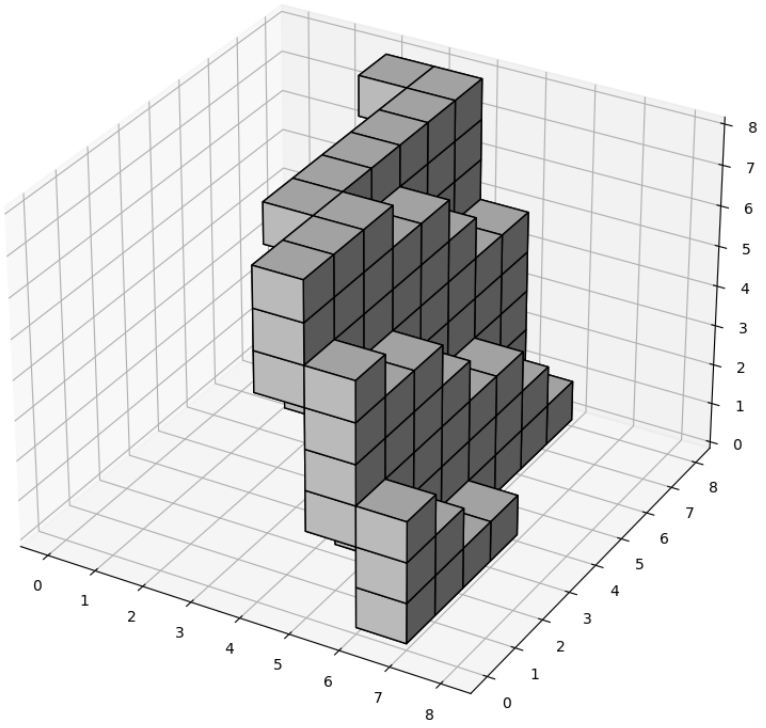


Figure 3.5: An example volume from the Toy 3D Dataset V2. © 2023 IEEE [5]

3.2.3 Synthetic Seismic Dataset

The synthetic seismic dataset was kindly provided by GeoTeric, the industry sponsor of my work. They provided me with a proprietary program that generates the synthetic seismic volumes, which I used to generate a dataset. The algorithm to generate the volumes is loosely based on Wu et al.'s approach [100], which was further explained in [101]. An example volume from this dataset is visualised in Figure 3.6.

The dataset I use consists of 1535 volumes, with 80% used for training, 10% used for validation, and 10% used for testing. Each volume has $96 \times 96 \times 96$ voxels and is annotated with the location of the faults, the location of the fault intersections, and fault segments which are defined as the faults minus the fault intersections. The orientation of the faults is defined as a normal vector to the faults fault segment voxels. Since the faults are flat planes, each voxel on the fault plane has the same orientation. The orientation of the fault intersections is also labelled and is defined as a vector in the direction of the intersection. It is calculated as the cross product of the orientations of the two faults that form the intersection.

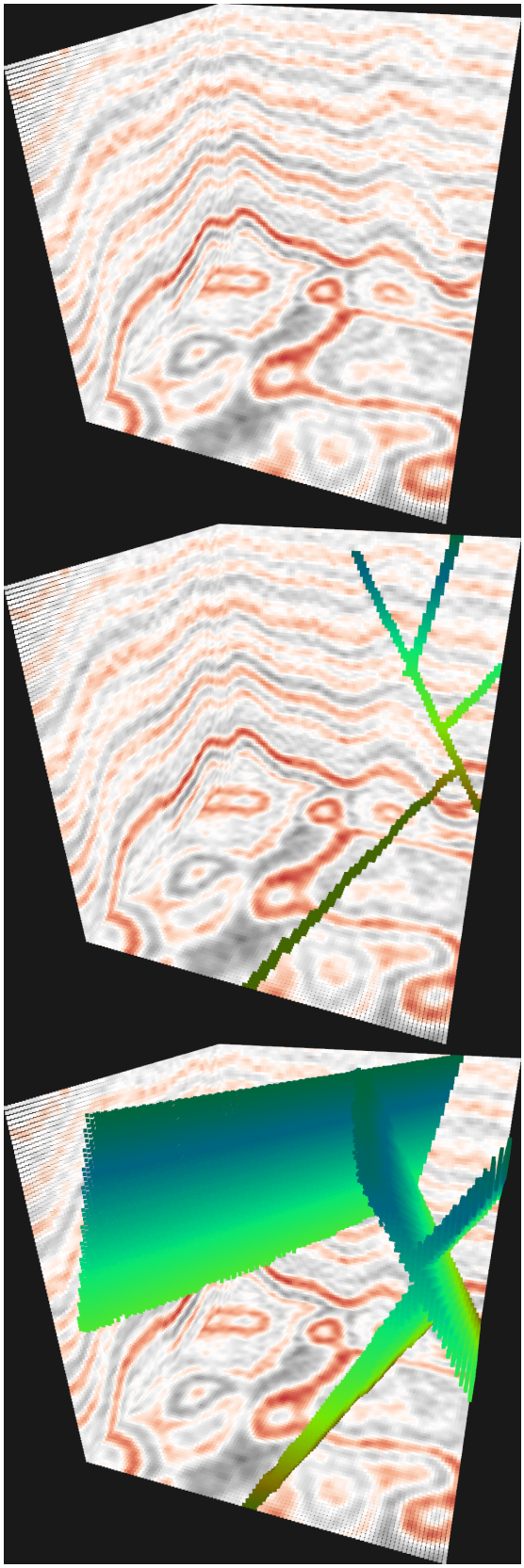


Figure 3.6: An example volume from the synthetic seismic dataset. The fault labels of the volume are also visualised. © 2023 IEEE [5]

The faults in the dataset are always flat planes and their dip angles are restricted, meaning that they do not span all possible orientations. The faults are also modelled in the same way regardless of the depth that they occur at. The synthetic seismic images are also only modelled as simple rock layers, without having other structures present, such as salt domes, a boundary with the sea floor, or edge artefacts at the ends of the seismic scan. Precisely this lack of variety is one of the biggest limiting factor of the synthetic images. Because of this, some model architectures such as transformer models that are capable of taking long distance relations in the images into account may not be beneficial when trained on this dataset. If the model is capable of better taking the geometry of the faults into account in its predictions, it may become sensitive to only predict faults that are flat planes and translate badly to real seismic data.

3.2.4 Real Seismic Datasets

Ultimately, the aim of my work is to automatically interpret real seismic surveys. There are numerous available seismic surveys, however, they do not contain readily accessible interpretations as labels. Most existing research on these surveys is in the form of descriptions and slices through the volume in research papers, which cannot easily be gathered to form a labels for supervised learning or evaluation. Additionally, one cannot truly rely on an expert's interpretation as a 100% correct ground truth, because of the uncertainty present in the interpretations and variation between different interpreters' habits, as seen in Section 2.2.1. This leaves two possible approaches to evaluate the models' performances. Firstly, to gather interpretations of the data independently, forming a ground truth to which the prediction is compared to. Secondly, to take the prediction made by the model and have it scrutinised by a geologist.

The main seismic survey that I use in my experiments is the Laminaria volume [102] from the Australian North West shelf imaging the thick Paleozoic, Mesozoic and Cenzoic sedimentary sequences deposited in a series of rifting phases. This is a well-known high quality dataset that my supervisors, Kenneth J. W. McCaffrey and Thomas B. Phillips have experience with, having recently published an paper

on parts of the dataset [6]. The dataset covers an area of 680km^2 with an inline and crossline spacing of 12.5m and records to 3s TWT depth. Due to the size of the dataset, it becomes unmanageable to evaluate all of it. Instead, I used small selections and slices of the dataset for any evaluations.

In order to gather independent ground truth annotations, 15 geologists from the Department of Earth Sciences at Durham University kindly annotated four slices through the volume. Three slices are in the North-South direction, while one is East-West. Due to the general trend of most faults being ENE-WSW trending, the E-W slice is nearly parallel with the faults causing them to be very difficult to see in the slice. The N-S slices are much more useful, however, the interpretations still vary greatly. Notably, these interpretations only annotated fault presence and not the fault orientations. The results of the experiment are described in more depth in Chapter 6.2.

Lastly, Thebes is a second seismic survey is noteworthy because of it being published with annotations of faults [103]. The seismic survey is called Thebe Gas Field and is located in the Exmouth Plateau of the Carnarvan Basin on the NW shelf of Australia. With fault labels that can be used as ground truth labels, it seems promising to use it as ground truth to evaluate my models. However, it suffers from some major flaws. Firstly, not all faults are annotated. In fact, the faults that are annotated are mostly present at a specific depth of the volume, with large parts being completely unannotated. In fact, the authors of the interpretation suggest only using cropped images from the survey that contain faults because of this. Moreover, even in the regions that are annotated, only selected faults are labelled, which are the biggest and most visible faults. In comparison, my models predicted many more smaller faults as well, most of which were perceived as correct by my supervisor Kenneth J. W. McCaffrey who is a geologist. One can therefore not simply use the annotations of the Thebe dataset as a complete ground truth, but only as partial annotations. Additionally, the associated paper with the publication of the dataset [104] claims that the dataset is good enough to train models on and compare it to Wu et al.'s synthetically trained model [105]. I do not believe that this is a fair comparison, because both the training and evaluation was performed

on different parts of the same Thebe dataset, which share the same biases for which faults are annotated and which are ignored. The model can therefore learn the appropriate strategy to selectively label and ignore certain faults to most closely match the dataset, which does not reflect the true performance of detecting all faults.

CHAPTER 4

2D Orientation Representation

For the purposes of this thesis, the orientation is defined as the set of all lines passing through the origin. Equivalently, we could think of it as the set of all unit vectors, where any vector $\mathbf{v}$ and its negative $-\mathbf{v}$ represent the same orientation. It is in essence a unit vector without its arrowhead, in other words where we do not care about the vector's sense. In mathematics, projective geometry offers a natural framework for this concept of orientation. Specifically, projective geometry deals with properties that remain invariant under projection, which is beneficial for understanding orientations.

In this context, the orientation could be represented using a real projective space $\mathbb{RP}$ (or $\mathbb{RP}^2$ for three dimensional vectors). The space $\mathbb{RP}$ consists of equivalence classes of non-zero vectors in $\mathbb{R}^2$, where two vectors are equivalent if they are scaled versions of each other $\mathbf{v} = \lambda\mathbf{v}\ \forall\lambda \in \mathbb{R}$. A euclidean vector $\mathbf{v} = (x, y)$ would be expressed as the homogeneous coordinate $(x : y)$ in this projective space, which would be equivalent to all homogeneous coordinates $(\lambda x : \lambda y)\ \forall\lambda \in \mathbb{R}$. However, the projective spaces cannot effectively be used as a representation of orientation predicted by a neural network under supervised learning. That is because neural network require a single unique label value that serves as a ground truth, to which

the predicted value is compared using the loss function. However, the framework of this projective space relies on the principle of various homogeneous coordinates being equivalent, which is the opposite of having a unique label.

Orientation can also be understood as a direction of something that has 180° rotational symmetry. For example, the orientation of a fence or a road. We could call it north-south facing or south-north facing. Similarly, we could describe its orientation with bearings: 6° or 186°; or with unit vectors $(0.10, 0.99)$ or $(-0.10, -0.99)$. In 3D, we could represent the orientation of a plane using a normal unit vector to the plane, where any negative vector represents the same plane. The orientation of a line in 3D could be described using its tangent unit vector. We could even equivalently consider orientation as a negation invariant vector in any number of spacial dimensions.

I will define orientation representations in terms of unit vectors $\mathbf{v}$ as a function $r : \{\mathbf{v} \in \mathbb{R}^n \mid \|\mathbf{v}\| = 1\} \to \mathbb{R}^m$ that maps any direction $\mathbf{v}$ onto the representation of the orientation. This orientation representation function r is even, where $r(\mathbf{v}) = r(-\mathbf{v})$. Unlike homogeneous coordinates in $\mathbb{RP}$, these vectors are only negation invariant since their magnitude must be 1. However, in order for this representation of orientation to be effective, it needs to have a number of other properties as well.

We want the representation of orientation to uniquely represent any orientation. This means that any for two distinct directions as unit vector $\mathbf{v}$ and $\mathbf{u}$, where $\mathbf{v} \neq \mathbf{u}$ and $\mathbf{v} \neq -\mathbf{u}$, we want $r(\mathbf{v}) \neq r(\mathbf{u})$. These properties satisfy the basic requirements for a representation of orientation.

If we want to use a neural network to predict orientation, not all representations of orientation are equally effective. We need to consider how the network returns the values of the representation, what activation function it uses, what loss function we use to train the network, and how many network outputs are required to represent said orientation. Moreover, the complexity of the representation also plays a role in how difficult it is for the network to predict. There are a number of principles that I believe make an effective representation of orientation, which I try to justify both theoretically and empirically.

The output from a neural network is inherently continuous, since neural networks

are made of components that are all continuous, such as linear combinations, non-linear activation functions, normalisation layers, additions, and multiplications. We can therefore expect that small variations in the input will yield small variations in the output of the model. For this reason, we want the representation r to also be continuous. We cannot expect a neural network to correctly model a discontinuous function. If we rotate the unit vector $\mathbf{v}$ by a small amount to get $\mathbf{u}$, so $\mathbf{v} \approx \mathbf{u}$, but $r(\mathbf{v}) \not\approx r(\mathbf{u})$, then a neural network would struggle to predict this representation of orientation effectively.

Representations used for neural networks are commonly continuous, such as the following example of representing angles. If we represented an angle θ directly using the representation $r(\theta) = \theta \mod 360°$, we would get a discontinuity every 360°. This would be problematic, and we would expect a neural network's approximation of the function to have errors near the discontinuity. This is visualised in Figure 4.1. Instead, a commonly used representation is $\cos(\theta)$ and $\sin(\theta)$. Together, these two values uniquely define every angle, while also being continuous. This representation can also be considered as a unit vector pointing in the direction of the anticlockwise angle from the x-axis.

4.1 Representation Definitions

In order to represent orientation in 2D in a continuous manner, we have to represent an angle that repeats every 180°, i.e. $\theta \mod 180°$, or equivalently points along a semi-unit-circle where its endpoints map to the same point. The best way to represent this is using $\cos(2\theta)$ and $\sin(2\theta)$. I call it the Doubleangle representation. I will first talk about the properties of this representation, how and where it is useful, before moving onto imperfect alternatives that I considered before arriving at the Doubleangle representation. Piecewise representations are also a type of representation that I also use for representing 2D orientations on the 2D datasets. I will describe them separately in Section 4.1.5, because of the many iterations they went through and because they can also be applied to 3D orientation and in theory any n-dimensional negation invariant representation.

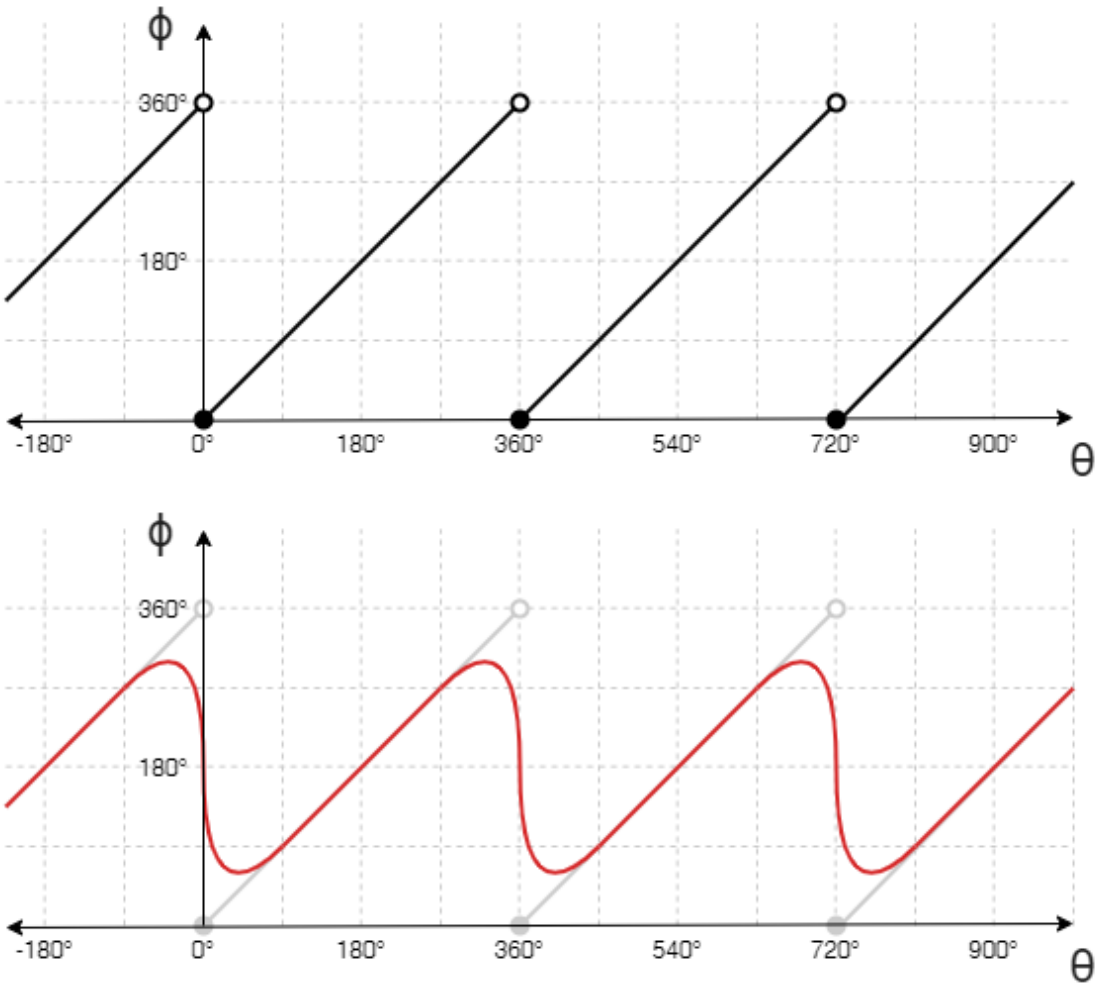


Figure 4.1: An example of a periodic function that is discontinuous shown in black, with a continuous approximation of the function shown in red. The function shown is defined as $\phi(\theta) = \theta \mod 360°$.

4.1.1 Doubleangle Representation

The Doubleangle representation can be defined in two ways. In both cases it represents a direction, but the direction can be expressed in different ways. First, as an angle θ, where θ and $\theta + 180n^\circ$ for $n \in \mathbb{Z}$ all map to the same orientation. Second, the direction can be expressed as a vector $\mathbf{v}$, where $\mathbf{v}$ and $-\mathbf{v}$ map to the same orientation. First, let us define how the direction θ as an angle relates to the direction $\mathbf{v}$ as a vector:

$$\begin{aligned} \theta &= \operatorname{atan2}(v_1, v_0), \\ \mathbf{v} &= \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}. \end{aligned} \tag{4.1}$$

The Doubleangle representation f is defined as follows in terms of θ:

$$\begin{aligned} r &: \mathbb{R} \to \mathbb{R}^2, \\ r(\theta) &= \begin{bmatrix} \cos(2\theta) \\ \sin(2\theta) \end{bmatrix}, \\ r^{-1}(\mathbf{u}) &= \frac{1}{2} \operatorname{atan2}(u_1, u_0). \end{aligned} \tag{4.2}$$

Note that $r^{-1}(f(\theta)) = \theta \mod 180^\circ$. Equivalently, the Doubleangle representation r is defined as follows in terms of $\mathbf{v}$:

$$\begin{aligned} r &: \mathbb{R}^2 \to \mathbb{R}^2, \\ r(\mathbf{v}) &= \|\mathbf{v}\| \begin{bmatrix} \cos\big(2 \operatorname{atan2}(v_1, v_0)\big) \\ \sin\big(2 \operatorname{atan2}(v_1, v_0)\big) \end{bmatrix} = \frac{1}{\|\mathbf{v}\|} \begin{bmatrix} v_0^2 - v_1^2 \\ 2 v_0^2 v_1^2 \end{bmatrix}, \\ r^{-1}(\mathbf{u}) &= \|\mathbf{u}\| \begin{bmatrix} \cos\big(\frac{1}{2} \operatorname{atan2}(u_1, u_0)\big) \\ \sin\big(\frac{1}{2} \operatorname{atan2}(u_1, u_0)\big) \end{bmatrix} = \|\mathbf{u}\| \begin{bmatrix} \sqrt{\frac{1}{2} + \frac{u_0}{2\|\mathbf{u}\|}} \\ \operatorname{sign}(u_1) \sqrt{\frac{1}{2} - \frac{u_0}{2\|\mathbf{u}\|}} \end{bmatrix}, \end{aligned} \tag{4.3}$$

Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$, since both directions $\mathbf{v}$ and $-\mathbf{v}$ have the same orientation. Also note that when defined in terms of $\mathbf{v}$, the magnitude of the vectors is allowed to be different from 1 and is maintained, such that $\|r(\mathbf{v})\| = \|\mathbf{v}\|$ and $\|r^{-1}(\mathbf{u})\| = \|\mathbf{u}\|$. So far, we have only talked about the case where the magnitude is fixed at 1, however,

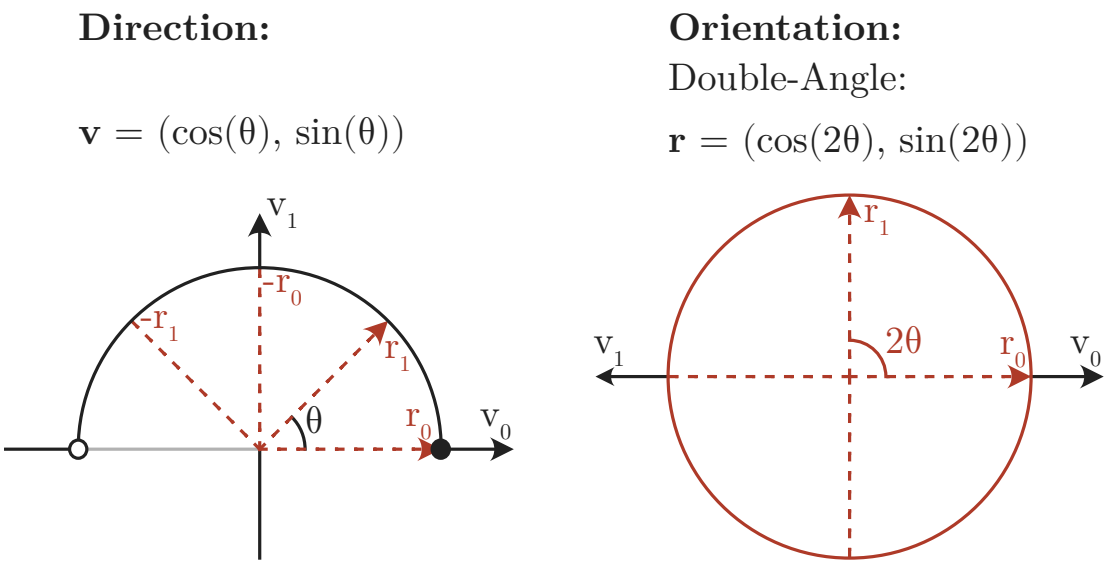


Figure 4.2: A visualisation of the Doubleangle representation. The left diagram visualises direction that is represented **v**, while the right diagram is aligned with the axes of the representation itself **r**, shown in red.

in future uses, this principle will become useful.

There are a number of ways to visualise this representation. The easiest is to consider the semi-circle formed by $(\cos(\theta), \sin(\theta))$ for $\theta \in [0°, 180°]$, as the directions that we are trying to represent. These angles are then doubled, stretching the semi-circle to form a full circle. This is visualised in Figure 4.2.

This representation behaves in a very consistent manner for any direction θ, where a change in the direction leads to a proportional change in the representation. At the same time, when viewing a vector **v** in the representation space $r(\mathbf{v})$, it is difficult to get an intuition for what orientation it represents, unlike viewing the direction vector **v**, which can easily be interpreted as a tangent or normal vector.

Although this representation does not contain any of the properties that I identified as potentially problematic, this does not imply that it works well. Empirical evidence is required to demonstrate its effectiveness and to compare it with other representations. It is unclear how the dataset used or the neural network architecture affects the performance of the model with different representations. A very important principle that I have not addressed about the representation yet is the loss function used to train the model. I will discuss this in the Loss Functions Section 4.1.6.

4.1.2 Angle Representation

The simplest and most naive representation is to predict the angle $\theta \in [0°, 180°)$ directly. We measure this angle anticlockwise from the x-axis. Neural networks can be trained on regression tasks to directly predict output values of some hidden continuous function they are modelling. However, an angle in modulo space is not continuous, because of the jump from 0° to 180°. More formally, we get a discontinuity because of the following limits:

$$\begin{aligned} \lim_{\theta \to 0^+} (\theta \mod 180) &= 0°, \\ \lim_{\theta \to 0^-} (\theta \mod 180) &= 180°. \end{aligned} \tag{4.4}$$

The existence of a discontinuity does not mean that a neural network cannot learn to predict this representation. However, we cannot expect the network to perfectly predict the angle near the discontinuity, since its predictions must be continuous. We might expect the network to predict values similar to those seen in Figure 4.1. Moreover, what makes this representation problematic is that the errors are only introduced near specific orientations, namely 0° or 180°. This would make the network better at predicting certain orientations than others, which introduces an undesirable bias to the predictions.

When using this representation we want to ensure that the representation values are normalised to a sensible range. Angles in degrees have values that are too big for effective use with models and optimisers that work best with values normalised to the standard normal distribution. Instead, predicting values in the range $\theta \in [0, 1)$ is much more appropriate.

4.1.3 Vector Representation

The natural evolution of the Angle Representation is to take a vector pointing in one of the two directions that correspond to the orientation. It follows a commonly used trick for representing angles $\theta \mod 360°$, where $\cos(\theta)$ and $\sin(\theta)$ are used as the representation. This representation works perfectly for standard angles, that repeat every 360°, where it eliminates the discontinuity. However, when representing ori-

entation, we are restricted to angles $\theta \in [0°, 180°)$, which also leaves a discontinuity at 0° and 180°, since:

$$\begin{aligned} \lim_{\theta \to 0^+} (\cos(\theta \mod 180°)) &= \cos(0°) = 1, \\ \lim_{\theta \to 0^-} (\cos(\theta \mod 180°)) &= \cos(180°) = -1. \end{aligned} \tag{4.5}$$

The vector representation is defined using one of the following two equivalent definitions. Firstly, in terms of θ:

$$\begin{aligned} r : \mathbb{R} &\to \mathbb{R}^2, \\ r(\theta) &= \begin{bmatrix} \cos(\theta \mod 180) \\ \sin(\theta \mod 180) \end{bmatrix}, \\ r^{-1}(\mathbf{u}) &= \frac{1}{2} \operatorname{atan2}(u_1, u_0) \mod 180°. \end{aligned} \tag{4.6}$$

Secondly, the vector representation defined in terms of the direction vector $\mathbf{v}$:

$$\begin{aligned} r : \mathbb{R}^2 &\to \mathbb{R}^2, \\ r(\mathbf{v}) &= \begin{cases} \mathbf{v} & \text{if } v_1 > 0 \text{ or } (v_1 = 0 \text{ and } v_0 \geq 0) \\ -\mathbf{v} & \text{otherwise} \end{cases}, \\ r^{-1}(\mathbf{u}) &= \mathbf{u}. \end{aligned} \tag{4.7}$$

Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$.

4.1.4 Piecewise-Angle Representation

Piecewise representations are a category of representations that I developed, which explore how the issues with discontinuities could be addressed by modifying the loss functions. I came up with the initial idea before coming up with the Doubleangle representation, which proved to be superior. However, I revisited the idea of piecewise representations as a viable idea for 3D orientation. Only then did I develop the representation further, which proved to be scalable to any number of

spacial dimensions including 2D orientation. In this section I will explain the 2D piecewise representations first, with all of their iterations and variants. Only then will I move onto the 3D counterpart, which is also applicable to nD (any number of spacial dimensions). This order will make it easier to follow the explanations of the representations, even though it is not the chronological order in which I developed them.

The general principle behind the piecewise representations is, that we have the model predict multiple representations simultaneously. These representations may still suffer from discontinuities, but we ensure that for every orientation at least one representation is well defined. If we then select the most appropriate representation for every given orientation in a piecewise manner, we should get a well defined representation for all possible orientations.

However, we still must be careful with the discontinuities, since the model is required to predict all of the representations in the piecewise ensemble for all orientations. This means that we require the model to predict the representations even near their discontinuities, and also calculate the loss for these orientations. This could cause the model to spend substantial resources trying to model the representations near their discontinuities, even though we will end up ignoring these predictions due the piecewise choice on deployment. The model could therefore leave less attention to the representation/orientation combinations that we care about.

Fundamentally, I explored two types of representations that are used in the piecewise ensemble: The first is predicting the angle corresponding to the orientation in 2D. The second is predicting the vector point in the direction of the orientation, which can apply to vectors in 2D, 3D, or nD.

The Piecewise-Angle Representation is based on top of the Angle Representation from Section 4.1.2. If we predict the orientation as an angle $\theta \in [0°, 180°]$, then the representation's discontinuity occurs at $\theta = 0° = 180°$. We can then use a second representation that is shifted by 90° to form the full Piecewise-Angle Representation

in equation 4.8:

$$
\begin{aligned}
r &: \mathbb{R} \to \mathbb{R}^2, \\
r(\theta) &= \begin{bmatrix} \theta & \mod 180^\circ \\ \theta + 90^\circ & \mod 180^\circ \end{bmatrix}.
\end{aligned} \tag{4.8}
$$

In this representation, $r(\theta)_0$ has a discontinuity for orientations $\theta = 0 = \pi$, and $r(\theta)_1$ has a discontinuity at $\theta = \pi/2$. To invert the piecewise representation, we will choose $r(\theta)_0$ or $r(\theta)_1$ depending on which is further from its respective discontinuity as follows in equation 4.9:

$$
\begin{aligned}
r^{-1} &: \mathbb{R}^2 \to \mathbb{R}, \\
r^{-1}(\mathbf{u}) &= \begin{cases} u_0 \mod 180^\circ & \text{if } \min(u_0, 180^\circ - u_0) \geq \min(u_1, 180^\circ - u_1) \\ u_1 + 90^\circ \mod 180^\circ & \text{otherwise.} \end{cases}
\end{aligned} \tag{4.9}
$$

4.1.5 Piecewise-Vector Representation

The next version of the piecewise representation is built on top of the Vector Representation from Section 4.1.3. By using two variables to define the orientation, we have the freedom to modify the magnitude of the vector. We will be able to use this to our advantage to remove the discontinuity from the Vector Representation in the piecewise context.

1st Iteration Piecewise-Vector Representation

To shift the location of the discontinuity of the Vector Representation, we can align the vector with a different axis. To do this we ensure that the aligned axis always has positive values. This is expanding on the vector representation from equations 4.7, which always aligned with the y axis. The following equation 4.10 is the new expanded definition of the aligned vector representation:

$$
\begin{aligned}
a &\in 0, 1, \\
r_a &: \mathbb{R}^2 \to \mathbb{R}^2, \\
r_a(\mathbf{v}) &= \begin{cases} \mathbf{v} & \text{if } v_a > 0 \text{ or } (v_a = 0 \text{ and } v_{1-a} \geq 0) \\ -\mathbf{v} & \text{otherwise} \end{cases}, \\
r_a^{-1}(\mathbf{u}) &= \mathbf{u}.
\end{aligned} \tag{4.10}
$$

Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$.

The first iteration of the Piecewise-Vector Representation is then defined as the pair of $r_1(\mathbf{v})$ and $r_0(\mathbf{v})$ from equation 4.10. With $r_0(\mathbf{v})$ being aligned with the 0-axis, which is the x-axis, its discontinuities occur at $x = 0$, which corresponds to $\theta = -90° = 90°$. $r_1(\mathbf{v})$ has discontinuities near $y = 0$, which corresponds to $\theta = 0 = 180°$. Since these two representation have discontinuities in different places, we can use the piecewise principle to select the one that is further from its discontinuity, when inverting the representation. The vector with furthest from its discontinuity is also closest to the axis with which it is aligned ($r_0(\mathbf{v})$ with the x axis and $r_1(\mathbf{v})$ with the y axis). Therefore, the vector can be chosen based on the size of its vector element on the axis with which it is aligned. For example, $r_0(\mathbf{v})$ is chosen if $r_0(\mathbf{v})_0 > r_1(\mathbf{v})_1$. Note that we use the notation: $r_a(\mathbf{v})_b$ as the b-th element of $r_a(\mathbf{v})$. The following equation 4.11 is the definition of the first iteration of the Piecewise representation and is visualised in Figure 4.3.

$$
\begin{aligned}
r &: \mathbb{R}^2 \to \mathbb{R}^2 \times \mathbb{R}^2, \\
r(\mathbf{v}) &= \Big(r_0(\mathbf{v}), r_1(\mathbf{v})\Big) \text{ using } r_0 \text{ and } r_1 \text{ from 4.10}, \\
r^{-1}(\mathbf{u}_0, \mathbf{u}_1) &= \begin{cases} \mathbf{u}_0 & \text{if } u_{0,0} \geq u_{1,1} \\ \mathbf{u}_1 & \text{otherwise} \end{cases}.
\end{aligned} \tag{4.11}
$$

Where $u_{a,b}$ refers to the b-th element of vector $\mathbf{u}_a$. Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$.

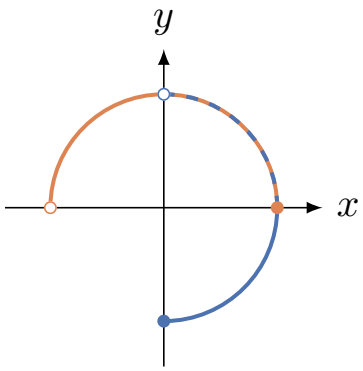


Figure 4.3: The first iteration of the Piecewise representation. The diagram shows the range curve of the representation. The first output in blue is aligned with the x-axis ($r(\mathbf{v})_0$ from equation 4.11) and the second output in orange is aligned with the y-axis ($r(\mathbf{v})_1$ from equation 4.11).

2nd Iteration Piecewise-Vector Representation

Even though the 1st Iteration Piecewise-Vector Representation avoids using the values near the discontinuities, the neural network is still asked to model the values near the discontinuities, which it cannot achieve in a continuous manner. The network will always have a greater error in this area, which might cause it to spend more resources on modelling this part of the output, even though we will ignore it in the end. Instead, we could slowly decrease the magnitude of the represented vector to 0 as it gets closer to the discontinuities. In this way, the discontinuity is avoided, since the representation will be $(0, 0)$. The downside of this approach is that we cannot tell what orientation is represented near $(0, 0)$, but that does not matter thanks to the piecewise principle and these outputs will be ignored anyway.

The question still remains of how the magnitude of the vector is tapered to zero. We will refer to this as the adjustment function. We could linearly reduce the magnitude to zero, as seen in Figure 4.4, or in a smooth sinusoidal manner as seen in Figure 4.5.

An advantage of using an adjustment function that tapers the discontinuity to the origin by multiplying the represented vector by 0 is that we can simplify the definition of the Aligned Vector Representation from equation 4.10. It no longer needs to distinguish the edge case where $v_a = 0$ and $v_b = \pm 1$. We get the following equation 4.12 for the second iteration of the Piecewise representation:

$$
\begin{aligned}
\operatorname{sign}(a) &= \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}, \\
r_a(\mathbf{v}) &= \operatorname{sign}(v_a) \cdot \mathbf{v}, \\
f_{\text{linear}}(\theta) &= \begin{cases} 1 & \text{for } \theta \in [0°, 45°) \\ \frac{90-\theta}{45} & \text{for } \theta \in [45°, 90°) \end{cases}, \\
f_{\text{smooth}}(\theta) &= \begin{cases} 1 & \text{for } \theta \in [0°, 45°) \\ \sin^2(2\theta) & \text{for } \theta \in [45°, 90°) \end{cases}, \\
\theta(\mathbf{u}, \mathbf{w}) &= \arccos\left(\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}\right) \\
r &: \mathbb{R}^2 \to \mathbb{R}^2 \times \mathbb{R}^2 \\
r(\mathbf{v}) &= \Big(f(\theta_0) \cdot r_0(\mathbf{v}),\ f(\theta_1) \cdot r_1(\mathbf{v})\Big) \\
&= \Big(f(\theta(r_0(\mathbf{v}), \mathbf{e}_0)) \cdot r_0(\mathbf{v}),\ f(\theta(r_1(\mathbf{v}), \mathbf{e}_1)) \cdot r_1(\mathbf{v})\Big), \\
r^{-1}(\mathbf{u}, \mathbf{w}) &= \begin{cases} \mathbf{u} & \text{if } u_0 \geq w_1 \\ \mathbf{w} & \text{otherwise} \end{cases},
\end{aligned}
\tag{4.12}
$$

where $\mathbf{e}_a$ is the basis vector for axis a, and two flavours of the representation exist depending on whether we use $f = f_{\text{linear}}$ or $f = f_{\text{smooth}}$. Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$.

3rd Iteration Piecewise-Vector Representation

The third and final iteration of the Piecewise representation addresses a minor issue, which stems from inaccuracies in the neural network predictions. I only developed this iteration later, after developing the 2nd iteration of the representation in 3D.

This iteration addresses a minor issue that stems from the piecewise nature of the representation. The representation suddenly switches from $r(\mathbf{v})_0$ to $r(\mathbf{v})_1$ and vice versa due to its piecewise nature. Although both representations should be identical at this point, we cannot guarantee that the model truly outputs the same values. It could be beneficial to gradually switch from one representation to the other instead. We could do this by performing a weighted average between $r(\mathbf{v})_0$ and $r(\mathbf{v})_1$ based

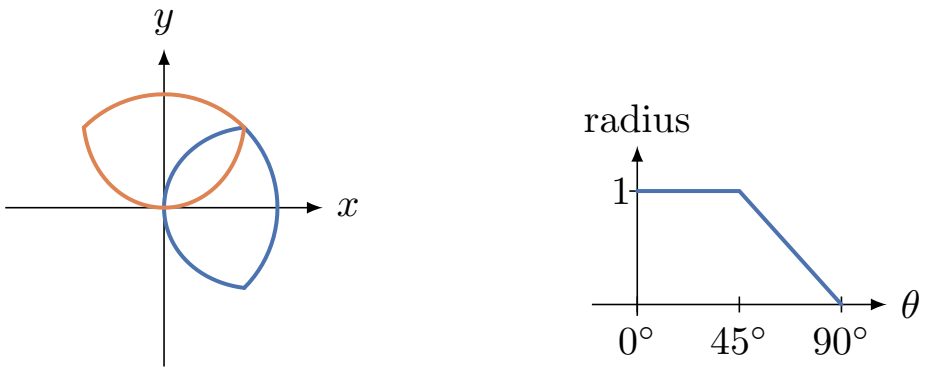


Figure 4.4: The Second Iteration Piecewise-Vector Representation with a linear adjustment function f_{linear}. The left diagram shows the range curve of the representation, while the right visualises the adjustment function or the magnitude of the representation vector. The first output in blue is aligned with the x-axis ($r(\mathbf{v})_0$ from equation 4.12) and the second output in orange is aligned with the y-axis ($r(\mathbf{v})_1$ from equation 4.12). θ in the adjustment function corresponds to the angle between the vector and the axis that it is aligned to.

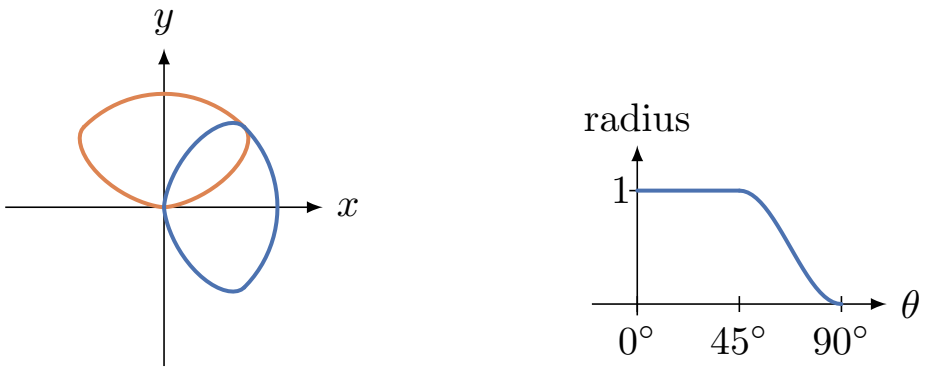


Figure 4.5: The Second Iteration Piecewise-Vector Representation with a smooth adjustment function f_{smooth}. This figure is otherwise similar to Figure 4.4.

on their magnitudes, which are determined by the adjustment function. We only have to be careful that both $r(\mathbf{v})_0$ and $r(\mathbf{v})_1$ point in the same direction and are not negatives of each other.

To perform a weighted average, we add the two vectors and divide by their magnitude. We could instead change the adjustment function, such that the two vectors' magnitudes should add up to 1. To do this, we can rely on the principle $\sin^2(x) + \cos^2(x) = 1$ and simply define the adjustment function as $\cos^2(\theta)$, which can be equivalently expressed as $r(\mathbf{v})_a^2$ for the output aligned with the axis a. Using the Pythagorean theorem, $r(\mathbf{v})_0^2 + r(\mathbf{v})_1^2 = \|r(\mathbf{v})\| = \|\mathbf{v}\| = 1$.

When inverting the representation, to ensure that the vectors $r(\mathbf{v})_0$ and $r(\mathbf{v})_1$ point in roughly the same direction and are not negatives of each other, we align them both with the same axis. However, if we arbitrarily chose an axis and the vectors have elements close to zero on that axis, small differences in the vectors could lead to a change in the sign of that axis element, leaving the aligned vectors as effectively negatives of each other. Therefore, we pick the axis to align with that has the biggest elements, which is the axis a where $r(\mathbf{v})_a$ has the largest magnitude. This makes it most likely that the two aligned vectors are closer to each other than if we multiplied one of the vectors by -1.

The third iteration of the Piecewise representation is defined as follows in equation 4.13 and shown in Figure 4.6:

$$
\begin{aligned}
\text{sign}(a) &= \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}, \\
r &: \mathbb{R}^2 \to \mathbb{R}^2 \times \mathbb{R}^2, \\
r(\mathbf{v}) &= \Big(v_0^2 \cdot \ \text{sign}(v_0) \cdot \mathbf{v},\ v_1^2 \cdot \ \text{sign}(v_1) \cdot \mathbf{v} \Big), \\
\text{Let } a, b \in \{0,1\} \quad &\text{s.t. } \|\mathbf{u}_a\| \geq \|\mathbf{u}_b\|, \\
r^{-1}(\mathbf{u}, \mathbf{w}) &= \ \text{sign}(u_a) \cdot \mathbf{u} + \text{sign}(w_a) \cdot \mathbf{w}
\end{aligned}
\tag{4.13}
$$

Additionally, if we wanted ensure that $r^{-1}(\mathbf{u}, \mathbf{w})$ is a unit vector even if $\mathbf{u}$ and $\mathbf{w}$ are not, we could normalise its magnitude using the expression $r^{-1}(\mathbf{u}, \mathbf{w}) \Big/ \|r^{-1}(\mathbf{u}, \mathbf{w})\|$.

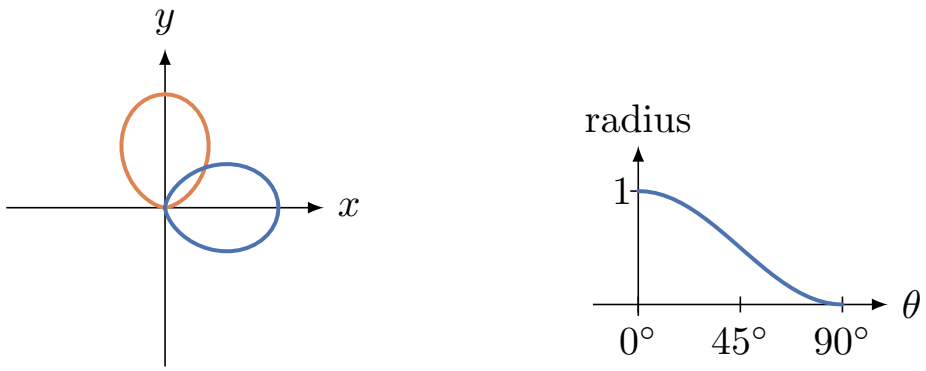


Figure 4.6: The third iteration of the Piecewise representation. The left diagram shows the range curve of the representation, while the right visualises the adjustment function or the magnitude of the representation vector. The first output in blue is aligned with the x-axis ($r(\mathbf{v})_0$ from equation 4.13) and the second output in orange is aligned with the y-axis ($r(\mathbf{v})_1$ from equation 4.13). θ in the adjustment function corresponds to the angle between the vector and the axis that it is aligned to. Note that the magnitude of $r(\mathbf{v})_0$ and $r(\mathbf{v})_1$ add up to 1.

Note that $r^{-1}(r(\mathbf{v})) = \pm\mathbf{v}$.

4.1.6 Loss Functions

As a regression task, the appropriate loss functions to try are the Mean Squared Error (MSE) and Mean Absolute Error (MAE) losses. These can be used to compare the prediction of the network with the target value represented as the representation.

There is another way to define the loss function, however, and that is to differentiate the representation itself. We could predict the values of the representation, transform it into an angle representation, and calculate the MAE or MSE between those angle values. Note that because of the 180° cyclical nature of the orientation, we must consider angles mod 180° and whether it is a smaller distance to go across the 0° to 180° discontinuity. We can calculate this angular distance mod 180° using the following equation 4.14:

$$\text{angle\_distance}(x, y) = 90° - ||y - x| - 90°|, \tag{4.14}$$

where x and y are the predicted and target orientations represented as angles. Note that this loss function relies on the differentiation of the representation, which can

be difficult or cause numerical stability issues. I found that this form of loss function does not yield better results and abandoned it for the simpler alternative of comparing representation values. I report on the findings of these loss functions in the Experiments Section 5.2 and refer to the losses as Angular MAE and Angular MSE.

4.2 Experiments

In order to train a Deep Learning Model to predict the orientation of features, we can use the aforementioned representations. The model predicts values, which are compared using a loss function to the representation of the target orientation. In my work, I predict the orientation of features in 2D and 3D images. This is done by either predicting a single feature for the whole image, or in a pixel-wise manner, where an orientation is predicted for every pixel in the image. In the latter case, the model can also predict other features, such as a semantic segmentation, as different channels of the output. Each feature may be represented by multiple channels in the network output. The channels of the output of the model are split by feature and a separate loss function is applied for each predicted feature to its respective channels.

The metric used to evaluate the error in the predicted orientation is the angle between the predicted and target vector. This is calculated as follows in equation 4.15:

$$\text{angle\_error}(\mathbf{x}, \mathbf{y}) = \text{acos}\left(\frac{|\mathbf{x} \cdot \mathbf{y}|}{\|\mathbf{x}\|\|\mathbf{y}\|}\right), \tag{4.15}$$

for prediction and target vectors $\mathbf{x}$, $\mathbf{y}$ representing orientation. Note the absolute value of the dot product, which guarantees that the lesser of the two angles is chosen for $\pm\mathbf{x}$ due to the negation invariance of the orientation. This metric can then be reported as either the average or the maximum error. For the average error, the mean across the whole dataset is taken, for every pixel that contains valid orientation annotations. For the maximum error, the maximum error is calculated for every image in the dataset and the resulting value is averaged across the images

in the dataset.

The motivation behind predicting the orientation of features stemmed from the seismic field, to predict the orientation of faults in 3D which can be used to distinguish different faults even across their intersections. To tackle this task, I simplified it to a related task in 2D, to separate overlapping lines in 2D. Separating overlapping objects into individual instances belongs to its own field within Deep Learning called Instance Segmentation. However, in the case of seismic tomographic images, there assumptions that can be made, such as faults always being flat 2D surfaces within the 3D space. In order to exploit these properties, I turned to using orientation as a distinguishing factor between fault instances. I talk about the approaches to separate overlapping instances in Chapter 6.

My first experiments were on the dummy 2D dataset in Section 3.1.1, which consists of images with drawn lines and curves. A model is trained to predict the locations with a curve excluding the areas of intersections. These pixels are also annotated with the orientation of the curve defined as a tangent vector to the curve. Note that any vector and its negative are both valid tangents, where the purpose of a good representation is to deal with this principle. With an abundance of images in the synthetic dataset, these experiments used a simple train/validation/test split of 16:1:1.

I performed further experiments on the 2D orientation representation on the chromosome dataset from Section 3.1.2. It is used in the same way as the dummy 2D dataset, but predicts the orientation of chromosomes instead of random curves.

4.2.1 Network Architecture

The model architecture that I used on the dummy 2D dataset is visualised in Figure 4.7, while the model used for the chromosomes is visualised in Figure 4.8. They are both similar architectures, but differ in the depth and with of the network, since the chromosome dataset is more complex and requires more computation than the dummy dataset. They are convolutional UNet architectures [106] with some modifications. Max-pooling and max-unpooling operations are used for the downsampling and upsampling respectively. The max-pooling operation halves the resolution and

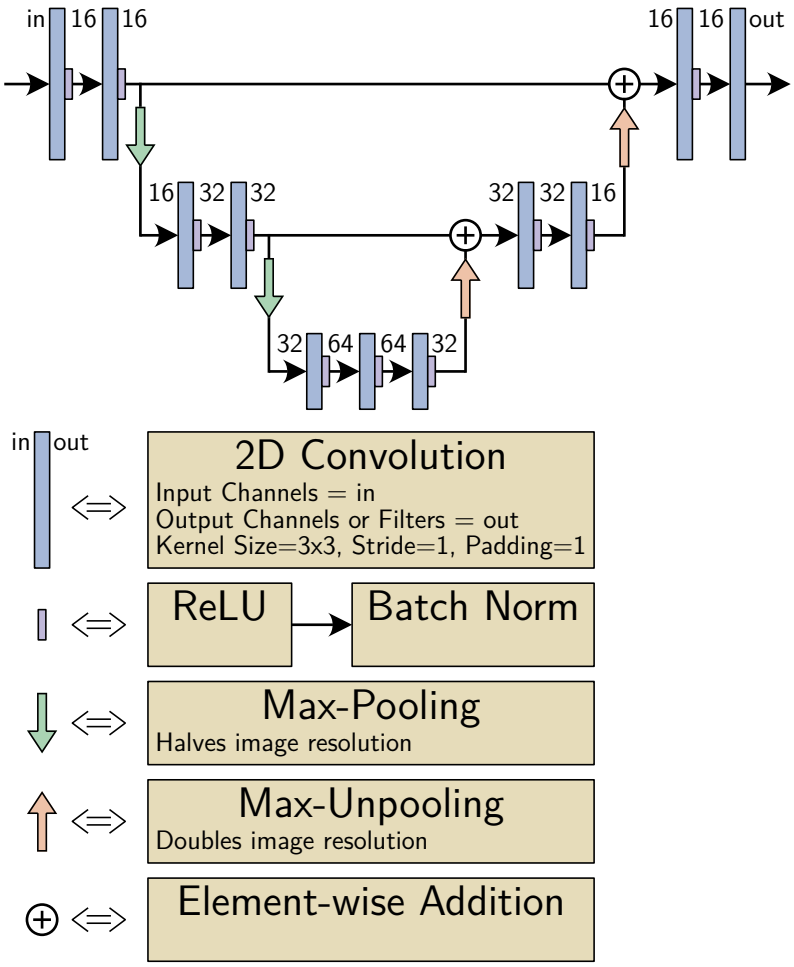


Figure 4.7: Network Architecture used for the experiments on the 2D Dummy Dataset.

saves the indices for which pixel contained the maximal value that is used in the low resolution image. The max-unpooling operation then doubles the resolution and populates the corresponding pixels used in the corresponding pooling operation, leaving the others as zero. This helps to maintain some sub-pixel accuracy despite losing resolution with the pooling-unpooling process. The models use the ReLU activation function and batchnorm as the normalisation layer.

4.2.2 Results

I performed numerous small experiments while designing and implementing the representations, the neural networks, and the 2D dummy dataset itself. Of interest is the final set of experiments that compare the 2D orientation representations with various loss functions. Within this experiment all the models were trained in the

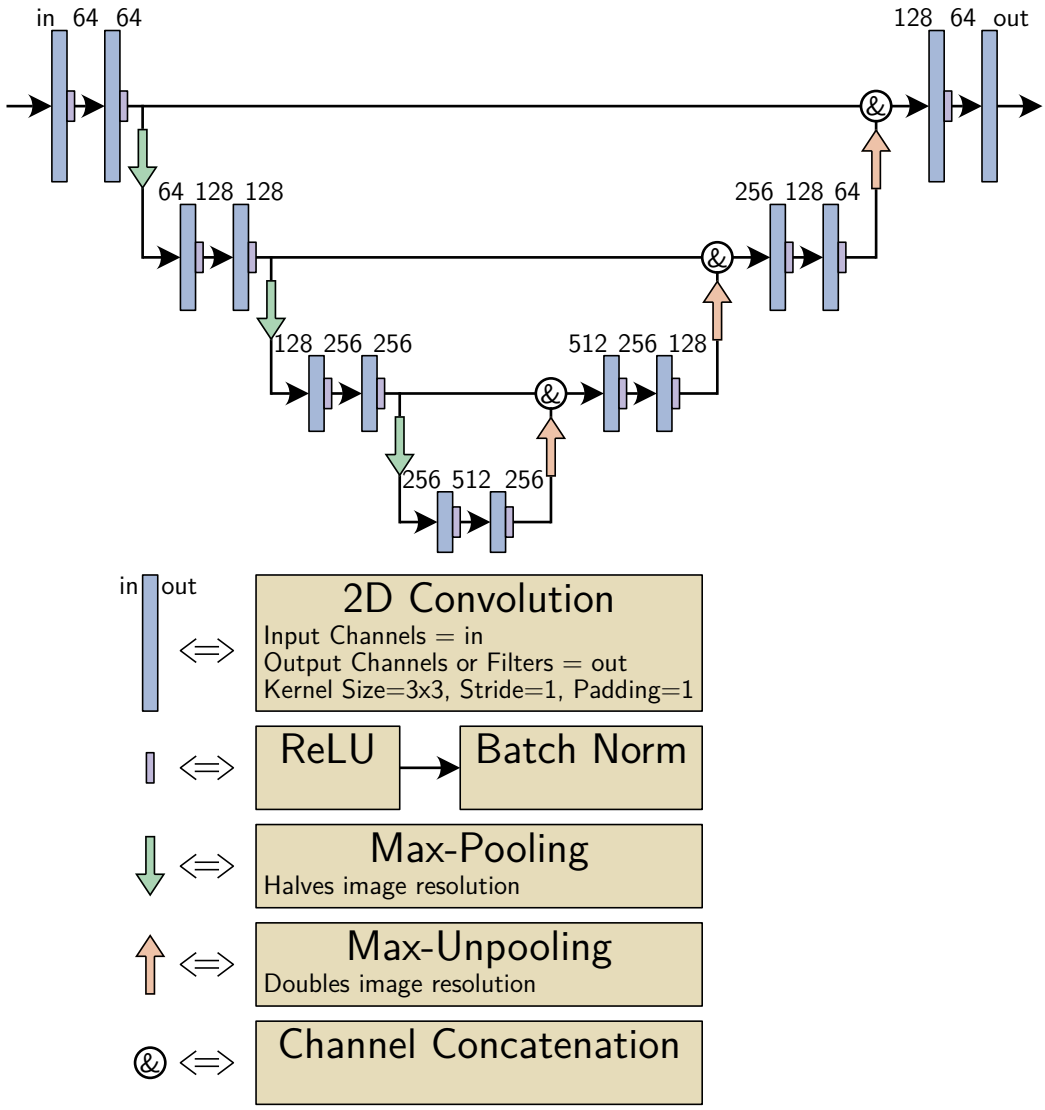


Figure 4.8: Network Architecture used for the experiments on the Chromosome Datasets.

Representation	Loss Function	Mean Angle Error [°]	Max Angle Error [°]
Angle	MAE	4.6	62
Angle	MSE	6.8	75
Vector	MAE	3.6	37
Vector	MSE	4.6	55
Vector	Angular MAE	17	68
Vector	Angular MSE	4.0	31
Doubleangle	MAE	**3.1**	22
Doubleangle	MSE	3.2	21
Doubleangle	Angular MAE	7.1	42
Doubleangle	Angular MSE	3.2	**20**
Piecewise Angle	MAE	4.5	60.7
Piecewise Angle	MSE	6.2	73.6

Table 4.1: Dummy 2D Dataset results. The metrics are reported on the validation set at the end of training.

same way, with the same models, the same training strategy and the same number of epochs. Individual graphs showing how the metrics evolved during training were logged and used to verify that the 32 epochs that were used for training are sufficient for performance to plateau for all models. Of interest are the final metric values at the end of training for each representation in Table 4.1. Note the absence of the Piecewise-Vector Representation, which I hadn't developed yet at the time of these experiments.

The next set of results came from the Synthetic Chromosome Dataset. It offered a more realistic and useful environment to test the orientations in, which has direct applications and led to my publication [7]. The task is in essence the same as the task from the Dummy 2D Dataset and I would expect the same representations to perform well on both datasets. The results are seen in Table 4.2.

Although it is very difficult to evaluate the performance of the representations visually by looking at the predicted images, it provides a useful sanity check. We can verify that there isn't some error in the metrics and that the picture they paint matches with what we see in the predictions. Examples images from the validation set of the Synthetic Chromosome Dataset can be found in Figures 4.9, 4.10, 4.12, 4.13, while Figures 4.11, and 4.14 show an example of a real image of intersecting chromosomes. Figures 4.9, 4.10 and 4.11 show the complete prediction of the model,

Representation	Loss Function	Mean Angle Error [°]	Max Angle Error [°]
Angle	MAE	5.1	59.0
Angle	MSE	6.7	62.7
Vector	MAE	4.9	59.1
Vector	MSE	5.8	60.7
Vector	Angular MAE	4.2	54.7
Vector	Angular MSE	4.9	56.2
Doubleangle	MAE	**4.1**	**46.1**
Doubleangle	MSE	4.5	47.6
Doubleangle	Angular MAE	20.6	64.7
Doubleangle	Angular MSE	4.6	50.2
Piecewise-Angle	MAE	5.6	73.0
PiecewiseAngle	MSE	6.2	74.0
Piecewise-Vector	MAE	7.6	70.1
Piecewise-Vector	MSE	7.3	72.7
Piecewise-Vector adjusted smooth	MAE	4.6	48.8
Piecewise-Vector adjusted smooth	MSE	5.1	51.0
Piecewise-Vector adjusted linear	MAE	4.4	48.2
Piecewise-Vector adjusted linear	MSE	5.1	50.5

Table 4.2: Chromosomes Synthetic Dataset results. The metrics are reported on the validation set at the end of training.

where the predicted orientation is only visualised for the valid points that the model predicted as belonging to a chromosome. In Figures 4.12, 4.13, and 4.14 we can see the raw output of the orientation for every pixel, even the places where the orientation is considered invalid. It is interesting to see the orientations that the model predicts for pixels that do not lie on a chromosome and the number of pixels for which the orientations remain similar outside of the chromosomes. However, we cannot pay too much attention to these values, since the model was never trained to predict them.

4.3 Discussion

The results presented support many of the issues of the representations that I identified in theory. Although the Mean Angle Error is very similar between the representations, the Max Angle Error shows more profound differences between the representations. This is expected, since the discontinuities mainly cause issues for

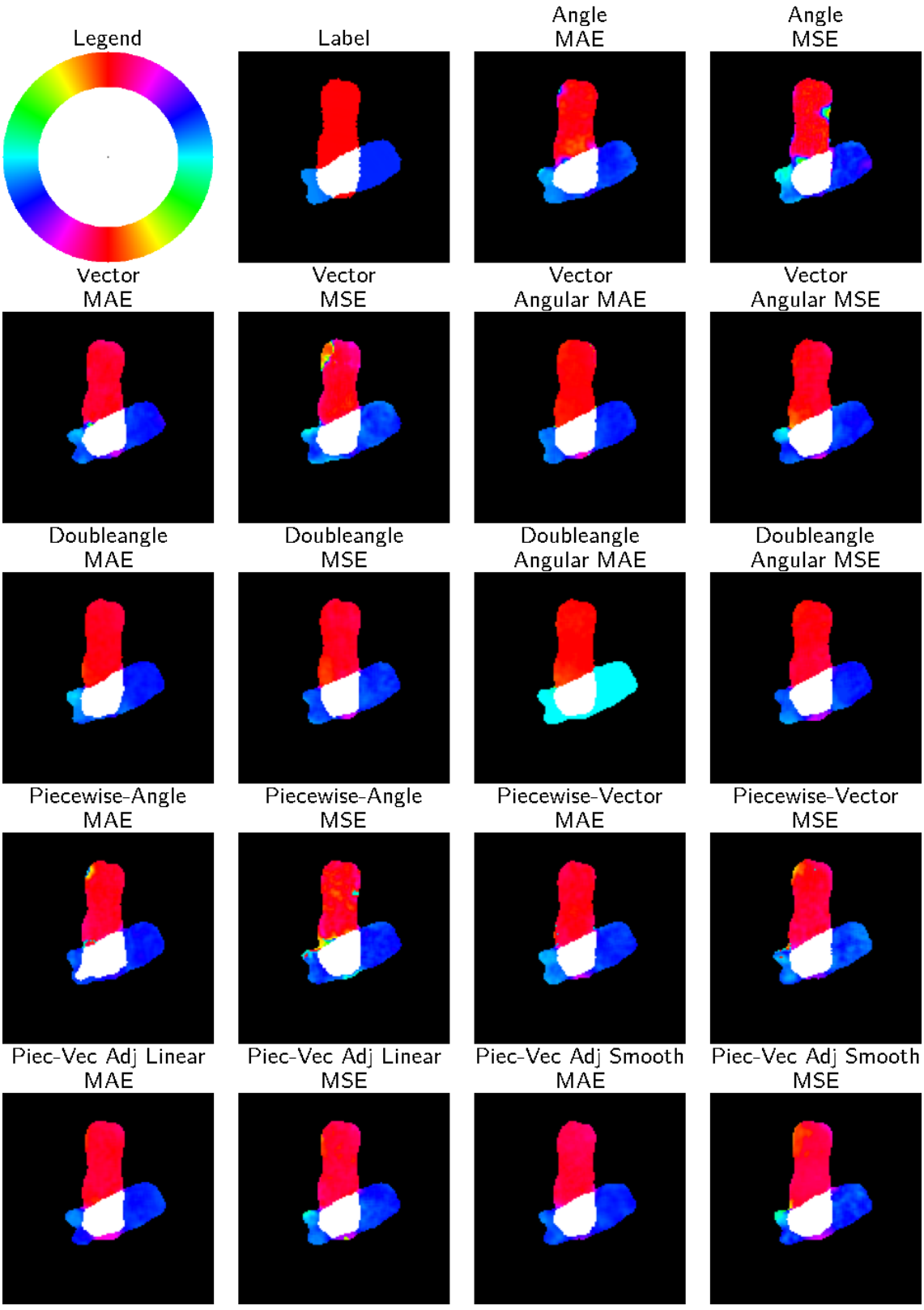


Figure 4.9: Example image from the validation set of the Synthetic Chromosome Dataset. Pixels on a single chromosome are colour-coded for the orientation, while the predicted background is black and the predicted overlap is white.

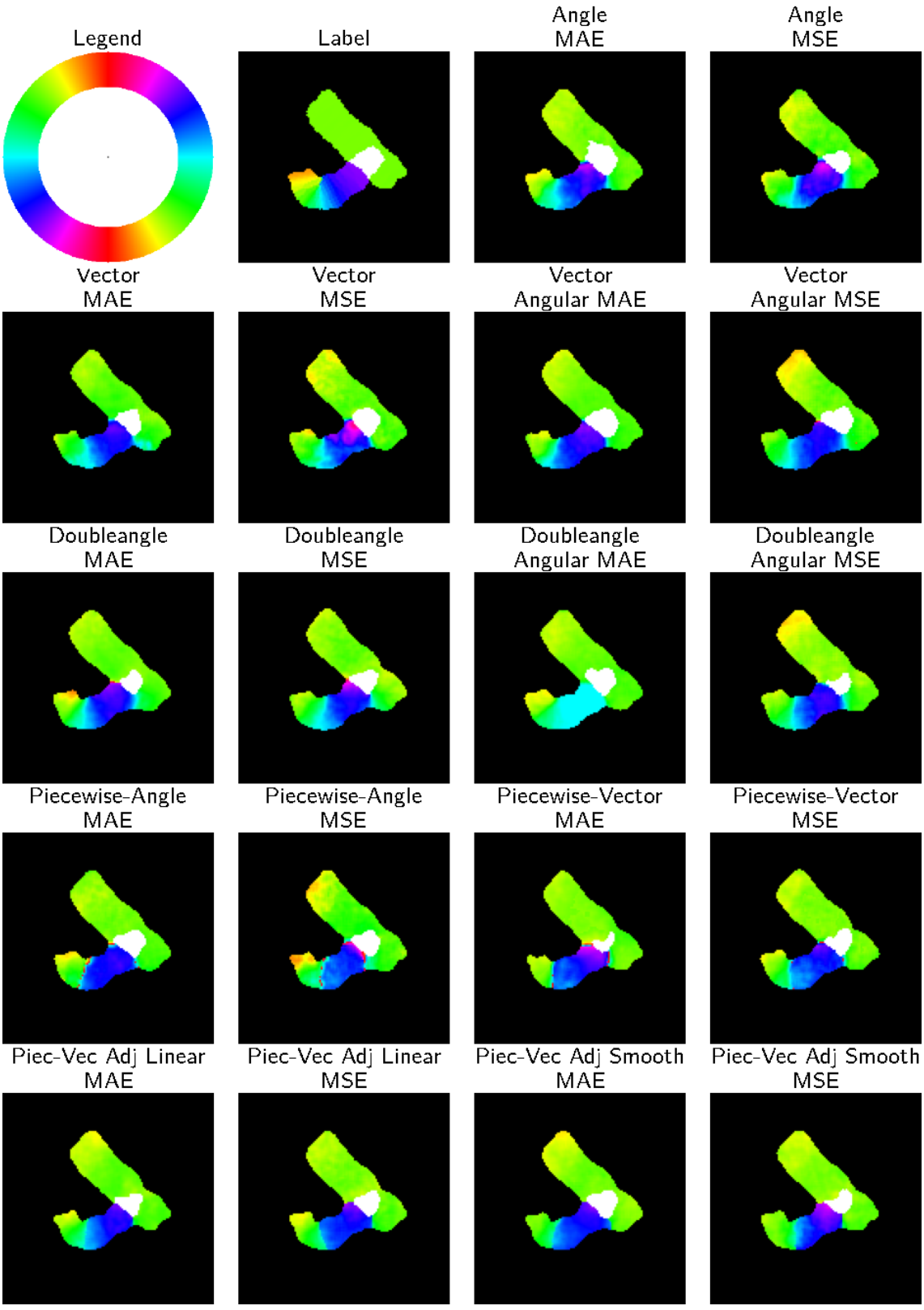


Figure 4.10: Example image from the validation set of the Synthetic Chromosome Dataset. Pixels on a single chromosome are colour-coded for the orientation, while the predicted background is black and the predicted overlap is white.

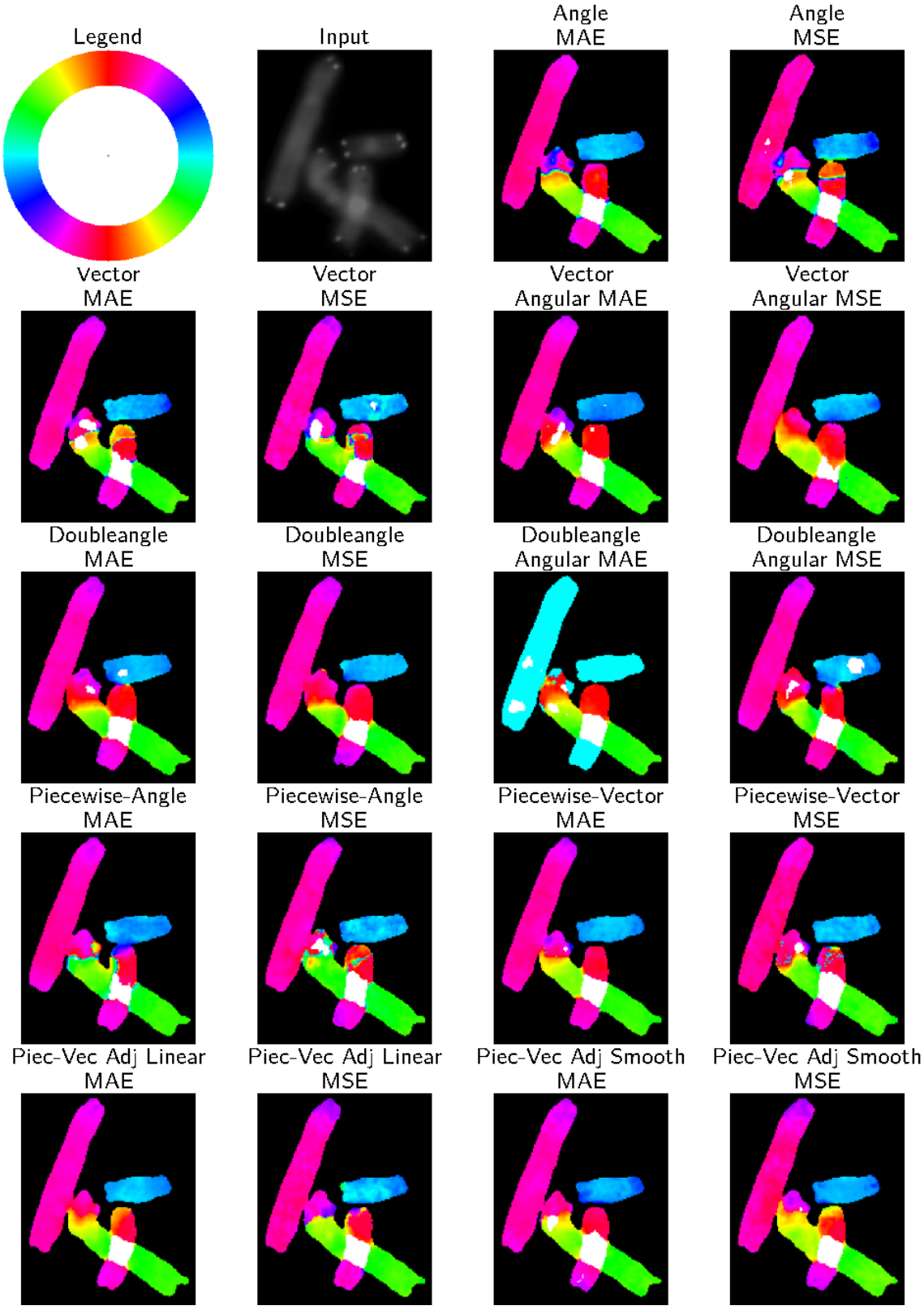


Figure 4.11: Example image of a real (non-synthetic) image of overlapping chromosomes. This image does not have a ground truth. Pixels on a single chromosome are colour-coded for the orientation, while the predicted background is black and the predicted overlap is white.

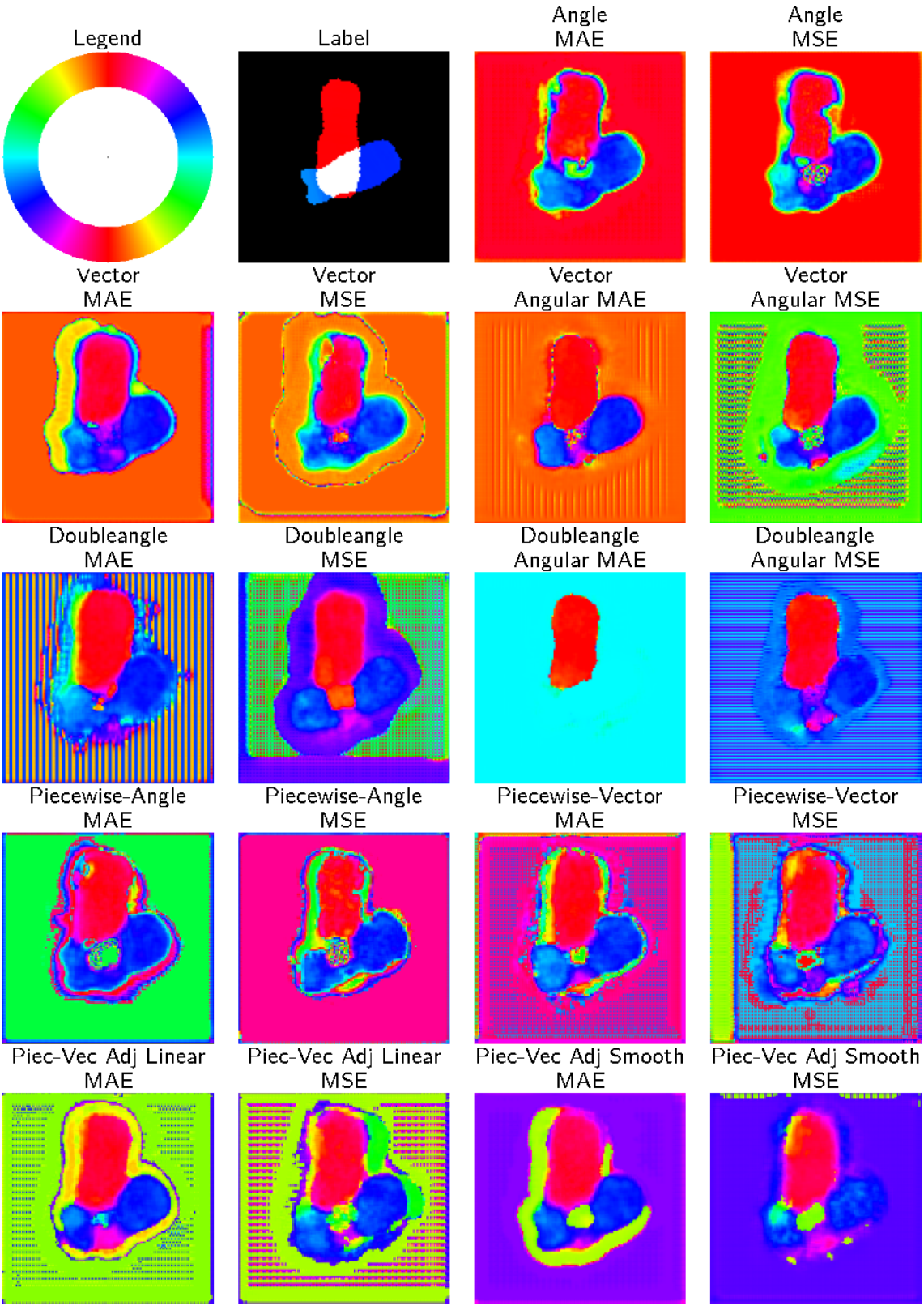


Figure 4.12: Example image from the validation set of the Synthetic Chromosome Dataset. All pixels are colour coded based on the predicted orientation, even the background and overlap pixels, where the values would normally be ignored.

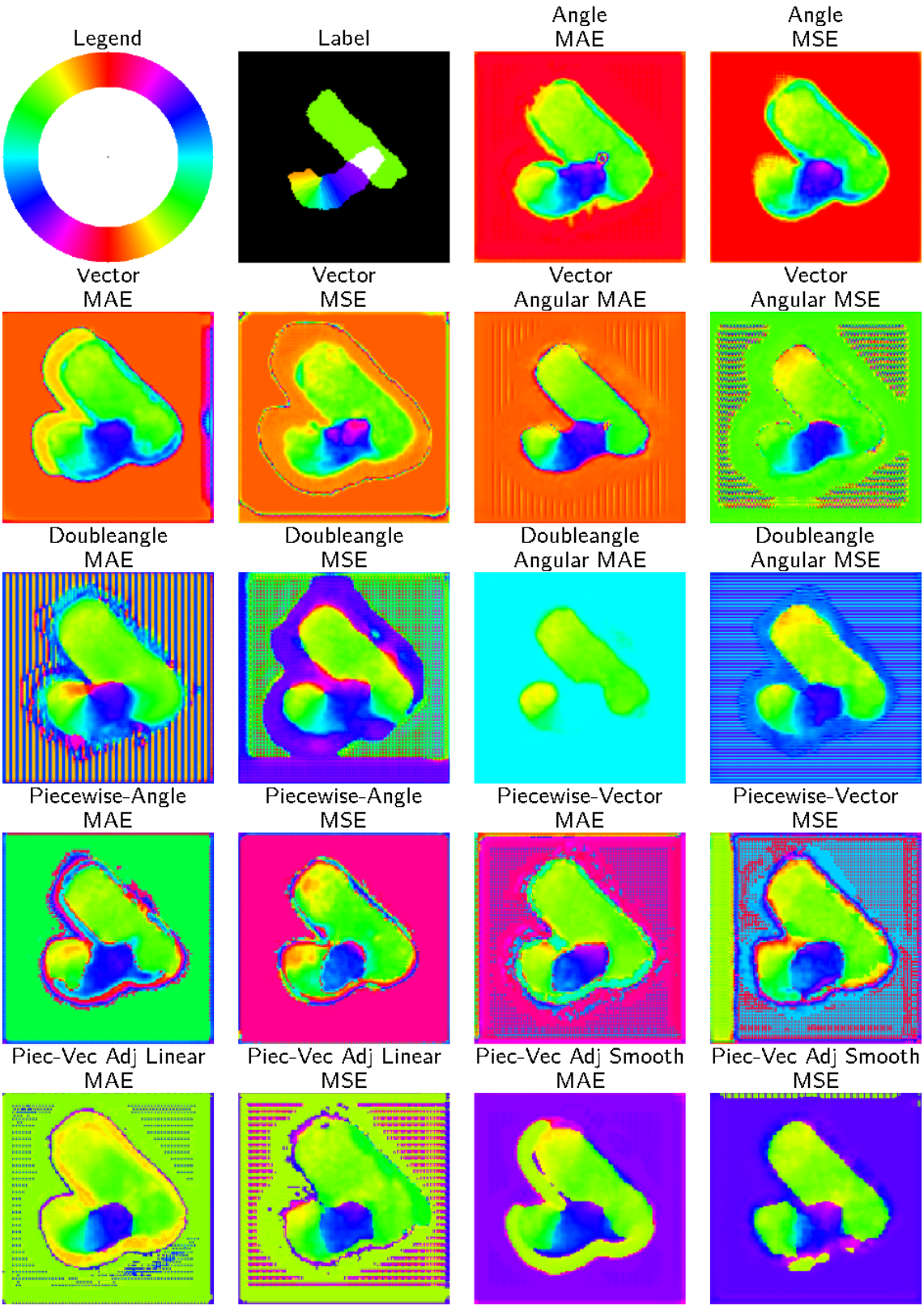


Figure 4.13: Example image from the validation set of the Synthetic Chromosome Dataset. All pixels are colour coded based on the predicted orientation, even the background and overlap pixels, where the values would normally be ignored.

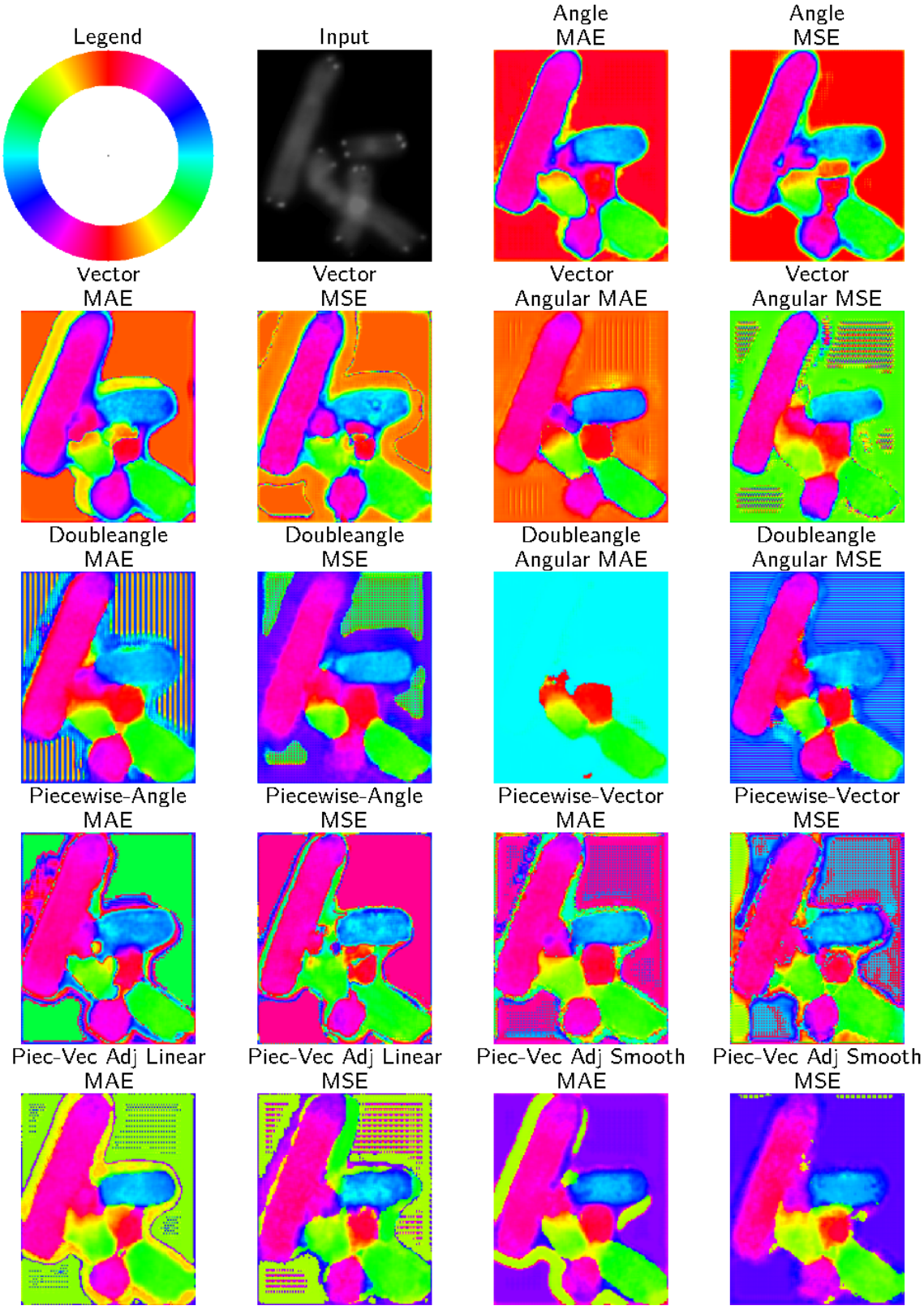


Figure 4.14: Example image of a real (non-synthetic) image of overlapping chromosomes. This image does not have a ground truth. All pixels are colour coded based on the predicted orientation, even the background and overlap pixels, where the values would normally be ignored.

specific orientations near the discontinuity. The rest of the orientations may still be predicted accurately.

We can also see that the performance on the chromosome dataset is worse than the dummy dataset, especially on the max angle errors. This is not surprising, since the dataset and task is much more difficult than with the dummy dataset. Instances where the model struggles to accurately identify chromosomes could cause large errors, especially in the Max Angle Error metric. This makes the metric a little less useful on the chromosome dataset and highlights the utility of the dummy dataset, where the metric highlights the differences the most.

The best representation is clearly the Doubleangle representation. It achieves the lowest errors across both metrics and across both datasets, albeit spread across different loss functions. The MAE loss, MSE loss, and the Angular MSE loss all perform very well, unlike the Angular MAE loss. I will summarise the performance of the angular losses later. These findings are in line with my expectations. The doubleangle representation elegantly avoids any discontinuities, while only requiring a vector of size 2 to represent one orientation.

The next best representations are the Piecewise-Vector Adjusted representations, with either the smooth or linear adjustment. These representations address all of the issues of discontinuities, although the issue remains of switching between the different piecewise predictions. This should not affect the metrics, but it could create pixelation and fuzziness near the orientations where one of the piecewise representations switches to the other, if they do not predict the exact same orientation. This effect is more pronounced with the less accurate piecewise representations, such as the Piecewise-Angle representation in Figure 4.10. Within these representations, the MAE slightly outperforms the MSE loss.

Next, the Vector representation performs reasonably well in the Mean Angle Error, however it struggles in the Max Angle Error metric. This can be explained by errors that occur for orientations near the discontinuity. Surprisingly, on the chromosome dataset, the Angular MAE loss performed well, unlike on the dummy dataset or other datasets. Of the remaining three losses, the MSE performs the worst, especially on the Mean Angle Error metric. This can perhaps be explained

by the MSE penalising large errors much more, which are inevitably present near the discontinuity of the representation. This means that the discontinuity disrupts the model more with the squared losses.

The Angle representation is the next best representation. Although it suffers from a discontinuity, it only requires a vector of size 1 to represent an orientation. This can perhaps explain why it performs better than its Piecewise-Angle counterpart. Similarly to the Vector representation, the MSE performs worse than the MAE, presumably due to the larger losses near the discontinuity.

The Piecewise-Angle representation is the next worst representation. Although it has the potential to avoid the issues with the discontinuity once deploying the model, it still causes issues during training. Similarly to the others, the squared error causes worse performance. We do, however, observe a marginal improvement on the dummy dataset over the non-piecewise Angle representation. This does not translate to the more difficult task of the chromosomes though, where, presumably, the doubling of the size of the vector required to predict and orientation causes a sufficient increase in the complexity of the representation that makes the model suffer.

Lastly, the Piecewise-Vector representation. Out of the representations that have a discontinuity, it has the largest vector size of 4. The combination of these two factors seems to make it perform the worst. Interestingly, however, there is not a big difference between the MSE and MAE models.

Overall, when comparing the loss functions, the models trained with the MSE loss seem to suffer more from discontinuities than when trained using the MAE loss. The representations without discontinuities have much more similar MSE and MAE performances. MAE has a slight edge for the Mean Angle Error though. As for the Angular losses, they do not yield a sufficient improvement in performance to warrant the increased complexity of the loss, which could cause unforeseen problems. Moreover, the Angular MAE loss performs very poorly with one exception on the chromosome dataset on the Vector representation. Overall, the Angular losses are probably best avoided.

4.3.1 Multi-Orientation Representation

I invented another category of representation, which can simultaneously predict multiple distinct orientations, $\{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_n\}$. It could be used to effectively distinguish overlapping elements, such as overlapping chromosomes, since it would be able to continue predicting the orientation the chromosomes in the area of their overlap.

To define the multi-orientation representations we separate orientations into different categories. We can then predict an orientation and a confidence score for each category. All orientation categories that do not contain an orientation are trained to predict a low confidence score, while categories with a valid orientation are trained to predict the orientation and a high confidence score. If an orientation lies close to a boundary point between orientation categories, then both adjacent classes may get a high confidence score while still predicting the same orientation, not two distinct similar orientations. To distinguish this instance we also have to check how similar the predicted orientations are, and only allow sufficiently distant orientations. The condition for allowed simultaneous orientations is then two fold: they must be sufficiently different and fall into distinct categories.

However, this multi-orientation representation is encoded with very large vectors, with the number of bins times the number of features per representation and the confidence score for each bin. As such, its large complexity requires more complex neural networks to predict, which require more data to train on. Although it was functional in my preliminary experiments, I decided not to pursue it further, since it only offered limited utility to the instance segmentation in Chapter 6.

CHAPTER 5

3D Orientation Representation

5.1 Representation Definitions

Creating continuous negation invariant representations for vectors in 3D is much more complicated than for vectors in 2D. If we simply align the vector with one of the axes, ensuring that it is positive, then we are left with a line of discontinuities that need to be addressed rather than just two points. We are also unable to make use of trigonometric functions as effectively in 3D space as in 2D.

I propose 4 novel representations for this purpose which stemmed from different early ideas that are all problematic in different ways. Spherical coordinates inspired the Piecewise-Aligned representation in Section 5.1.2. Projections inspired the Projection-Doubleangle representation in Section 5.1.3. Lastly, classification into discreet bins inspired my continuous Classification Dip-Strike and Classification Icosahedron representations in Section 5.1.4. As it turns out, the Projection-Doubleangle representation is the best representation, although the others also work well.

5.1.1 Exploration of 3D Representations

In this subsection, I will talk about 3D representations in general, issues with simpler naive representations, and the process of arriving at the 3D Piecewise-Vector Representation.

The natural starting point for designing 3D representations is to consider spherical coordinates, as a counterpart to the polar coordinate angle θ that we used in 2D representations. One option would be to use the ISO standard [107]. However, since the application is mostly seismic imaging and faults, I will instead use the dip and strike angles of faults as an equivalent definition of spherical coordinates. Let us define the transformation function of dip (δ) and strike (ϕ) angles of a fault in terms of the normal vector $\mathbf{v}$ to the fault plane as follows in equation 5.1. This is visualised in Figure 5.1:

$$\begin{aligned} \delta(\mathbf{v}) &= \text{acos}(v_z), \\ \phi(\mathbf{v}) &= \text{atan2}(-v_y, v_x), \\ \mathbf{v}(\delta, \phi) &= \begin{bmatrix} \cos(\phi) \cdot \sin(\delta) \\ -\sin(\phi) \cdot \sin(\delta) \\ \cos(\delta) \end{bmatrix}. \end{aligned} \tag{5.1}$$

Note that the only difference between this definition of the dip and strike angles and the ISO spherical coordinates is: The strike (ϕ) is measured clockwise from the x axis, while the ISO spherical coordinates measure an angle counter-clockwise from the x axis.

In order to represent an orientation using the dip and strike angles, we have to halve the domain of either one of the angles: δ or ϕ. This way, the represented points would cover a hemi- unit sphere. Let us consider the two possibilities as follows: Let the "Dip 90° Strike 360°" representation correspond to the halving of the domain for the δ angle with $\delta \in [0°, 90°]$ and $\phi \in [0°, 360°)$. Let the Dip 180° Strike 180° representation correspond to the halving of the domain for the ϕ angle with $\delta \in [0°, 180°]$ and $\phi \in [0°, 180°)$. These representations are shown in Figure 5.2.

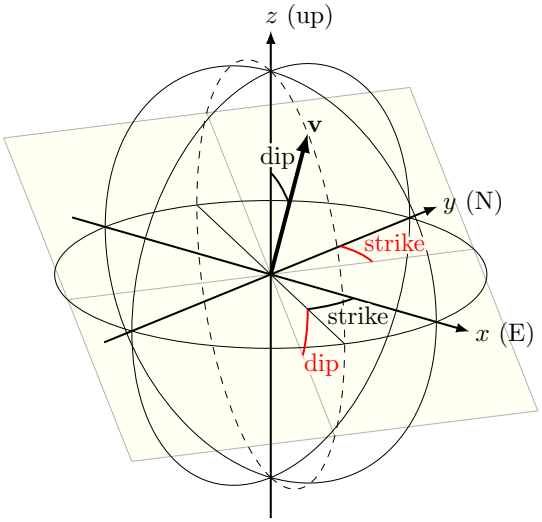


Figure 5.1: Definition of δ (dip) and ϕ (strike) angles. The angles are measured from the plane in red, while the same angles from the normal vector to the plane are seen in black.

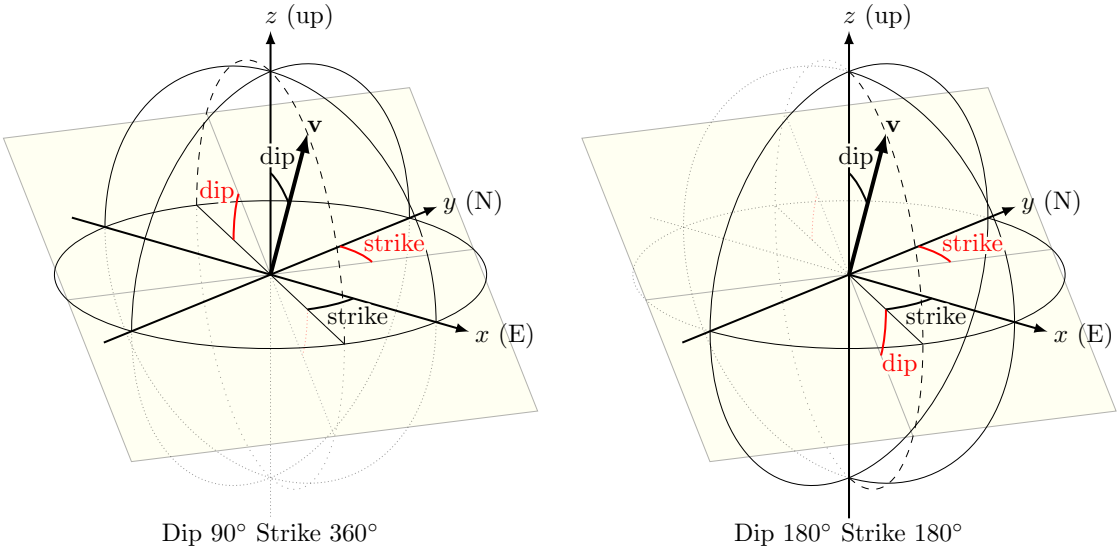


Figure 5.2: Diagram showing the definition of dip and strike angles of a plane defined using its normal unit vector $\mathbf{v} = (x, y, z)$, as $\delta = \text{acos}(z)$ and $\phi = \text{atan2}(x, -y)$. The following two domains are visualised:
(a) $\delta \in [0°, 90°]$, $\phi \in [0°, 360°)$;
(b) $\delta \in [0°, 180°]$, $\phi \in [-90°, 90°)$.

We have to be concerned with any discontinuities that might occur at the points where the angles reach the end of their domain. To eliminate said discontinuities, we can look at using trigonometric functions similarly to the 2D representation counterparts. When considering direction spanning the whole unit sphere

with $\delta \in [0°, 180°)$ and $\phi \in [0°, 360°)$, the problematic discontinuities occur at $(\delta, \phi) = (\delta, 0° + \epsilon) = (\delta, 360° - \epsilon)$ as ϵ approaches 0, for any $\delta \in [0°, 180°]$. When varying δ and keeping ϕ constant, then we get another problematic discontinuity, where $(\delta, \phi) = (-\epsilon, s) = (+\epsilon, s + 180° \mod 360°)$ as ϵ approaches 0. We can eliminate the discontinuity when varying ϕ, by representing it as $(\cos(\phi), \sin(\phi))$. We are still left with the discontinuity at $\delta = 0° = 180°$, which can be addressed by reducing the magnitude of the vector representing ϕ as δ approaches 0° and 180°. Let us do this using the sine of the dip angle to get the following representation for the strike: $(\cos(\phi) \cdot \sin(\delta), \sin(\phi) \cdot \sin(\delta))$. This representation for ϕ would be sufficient to remove all discontinuities, however, we might as well represent δ using $\cos(\delta)$, so that it behaves more similarly to the ϕ. We have now arrived at the Cartesian coordinates from equation 5.1.

We could attempt to use a similar principle for representing the orientation, or negation invariant vectors with the "Dip 90° Strike 360°" and "Dip 180° Strike 180°" coordinates. We can attempt to make use of the principles from the Doubleangle representation counterpart in 2D, but we find that it cannot solve all of our issues.

Let us start by considering Dip 90° Strike 360°. Similarly to when we represented direction, we could eliminate the discontinuities at $\phi = 0° = 360°$ by using $(\cos(\phi), \sin(\phi))$ as a representation for ϕ. We can further eliminate the discontinuity at $\delta = 0°$ by modifying the representation of ϕ to $(\cos(\phi) \cdot \sin(\delta), \sin(\phi) \cdot \sin(\delta))$. However, this time, we are still left with a discontinuities caused by the negation of the vector. Let us first consider the edge case at $\delta = 90°$. More explicitly, $(\delta, \phi) = (90° + \epsilon, s) = (90° - \epsilon, s + 180° \mod 360°)$. The only way to avoid this discontinuity would be to reduce the magnitude of the vector to 0 as it approaches $\delta = 90°$. However, in this way it would be impossible to distinguish the orientations at $(\delta, \phi) = (90°, s)$ for any $s \in [0°, 360°)$. To address this issues, we could move to a piecewise representation. Nonetheless, I also performed experiments with the imperfect representation that has discontinuities at $\delta = 90°$ that is defined with the

following equation 5.2, which I refer to as the Dip 90° Strike 360° representation:

$$
\begin{aligned}
r &: \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3, \\
r(\delta, \phi) &= \begin{bmatrix} \frac{\delta}{45^\circ} - 1 \\ \cos(\phi) \\ \sin(\phi) \end{bmatrix}, \\
r^{-1}(\mathbf{u}) &= \Big(\delta, \phi\Big) = \Big((u_0 + 1) \cdot 45^\circ,\ \text{atan2}(u_2, u_1)\Big),
\end{aligned}
\tag{5.2}
$$

where we normalise δ to the range $[-1, 1]$.

Before exploring the piecewise representation, let us consider Dip 180° Strike 180°. We could use the Doubleangle representation from equation 4.2 to represent ϕ. This way, we seemingly remove all discontinuities at $\phi = 0^\circ = 180^\circ$ and the discontinuity at $\delta = 0^\circ$ and $\delta = 180^\circ$ also seemingly disappears when compared to the direction representation, where we had $(\delta, \phi) = (-\epsilon, s) = (+\epsilon, s + 180^\circ \mod 360^\circ)$ as ϵ approaches 0, since our ϕ is now 180° periodic. However, the way we achieved 180° periodicity is by multiplying the vector by -1, so that it always appears in the hemisphere that we want. We therefore get the following relationship instead, which is still discontinuous: $(\delta, \phi) = (d, 0^\circ + \epsilon) = (180^\circ - d, 180^\circ - \epsilon)$ as ϵ approaches 0. The following equation 5.3 is the definition of this Dip 180° Strike 180° representation that I use in my experiments:

$$
\begin{aligned}
r &: \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3, \\
r(\delta, \phi) &= \begin{bmatrix} \frac{\delta}{90^\circ} - 1 \\ \cos(2 \cdot \phi) \\ \sin(2 \cdot \phi) \end{bmatrix}, \\
r^{-1}(\mathbf{u}) &= \Big(\delta, \phi\Big) = \Big((u_0 + 1) \cdot 90^\circ,\ \frac{\text{atan2}(u_2, u_1)}{2}\Big).
\end{aligned}
\tag{5.3}
$$

Let us now consider piecewise representations as a continuation from Dip 90° Strike 360°. The representation $(\delta, \cos(\phi) \cdot \sin(\delta), \sin(\phi) \cdot \sin(\delta))$ suffers from discontinuities at $\delta = 90^\circ$ in the other two dimensions. If we further adjusted the representation to go to zero near $\delta = 90$, then we would avoid discontinuities. We would then also need to come up with additional representations that we could use

near $\delta = 90°$, which is more difficult. Let us first rewrite the current representation in terms of the Cartesian vector $\mathbf{v} = (x, y, z)$. We can write $\sin(\delta) = \sqrt{x^2 + y^2}$, $sin(\phi) = \frac{-y}{\sqrt{x^2+y^2}}$ and $sin(\phi) = \frac{x}{\sqrt{x^2+y^2}}$. We can therefore rewrite the representation as $(\delta, -y, x)$. If we also rewrote δ as $\cos(\delta) = z$, then we arrive at effectively just having the Cartesian vector $\mathbf{v} = (x, y, z)$ directly, with $x, y \in [-1, 1]$ and $z \in [0, 1]$. This makes it easier to consider the adjustment functions that we used in the 2D piecewise representation in Section 4.1.5. Specifically, let us consider the adjustment function from the 3rd iteration and equation 4.13. In 3-dimensions, we get the following representation in equation 5.4, when aligned with the z-axis.

$$\text{sign}(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}, \tag{5.4}$$
$$r(\mathbf{v}) = \text{sign}(v_z) \cdot v_z^2 \cdot \mathbf{v}.$$

This representation is fully continuous, however, it is the zero vector for all vectors $\mathbf{v}$ with $z = 0$.

The remaining question is what representation can we use for vectors with $z = 0$ in a piecewise manner? I could not find a singular representation for those points, however, the combination of two representations satisfies this constraint. If we reused the same representation from equation 5.4, but aligned it with the x-axis for the second representation, and the y-axis for the third. When we align with any axis $a \in x, y, z$, then the representation is equal to $\mathbf{0}$ (the zero vector) for values with $v_a = 0$. Any vector $\mathbf{v}$ on the unit sphere that we are trying to represent satisfies the property $v_x^2 + v_y^2 + v_z^2 = 1$ and therefore at most two dimensions out of x, y, z are close to 0. In other words, we could rely on the representation that is aligned with the axis a that has the largest v_a. This brings us to the definition of the representation.

5.1.2 Piecewise-Vector Representation

a.k.a. Piecewise-Aligned Representation [5]

We can write this representation as follows in equation 5.5:

$$\begin{aligned} \text{sign}(a) &= \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a \leq 0 \end{cases}, \\ \forall a &\in \{0,1,2\} : \\ s_a(\mathbf{v}) &= \text{sign}(v_a) \cdot v_a^2 \cdot \mathbf{v}, \\ r &: \mathbb{R}^3 \to \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3, \\ r(\mathbf{v}) &= \Big(s_0(\mathbf{v}), s_1(\mathbf{v}), s_2(\mathbf{v})\Big). \end{aligned} \tag{5.5}$$

In fact, the same principle from the 3rd iteration of the 2-dimensional piecewise representation also holds in three dimensions, namely: the magnitudes of the three parts $r(\mathbf{v})_0, r(\mathbf{v})_1, r(\mathbf{v})_2$ add up to $v_0^2 + v_1^2 + v_2^2 = 1$. Therefore, all we have to do to invert the representation is to find an axis to align all three vectors $r(\mathbf{v})_0, r(\mathbf{v})_1, r(\mathbf{v})_2$ to and add them up. We do this by selecting the axis b for which the vector $r(\mathbf{v})_b$ has the greatest magnitude $\|r(\mathbf{v})_b\|$, which is equivalent to the largest $r(\mathbf{v})_{b,b}^2$, which is also equivalent to the largest $|r(\mathbf{v})_{b,b}^2|$. We can express the inverse of this representation as follows, in equation 5.6:

$$\begin{aligned} r^{-1} &: \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3, \\ \text{Let } b \in \{0,1,2\} \quad &\text{s.t.} \quad \max(\|\mathbf{u}_0\|, \|\mathbf{u}_1\|, \|\mathbf{u}_2\|) = \|\mathbf{u}_b\|, \\ r^{-1}(\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2) &= \sum_{a \in 0,1,2} (\text{sign}(u_{a,b}) \cdot \mathbf{u}_a). \end{aligned} \tag{5.6}$$

Note that this representation turned out to be a direct expansion of the 2D-Piecewise counterpart into three dimensions. In fact, it is possible to expand this representation to represent vectors $\mathbf{v}$ of any number of dimensions n. The resulting representation $\mathbf{r}$ would require n^2 dimensions. It is defined as follows, in equation

5.7:

$$
\begin{aligned}
\text{sign}(a) &= \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a \leq 0 \end{cases}, \\
\forall a &\in \{0, 1, ..., n-1\} : \\
s_a(\mathbf{v}) &= \text{sign}(v_a) \cdot v_a^2 \cdot \mathbf{v}, \\
r &: \mathbb{R}^n \to (\mathbb{R}^n)^n, \\
r(\mathbf{v}) &= \Big(s_0(\mathbf{v}), s_1(\mathbf{v}), ..., s_{n-1}(\mathbf{v})\Big), \\
\text{Let } b \in \{0, 1, ..., n-1\} \quad \text{s.t.} \quad & \max_{a \in \{0,1,...,n-1\}} (\parallel s_a(\mathbf{v}) \parallel) = \parallel s_b(\mathbf{v}) \parallel, \\
r^{-1}(\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{n-1}) &= \sum_{a \in \{0,1,...,n-1\}} (\text{sign}(u_{a,b}) \cdot \mathbf{u}_a).
\end{aligned}
\tag{5.7}
$$

I will also refer to this representation as the Piecewise-Aligned representation in 3D.

5.1.3 Projection-Doubleanlgle Representation

Another fundamental approach that we could consider is to look at spherical projections. To represent the orientation, we need to map the points on a hemisphere, in a continuous manner. Let us consider the $z >= 0$ hemi unit sphere, which is equivalent to Dip 90° Strike 360°. Unlike the previous representations that used 3-dimensions, 2 are sufficient to represent the surface of the sphere. The simplest projection we could consider is using parallel perspective, where we only take the x and y coordinates of the Cartesian coordinates of the vector. However, we arrive at problems near $z = 0$, where $x^2 + y^2 = 1$. Not only do we find a discontinuity, where $(x, y) = (-x, -y)$, where $x^2 + y^2 = 1$, but a small change in the magnitude of the vector has a large change of the resulting value of the z-axis, unlike the points where $x^2 + y^2$ is closer to 0. To address this issue, we could instead look at using

the following angles in equation 5.8:

$$\begin{aligned}\theta_{yz} &= \text{atan2}(y, z) = \text{asin}(\frac{y}{\sqrt{1-x^2)}}),\\ \theta_{xz} &= \text{atan2}(x, z) = \text{asin}(\frac{x}{\sqrt{1-y^2}}),\end{aligned} \tag{5.8}$$

with $\theta_{yz}, \theta_{xz} \in (-90°, 90°)$. These angles measure the x and y components from the z axis. We can also think of them as projections of the vector v onto the yz and xz planes, where we measure the angle. The notation used for θ_{yz} means that the vector was projected onto the yz plane where $x = 0$. Unfortunately, these angles are poorly defined for vectors close to $z = 0$. However, vectors close to $z = 0$ could be distinguished using a thrid angle, the ϕ angle, or equivalently, the projection onto the xy plane θ_{xy}, as seen in Figure 5.3. Together, these three angles define the vector location well. We are now faced with two problems: Firstly, how do we define the angles for vectors with one axis element equal to zero? Secondly, how can this representation be adjusted to represent orientation, or negation invariant vectors?

We cannot easily address the first question yet, but we will be able to once we address the second question. We could use the Doubleangle representation from equation 4.2 for the angles $\theta_{yz}, \theta_{xz}, \theta_{xy}$. My initial intuition was to only use the doubleangle representation for two angles θ_{yz} and θ_{xz}, while leaving $\theta_{xy} \in [0°, 360°)$, however, we will find that representing all three angles in the range $[0°, 180°)$ using the Doubleangle representation leaves us with sufficient information to invert the vector.

Now that we use the Doubleangle representation to represent the angles $\theta_{yz}, \theta_{xz}, \theta_{xy}$, we are free to modify the magnitude of each of these vectors. We want to make use of this to address the first question, and taper the vector to zero as it approaches the ill-defined region. To do this, we multiply the Doubleangle representation by the magnitude of the projected vector: $\sqrt{a^2 + b^2}$ for θ_{ab}. We can do this automatically when using the Doubleangle representation from equation 4.3 to directly express the projection of $\mathbf{v}$, rather than first finding the angles.

Overall, this leads to the following definition of the representation: First, in terms of the angles $\theta_{yz}, \theta_{xz}, \theta_{xy}$ in equation 5.9. Second, without trigonometry directly via

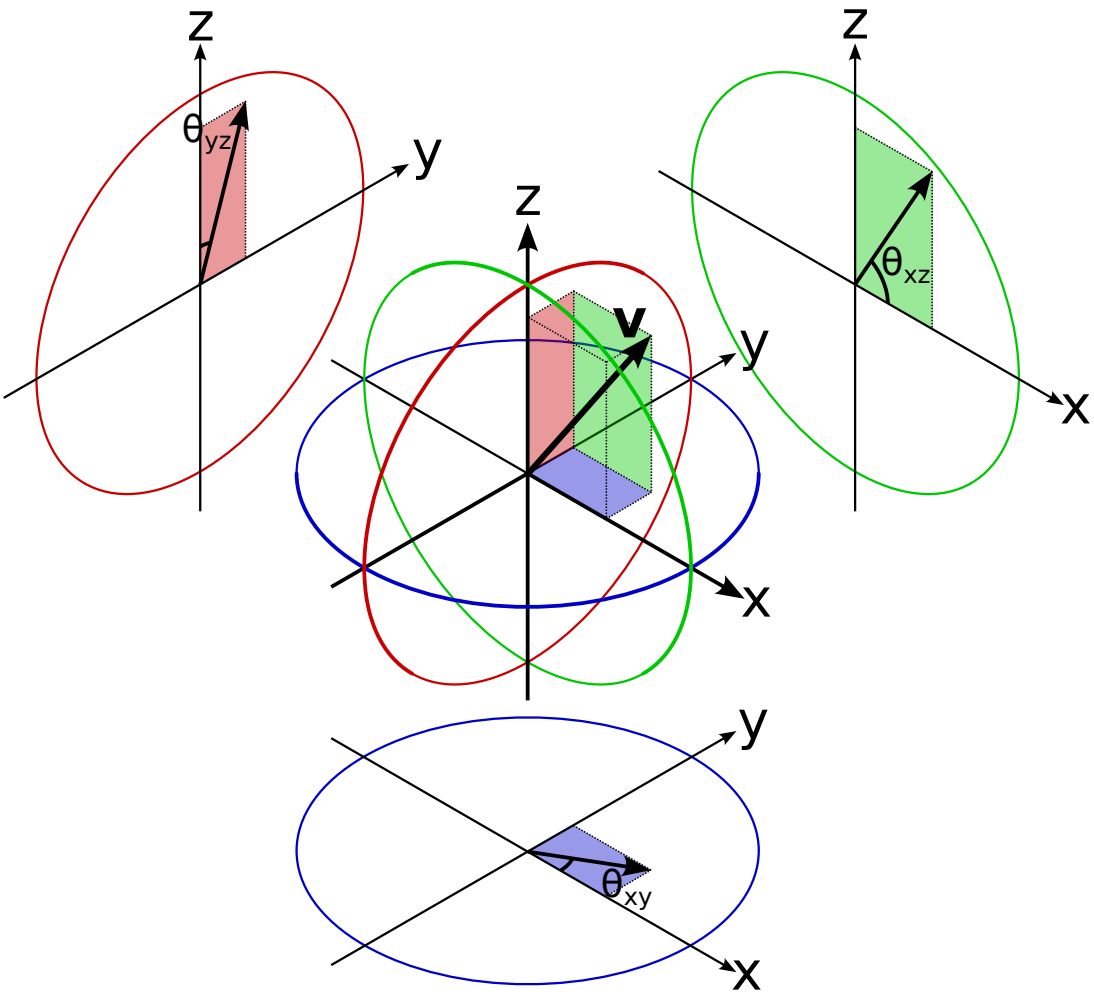


Figure 5.3: Diagram showing the angles $\theta_{xy}, \theta_{xz}, \theta_{yz}$ of the projection of the vector $\mathbf{v}$ onto the axis-planes. These angles are used in the Projection-Doubleangle representation.

the projected Cartesian coordinates in equation 5.10.

$$
\begin{aligned}
&\forall ab \in \{xz, yz, xy\} : \\
&\theta_{ab}(\mathbf{v}) = \text{atan2}(v_b, v_a), \\
&t_{ab}(\mathbf{v}) = \sqrt{v_a^2 + v_b^2} \begin{bmatrix} \cos(2 \cdot \theta_{ab}(\mathbf{v}) \\ \sin(2 \cdot \theta_{ab}(\mathbf{v}) \end{bmatrix}, \\
&r : \mathbb{R}^3 \to \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3, \\
&r(\mathbf{v}) = \Big(t_{yz}(\mathbf{v}),\ t_{xz}(\mathbf{v}),\ t_{xy}(\mathbf{v}) \Big),
\end{aligned}
\tag{5.9}
$$

and

$$
\begin{aligned}
&\forall ab \in \{xz, yz, xy\} : \\
&t_{ab}(\mathbf{v}) = \frac{1}{\sqrt{v_a^2 + v_b^2}} \cdot \begin{bmatrix} v_a^2 - v_b^2 \\ 2 v_a^2 v_b^2 \end{bmatrix}, \\
&r : \mathbb{R}^3 \to \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3, \\
&r(\mathbf{v}) = \Big(t_{yz}(\mathbf{v}),\ t_{xz}(\mathbf{v}),\ t_{xy}(\mathbf{v}) \Big),
\end{aligned}
\tag{5.10}
$$

Next, we have to consider how we could invert this representation. We can invert the doubleangle representation to recreate the three projected vectors $\mathbf{p}_{yz}, \mathbf{p}_{xz}, \mathbf{p}_{xy}$ using the following equation 5.11

$$
\begin{aligned}
&\forall ab \in \{yz, xz, xz\} : \\
&p_{ab}(\mathbf{u}) = \sqrt{u_a^2 + u_b^2} \begin{bmatrix} \sqrt{\frac{1}{2} + \frac{u_a}{2\sqrt{u_a^2 + u_b^2}}} \\ \text{sign}(u_b) \sqrt{\frac{1}{2} - \frac{u_a}{2\sqrt{u_a^2 + u_b^2}}} \end{bmatrix},
\end{aligned}
\tag{5.11}
$$

which makes use of the equation 4.3.

Crucially, any of these vectors could be multiplied by ± 1, due to the negation-invariant property of the Doubleangle representation. We can now compare the 6 values from these 3 projected vector, to deduce which should be multiplied by -1. If we knew these values of ± 1, then we could invert the representation using the

following equation 5.12:

$$r^{-1}(\mathbf{u}) = \begin{bmatrix} \frac{1}{2} \cdot (s_{xz} \cdot p_{xz}(\mathbf{u})_0 + s_{xy} \cdot p_{xy}(\mathbf{u})_0) \\ \frac{1}{2} \cdot (s_{yz} \cdot p_{yz}(\mathbf{u})_0 + s_{xy} \cdot p_{xy}(\mathbf{u})_1) \\ \frac{1}{2} \cdot (s_{yz} \cdot p_{yz}(\mathbf{u})_1 + s_{xz} \cdot p_{xz}(\mathbf{u})_1) \end{bmatrix}, \tag{5.12}$$

where $s_{yz}, s_{xz}, s_{xy} \in \{-1, 1\}$.

Now, we should be able to set the values of s_{yz}, s_{xz}, s_{xy} such that for every row in equation 5.12 the two elements have the same sign. We could arbitrarily set the value of one of the three signs s_{yz}, s_{xz}, s_{xy}, because we don't care if our resulting vector is multiplied by ± 1, and deduce the value of the other two s variables. However, we must also account for possible inaccuracies of the prediction of any neural network used to predict these values. Especially when one of the values of $\mathbf{v}$ is close to 0, a small error in the representation could cause its sign to change. If that does happen, then we want the mismatched signs to be in the row of equation 5.12 with the smallest values. We determine the smallest row using the following equation 5.13:

$$m(\mathbf{u}) = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} |p_{xz}(\mathbf{u})_0| + |p_{xy}(\mathbf{u})_0| \\ |p_{yz}(\mathbf{u})_0| + |p_{xy}(\mathbf{u})_1| \\ |p_{yz}(\mathbf{u})_1| + |p_{xz}(\mathbf{u})_1| \end{bmatrix}, \tag{5.13}$$

and let $k \in \{x, y, z\}$ s.t. $\min(m_x, m_y, m_z) = m_k$.

We can now determine the values of s_{yz}, s_{xz}, s_{xy} using the following equation 5.14:

$$s_{ab} = \begin{cases} \begin{cases} 1 & \text{if } \operatorname{sign}(p_{ab}(\mathbf{u})_1) = \operatorname{sign}(p_{bc}(\mathbf{u})_0) \\ -1 & \text{if } \operatorname{sign}(p_{ab}(\mathbf{u})_1) \neq \operatorname{sign}(p_{bc}(\mathbf{u})_0) \end{cases} & \text{if } k = a \\ \begin{cases} 1 & \text{if } \operatorname{sign}(p_{ab}(\mathbf{u})_0) = \operatorname{sign}(p_{ac}(\mathbf{u})_0) \\ -1 & \text{if } \operatorname{sign}(p_{ab}(\mathbf{u})_0) \neq \operatorname{sign}(p_{ac}(\mathbf{u})_0) \end{cases} & \text{if } k = b \\ 1 & & \text{if } k = c \end{cases}. \tag{5.14}$$

To explain this equation 5.14, let us consider an example where $k = z$ is the axis

with the smallest elements. We would therefore set $s_{xy} = 1$ and deduce s_{yz} such that the two values forming v_x, which are $p_{xz}(\mathbf{u})_0$ and $p_{xy}(\mathbf{u})_0$, have the same sign. We would also set s_{xz} such that the two values forming v_y, which are $p_{yz}(\mathbf{u})_0$ and $p_{xy}(\mathbf{u})_1$, have the same sign. However, we could still get $s_{yz} \cdot p_{yz}(\mathbf{u}_1)$ and $s_{xz} \cdot p_{xz}(\mathbf{u})_1$ with different signs. This is the desired behaviour, since we selected the axis $k = z$ that has the smallest values and to minimise the error caused by this sign mismatch.

This Projection-Doubleangle representation is now fully defined together with its inverse using equations 5.10, 5.11, 5.12, 5.13, 5.14. It is fully continuous and requires 6 dimensions for the representation. In my empirical evaluation, I found this representation to be the most effective.

5.1.4 Classification Representations

Classification-style representations came about as an extension of a standard classification task to a continuous space. Consider a classification task, where we quantise the possible orientations into distinct and mutually exclusive categories. We could then use a neural network to predict what category or bin any given orientation belongs to. This approach is something that Wu et al. used [108], which inspired me to pursue this direction further. The obvious shortcoming of this method is its discrete nature, in which we cannot distinguish between different orientations within the same category. Therefore, the granularity of the representation is determined by the number of categories that we use. However, the larger the number of categories, the more computationally expensive the representation becomes. This size of the representation is especially problematic in application where a large number of orientations are predicted, such as predicting an orientation for every voxel in 3D space. Nevertheless, if the task is difficult and we cannot expect a high accuracy from the model, then a small number of categories may be appropriate, making this approach feasible.

However, consider the case where an orientation is on the boundary between two bins. We would expect the model to give a higher probability to both adjacent bins. Could we rely on the probability of adjacent categories to determine a more exact orientation within the category? It turns out that this is possible if we train the

model to predict very specific category probabilities based on the exact orientation.

1D Continuous Classification

Consider an example using a one-dimensional vector $v \in [0, 1)$. We could define 10 bins: $i \in \{0, 1, ..., 9\}$, where each bin i represents values of v in the range $[0.1 \cdot i, 0.1 \cdot (i + 1)]$. We could then say that these bins i are centred around the value $c_i = 0.1 \cdot i + 0.05$, which is $(0.05, 0.15, ..., 0.95)$. If we wanted to represent the value that sits exactly on the centre of a bin, such as $v = 0.15$ for bin 1 with $c_1 = 0.15$, then bin 1 gets an assignment or probability of $r(\mathbf{v})_1 = 1$, while all other categories get $r(\mathbf{v})_{i\neq 1} = 0$. To represent a value on the boundary between categories, such as $v = 0.2$, we would assign a probability of $r(\mathbf{v})_1 = r(\mathbf{v})_2 = 0.5$ for both bins 1 and 2. For any values $v \in [0.15, 0.25]$, $r(\mathbf{v})_1$ would be assigned the probability of $10 \cdot (0.25 - v)$, and $r(\mathbf{v})_2$ a probability of $10 \cdot (v - 0.15)$. Note that these two values add up to 1. The last consideration we have to make is how to treat edge values. In its current state, it is not possible to fully represent values below 0.05 or above 0.95, because there is no further adjacent bin at the extremes past c_0 and c_9. To fully represent the numbers $v \in [0, 1]$, we need to use 11 categories that are centred on $(0, 0.1, ..., 1)$.

Let us formalise this representation in one dimension. Consider a value v that we want to represent in the range $v \in [b_l, b_u]$ for any real values $b_l, b_u \in \mathbb{R}$. If we subdivide this numberline into n bins for a standard classification, then we need $n+1$ categories for this continuous classification. The width of a bin is $w = (b_u - b_l)/n$. Each category $i \in \mathbb{N} \cap [0, n + 1)$ is centred around the point $c_i = b_l + w \cdot i$. Lastly, the probability value assigned to a category i is defined as:

$$r_i(v) = \max(0, 1 - \frac{|c_i - v|}{w}) = \max(0, 1 - \frac{|r_l + w \cdot i - v|}{w}). \tag{5.15}$$

This equation can be explained as the distance between the point v and the centre of the category c_i in terms of the number of widths of the bins w. However, we want the probability to be 1 when the distance is zero, and zero for any distances larger than w, which is achieved using the expression $\max(0, 1 - x)$. You can see

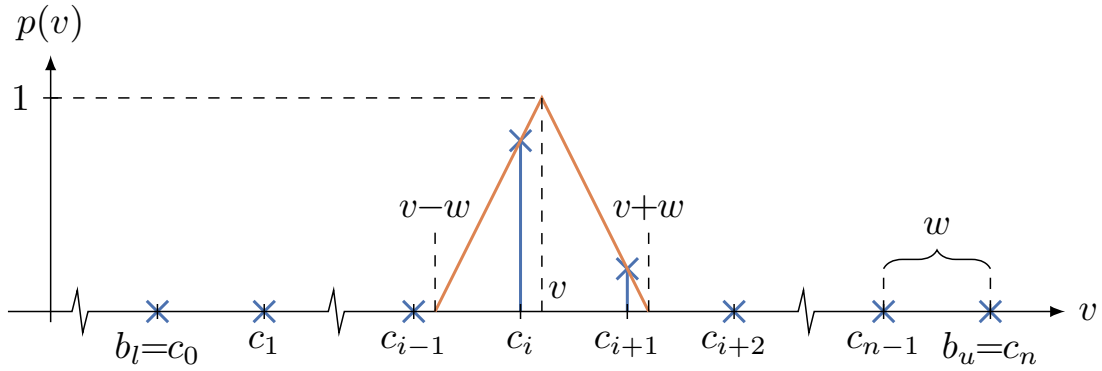


Figure 5.4: Graph showing the continuous classification in one dimension from equation 5.15. The points in blue correspond to the probabilities r_i assigned to the categories i that are centred around points c_i. The line in orange shows us the equation 5.15 for a fixed point v and points c_i varying along the v-axis.

this representation visualised in Figure 5.4.

Let us now consider how we could invert this representation. Even if we train a neural network to predict the probabilities $r(\mathbf{v})_i$ exactly, we still have to expect some error. Only two classes should ever have probabilities higher than zero. Therefore, let us find which two adjacent classes have the highest sum of probabilities in equation 5.16:

$$\begin{aligned} &\text{Find } j \in \mathbb{N} \cap [0, n) \quad \text{s.t.} \quad : \\ &\max_{i \in \mathbb{N} \cap [0,n)} (u_i + u_{i+1}) = u_j + u_{j+1}. \end{aligned} \tag{5.16}$$

We now know that the point $r^{-1}(\mathbf{u})$ lies somewhere on or between c_j and c_{j+1}. We can use the ratio between u_j and u_{j+1} to determine this exact location as follows in equation 5.17:

$$r^{-1}(u_j, u_{j+1}) = \frac{c_j \cdot u_j + c_{j+1} \cdot u_{j+1}}{u_j + u_{j+1}}. \tag{5.17}$$

Note that this definition also works for cases, where the point v is equal to one of the categories' centre points c_i. We could arbitrarily use either one of its neighbours $i - 1$ or $i + 1$ for the equations and it would correctly get assigned a probability value of 0.

Next, let us consider how we could use this representation for cyclical values.

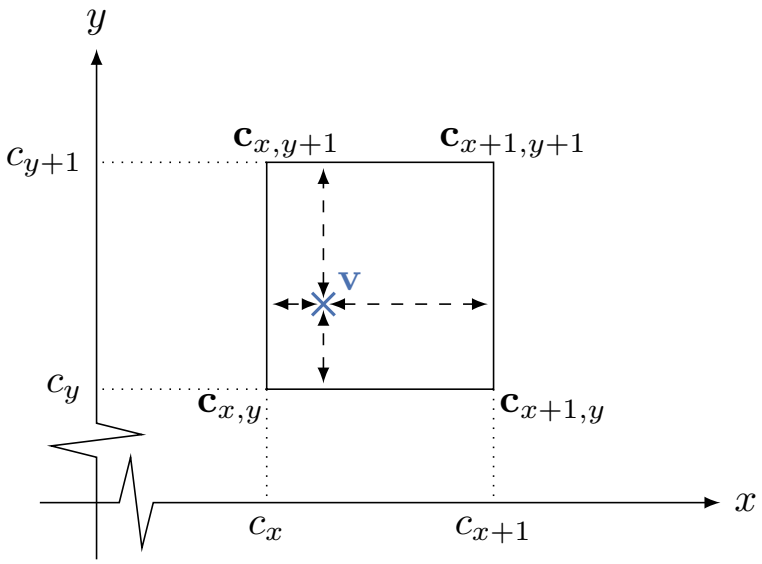


Figure 5.5: Diagram showing the layout of the classes' centre points for the 2D Continuous Classification. The arrows show the distances that we consider for the x and y components of $\mathbf{v}$.

Imagine if we wanted the two extremes of the range $[b_l, b_u]$ to be represented as the same value. We could do this by unifying the categories 0 and n. When determining the representation, we could calculate both r_0 and r_n, and predict the larger value. To invert the representation, we would assign the same predicted value to both r_0 and r_n, and use equations 5.16 and 5.17 as usual.

2D Continuous Classification

Let us now consider how we could expand this representation to two dimensions $\mathbf{v} = (v_x, v_y)$. We could distribute the categories with their centre points on a grid $\mathbf{c}_{x,y}$. Any point $\mathbf{v}$ would fall on a face formed by four points $\mathbf{c}_{x,y}, \mathbf{c}_{x+1,y}, \mathbf{c}_{x,y+1}, \mathbf{c}_{x+1,y+1}$. We could then use the ratio between the probabilities of $(r_{x,y} + r_{x,y+1})$ and $(r_{x+1,y} + r_{x+1,y+1})$ to determine the value of $\mathbf{v}_x$ and the ratio between $(r_{x,y} + r_{x+1,y})$ and $(r_{x,y+1} + r_{x+1,y+1})$ to determine the value of $\mathbf{v}_y$. This principle is visualised in Figure 5.5.

We can now define equation 5.18 to assign probability values r to the four adjacent classes of our vector $\mathbf{v}$. We use the notation c_x to refer to the x-axis element

of $\mathbf{c}_{x,y}$ and $\mathbf{c}_{x,y+1}$. Similarly, c_y is the y-axis element of $\mathbf{c}_{x,y}$ and $\mathbf{c}_{x+1,y}$.

$$\begin{aligned}
p_x(\mathbf{v}) &= \frac{c_{x+1} - v_x}{c_{x+1} - c_x}, \\
p_{x+1}(\mathbf{v}) &= \frac{v_x - c_x}{c_{x+1} - c_x} = 1 - p_x, \\
p_y(\mathbf{v}) &= \frac{c_{y+1} - v_y}{c_{y+1} - c_y}, \\
p_{y+1}(\mathbf{v}) &= \frac{v_y - c_y}{c_{y+1} - c_y} = 1 - p_y, \\
r_{x,y}(\mathbf{v}) &= p_x(\mathbf{v}) \cdot p_y(\mathbf{v}), \\
r_{x+1,y}(\mathbf{v}) &= p_{x+1}(\mathbf{v}) \cdot p_y(\mathbf{v}), \\
r_{x,y+1}(\mathbf{v}) &= p_x(\mathbf{v}) \cdot p_{y+1}(\mathbf{v}), \\
r_{x+1,y+1}(\mathbf{v}) &= p_{x+1}(\mathbf{v}) \cdot p_{y+1}(\mathbf{v}).
\end{aligned} \tag{5.18}$$

We can show that the four values of r add up to 1 as follows:

$$\begin{aligned}
& p_{x+1}(\mathbf{v}) = \frac{v_x - c_x}{c_{x+1} - c_x} \\
=& \frac{v_x - c_x}{c_{x+1} - c_x} + 1 - \frac{c_{x+1} - c_x}{c_{x+1} - c_x} \\
=& 1 + \frac{v_x - c_x - c_{x+1} + c_x}{c_{x+1} - c_x} \\
=& 1 - \frac{c_{x+1} - v_x}{c_{x+1} - c_x} \\
=& 1 - p_x.
\end{aligned} \tag{5.19}$$

The exact same logic works for p_y and p_{y+1}. Now we can show that:

$$\begin{aligned}
& r_{x,y} + r_{x+1,y} + r_{x,y+1} + r_{x+1,y+1} \\
=& p_x \cdot p_y + p_{x+1} \cdot p_y + p_x \cdot p_{y+1} + p_{x+1} \cdot p_{y+1} \\
=& \cancel{p_x \cdot p_y} + (1 - \cancel{p_x}) \cdot p_y + \cancel{p_x \cdot (1 - p_y)} + (1 - \cancel{p_x}) \cdot (1 - p_y) \\
=& p_y + (1 - p_y) = 1.
\end{aligned} \tag{5.20}$$

To invert this representation $r^{-1}(\mathbf{u})$, we would first need to find which face the representation is on. Similarly to before, we consider all the possible faces and add up the four values $u_{x,y} + u_{x+1,y} + u_{x,y+1} + u_{x+1,y+1}$ for each face. We then pick the

face that has the highest sum. Next, we can use the following equation 5.21 to invert the representation:

$$r^{-1}(u_{x,y}, u_{x+1,y}, u_{x,y+1}, u_{x+1,y+1}) = \left[\begin{array}{c} \frac{c_x \cdot (u_{x,y}+u_{x,y+1}) + c_{x+1} \cdot (u_{x+1,y}+u_{x+1,y+1})}{u_{x,y}+u_{x+1,y}+u_{x,y+1}+u_{x+1,y+1}} \\ \frac{c_y \cdot (u_{x,y}+u_{x+1,y}) + c_{y+1} \cdot (u_{x,y+1}+u_{x+1,y+1})}{u_{x,y}+u_{x+1,y}+u_{x,y+1}+u_{x+1,y+1}} \end{array} \right]. \quad (5.21)$$

Note the similarity between this equation 5.21 and its 1D counterpart 5.17. We merely consider each axis x and y separately and add the two probabilities r that should like on the same point for either axis.

Also, similarly to the 1D counterpart, we could unify the points on the extremes of the defined space to achieve a cyclic property for the represented space.

Classification Dip-Strike

In order to represent orientations and negation invariant 3D vectors, we need to represent points on the surface of a hemisphere. Also, we need to make sure that the points along the equator of the hemisphere, where it is cut in two, are cyclical and points on opposite sides are unified to the same orientation. To achieve this, we could draw lines of latitude and longitude on the hemisphere, where their intersection points are considered the centre points for our continuous classification. Equivalently, we can call lines of latitudes as lines of constant dip δ and lines of longitude as lines of constant strike ϕ from the Dip 90° Strike 360° parametrisation.

In my experiments I found that incrementing strike ϕ by 60° and dip δ by 45° yielded the best performance. This leads to 10 category centre-points and is visualised in Figure 5.6. I will use these increments to explain the representation, but it can be defined similarly for other increments of δ and ϕ. In this layout, the category centre points lie on the following coordinates in equation 5.22:

Point	A	B	C	D	E	F	G	H	I	J
Dip	90°	90°	90°	45°	45°	45°	45°	45°	45°	0°
$Strike$	0°	60°	120°	0°	60°	120°	180°	240°	300°	n/a

(5.22)

Each point X also has a point on the opposite side of the sphere X'. These points

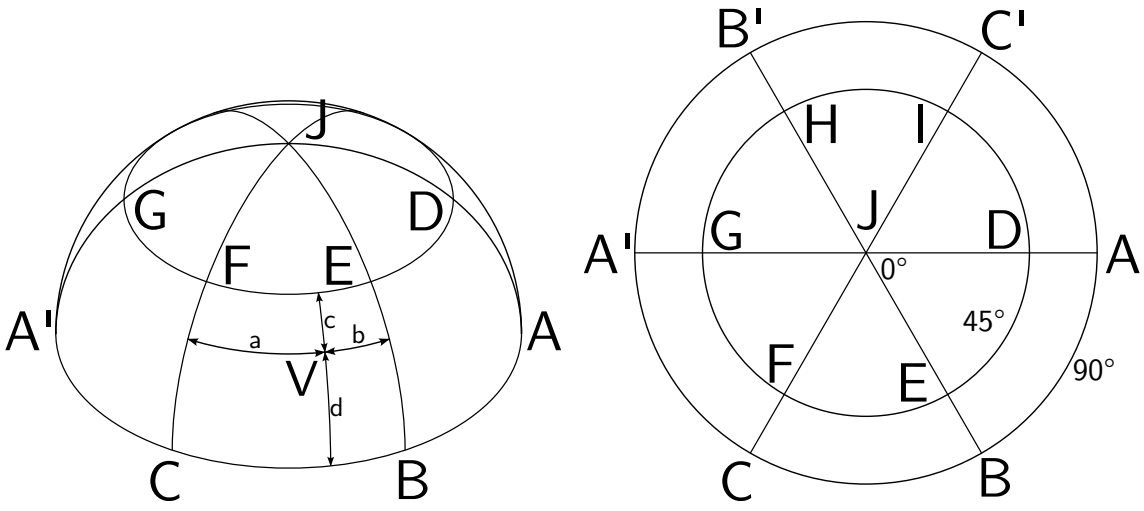


Figure 5.6: Diagram showing the points corresponding to categories in the Classification Dip-Strike representation. (b) is a top-down view of (a).

are defined as X' with $X'_\delta = 180° - X'_\delta$ and $X'_\phi = X_\phi + 180° \mod 360°$. However, the model will only predict a single probability value r_X for each pair of points X and X' to achieve negation invariance.

We can now use the 2D Continuous Classification representation, if we consider δ and ϕ as our coordinate system instead of x and y. This is certainly true for the faces with 4 vertices, such as CBFE. However, we also get faces with only 3 vertices at the pole of the sphere with $\delta = 0°$. The representation is slightly adjusted to deal with this case and is defined as follows:

We use equation 5.23 to represent points that lie on faces with four vertices. This is the case for vectors $\mathbf{v}$ with $\delta \in [45°, 90°]$ for this particular representation, or more generally with $\delta \in [\delta\_increment, 90°]$. We use the face EFBC as seen in Figure 5.6 for this definition:

$$\begin{aligned} p_\delta &= \frac{c}{c+d}\ , \\ p_\phi &= \frac{a}{a+b}\ , \\ r_E &= (1-p_\delta)\cdot(1-p_\phi), \\ r_F &= (1-p_\delta)\cdot p_\phi, \\ r_B &= p_\delta\cdot(1-p_\phi), \\ r_C &= p_\delta\cdot r_\phi. \end{aligned} \tag{5.23}$$

We use equation 5.24 for faces with three vertices, where one of the vertices is the pole at $\delta = 0°$. This is the case when representing vectors $\mathbf{v}$ with $\delta \in [0°, 45°)$ or, more generally, $\delta \in [0°, \delta\_increment)$. Note that if the represented vector has a dip of exactly $\delta\_increment$, then both equations 5.23 and 5.24 yield the same result.

$$\begin{aligned} p_\delta &= \frac{c}{c+d}\,, \\ p_\phi &= \frac{a}{a+b}\,, \\ r_J &= (1-p_\delta), \\ r_E &= p_\delta \cdot (1-p_\phi), \\ r_F &= p_\delta \cdot p_\phi. \end{aligned} \tag{5.24}$$

To achieve negation invariance, we assign the same value of r_X to both vertices on opposite sides of the sphere r_X and $r_{X'}$. We will only ask the model to predict one value r_X for each pair of vertices r_X and $r_{X'}$.

To invert the representation, we must first determine which face the represented vector lies on. This is a similar process to the 2D Continuous classification. We need to find the face with the largest sum of the predicted values of its vertices r_X. When performing this operation, we assign the predicted value of r_X to both vertices r_X and $r_{X'}$ that lie on opposite sides of the sphere. Next, depending on whether the selected face has four vertices or three vertices, we either use equation 5.25 or 5.26 respectively.

For the following four vertex equation 5.23, we use the face with vertices EFBC as seen in Figure 5.6:

$$\begin{aligned} \delta_{\mathbf{v}} &= \delta_B + (\delta_E - \delta_B) \cdot \frac{r_E + r_F}{r_E + r_F + r_B + r_C}, \\ \phi_{\mathbf{v}} &= \phi_B + (\phi_C - \phi_B) \cdot \frac{r_C + r_F}{r_E + r_F + r_B + r_C}. \end{aligned} \tag{5.25}$$

For the following three vertex equation 5.24, we use the face with vertices EFJ as

seen in Figure 5.6:

$$\begin{aligned} \delta_{\mathbf{v}} &= \delta_E + (\delta_J - \delta_E) \cdot \frac{r_J}{r_E + r_F + r_J}, \\ \phi_{\mathbf{v}} &= \phi_E + (\phi_F - \phi_E) \cdot \frac{r_F}{r_E + r_F}. \end{aligned} \tag{5.26}$$

As mentioned above, this representation can also be used with different increments of δ and ϕ. I will refer to these representations using this increment, for example, Classification Dip 45° Strike 60° for $\delta\_increment = 45°$ and $\phi\_increment = 60°$.

Classification Icosahedron

A potential downside of the aforementioned Classification Dip-Strike representation is that the points are not distributed completely uniformly. Points near $\delta = 90°$ are more widely spaced than points near $\delta = 90°$, where they merge into one, to form faces with only three vertices. Moreover, the increment of δ and ϕ can also be different. It could be beneficial to define a representation that is more uniformly distributed on the surface of a sphere. We could use the regular icosahedron, also known as a 20-faced die or D20, for this purpose. It is a platonic solid, and hence the most regular and perfectly distributed shape. Although there are other platonic solids that we could consider, the icosahedron has the largest number of vertices among them. Moreover, its faces have three vertices each, which makes it easier to define the representation.

Icosahedron Geometry We place the category centre-points on the vertices of the regular icosahedron. This icosahedron is positioned such that its vertices lie on a unit sphere. You can see the layout of the vertices and faces in Figure 5.7 and

their precise definition in the following equation 5.27:

$$\begin{aligned}
\mathbf{c}_0 &= s \cdot (0, 1, \varphi), & \mathbf{c}_6 &= s \cdot (0, -1, -\varphi), \\
\mathbf{c}_1 &= s \cdot (0, -1, \varphi), & \mathbf{c}_7 &= s \cdot (0, 1, -\varphi), \\
\mathbf{c}_2 &= s \cdot (\varphi, 0, 1), & \mathbf{c}_8 &= s \cdot (-\varphi, 0, -1), \\
\mathbf{c}_3 &= s \cdot (-\varphi, 0, 1), & \mathbf{c}_9 &= s \cdot (\varphi, 0, -1), \\
\mathbf{c}_4 &= s \cdot (1, \varphi, 0), & \mathbf{c}_{10} &= s \cdot (-1, -\varphi, 0), \\
\mathbf{c}_5 &= s \cdot (-1, \varphi, 0), & \mathbf{c}_{11} &= s \cdot (1, -\varphi, 0),
\end{aligned} \tag{5.27}$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio and $s = 1/\sqrt{\varphi^2 + 1^2} = \frac{5-\sqrt{5}}{10}$ is the factor we need to use to normalise the magnitude of the icosahedron to be on the unit sphere. Note that $\mathbf{c}_n = -\mathbf{c}_{n+6}$. These vertices lie on opposite sides of the sphere. We can pair up these points to achieve negation invariance in a similar way to the Classification Dip-Strike representation.

Next, we define the faces f_m of the icosahedron as a list of indices n of the vertices $\mathbf{c}_n$ that belong to the face. The first 10 faces are defined as 5.28:

$$\begin{aligned}
f_0 &= [1, 0, 2] & f_5 &= [1, 3, 10] \\
f_1 &= [0, 1, 3] & f_6 &= [10, 11, 1] \\
f_2 &= [0, 2, 4] & f_7 &= [1, 2, 11] \\
f_3 &= [4, 5, 0] & f_8 &= [2, 9, 11] \\
f_4 &= [0, 3, 5] & f_9 &= [2, 9, 4]
\end{aligned} \tag{5.28}$$

while the remaining 10 faces are identical, only using the opposite vertices. The relationship between faces and vertices can be seen in Fig. 5.7.

Representation Definition To define the representation on these points, we must find a similarity metric that is applied to any vertex $\mathbf{c}_i$ and the represented vector $\mathbf{v}$. This similarity must be higher the closer the two vectors are to each other. We also want the similarity between any two vertices $\mathbf{c}_i$ to be zero, and for any point lying on an edge to only have non-zero similarities between the two vertices forming

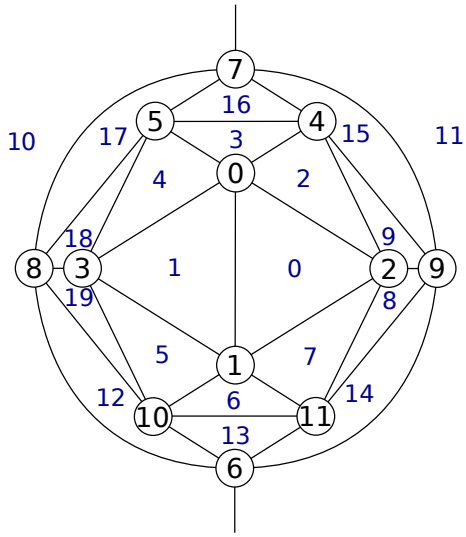


Figure 5.7: A diagram showing which vertices belong to which face of the icosahedron.

the edge. Similarly, we want a point lying on a vertex to have a similarity of 1 with the vertex and zero to all other vertices. The similarity should also be zero between any point $\mathbf{v}$ (which lies on face j) and all vertices $\mathbf{c}_i$ that do not belong to that face ($i \notin f_j$). Lastly, the sum of all the similarities between a point $\mathbf{v}$ and all the vertices $\mathbf{c}_i$ should be equal to one. Together, all of these properties should achieve a representation that behaves similarly to the Classification Dip-Strike representation or the 2D Continuous Classification.

A good starting point for the representation is to determine what face the vector $\mathbf{v}$ belongs to. We will then only allow the similarity for vertices on that face to be non-zero. This allows us to reduce the problem from points on the surface of a sphere to points on a triangle. If we find transformation matrices that align every face of the icosahedron to a standardised triangle, then we can reduce the problem to two dimensions, making it simpler.

Consider the triangle $\mathbf{abc}$, where $\mathbf{a} = (0, 0, 0), \mathbf{b} = (1, 0, 0), \mathbf{c} = (1/2, \sqrt{3}/2, 0)$. We can find transformation matrices T_j from any face j onto the triangle $\mathbf{abc}$. If we determine that a point $\mathbf{v}$ lies on face j, then applying T_j to $\mathbf{v}$ will give us $\mathbf{u}$, which exists in the same standardised space. We can then apply the same representation equations regardless of which face j belongs to. We can then calculate the similarity between $\mathbf{u}$ and points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, which will correspond to the similarity r_i for

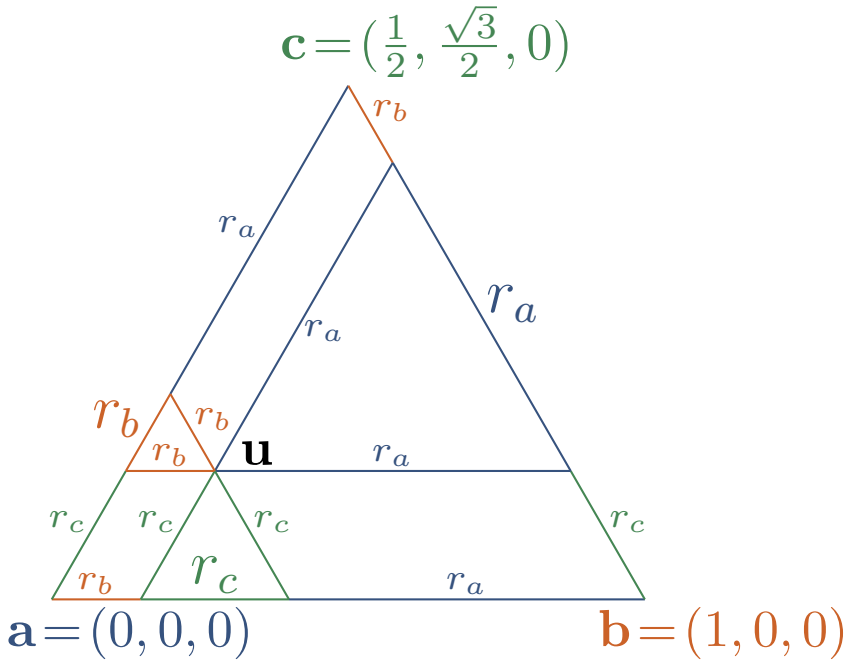


Figure 5.8: A diagram showing how we calculate similarity on a standardised triangle for any icosahedron face.

vertices $i \in f_j$.

To calculate the similarity we could use Barycentric coordinates, if they are normalised to add up to 1. Barycentric coordinates directly give the proportion of the area of the sub-triangles opposite to each vertex relative to the area of the whole triangle. By making the area of the triangle 1, we could use these values as our similarity measure. Wachspress Coordinates and Mean Value Coordinates are generalisations of Barycentric coordinates to more complex polygons, with identical values when applied to an equilateral triangle. An equivalent definition of all of these coordinates can be expressed using the geometry shown in Figure 5.8. We can use the side length r_a, r_b and r_c to form our representation, assuming that $\mathbf{v}$ lies on the triangle **abc**. However, even though the vertices on the icosahedron lie on the unit sphere, with a magnitude of 1, other points on the edges or faces of the icosahedron do not have a magnitude of 1. Therefore, after using transformation matrices T_j, the point $\mathbf{u}$ will not lie on the triangle **abc** and it will have a z-axis component that is not equal to zero. To address this, we will project the point $\mathbf{u}$ onto the surface of the triangle **abc** by setting its z-axis component to zero.

We can now formalise this representation using the following equations. Let us first define the representation r_a, r_b, r_c on the triangle **abc** in terms of the vector **u** in equation 5.30. From the geometry of the triangle in Figure 5.8 we can deduce the following equation 5.29:

$$\begin{aligned} u_x &= r_b + \frac{r_c}{2}, \\ u_y &= r_c \cdot \frac{\sqrt{3}}{2}, \\ 1 &= r_a + r_b + r_c. \end{aligned} \tag{5.29}$$

From these equations 5.29 we can deduce the following 5.30:

$$\begin{aligned} r_a(\mathbf{u}) &= 1 - \frac{u_x}{2} - \frac{u_y}{2\sqrt{3}}, \\ r_b(\mathbf{u}) &= \frac{u_x}{2} - \frac{u_y}{2\sqrt{3}}, \\ r_c(\mathbf{u}) &= \frac{u_y}{\sqrt{3}} \end{aligned} \tag{5.30}$$

Let us now follow the process of defining the representation from the start. First, we must find out what face **v** belongs to. We can do this by finding the face that minimises the angle, or the dot product between the vertices $\mathbf{c}_i$ and the vector **v** as follows in equation 5.31, which relies on equations 5.28, 5.27:

$$\text{face}(\mathbf{v}) = j \text{ s.t. } \min_i(\sum_{k \in f_i}(\mathbf{c}_k \cdot \mathbf{v})) = \sum_{k \in f_j}(\mathbf{c}_k \cdot \mathbf{v}) \tag{5.31}$$

Now that we know what face **v** belongs to, we need to find the transformation that maps **v** onto **u** on the standardised triangle **abc**. We do this by using transformation matrices T_j, T_j^{-1} that map the vertices $\mathbf{c}_i$ for a face $i \in f_j$ onto the vertices **a**, **b** and **c** respectively. The process to find these transformation matrices is defined in equation 5.47 at the end. Note that we use the convention from computer graphics and perform the multiplication as $\mathbf{v} \cdot T$ (not $T \cdot \mathbf{v}$). In other words, the vectors are row vectors. We will then def transformation as follows in equation 5.32 using

equations 5.31 and 5.47:

$$\mathbf{u}(\mathbf{v}) = \begin{bmatrix} [\mathbf{v} \times T_{\text{face}(\mathbf{v})}]_x \\ [\mathbf{v} \times T_{\text{face}(\mathbf{v})}]_y \\ 0 \end{bmatrix}. \tag{5.32}$$

The full representation is then defined as follows in equation 5.33, using equations 5.28, 5.30, 5.31, 5.32:

$$\begin{aligned} s_i(\mathbf{v}) &= \begin{cases} r_a(\mathbf{u}(\mathbf{v})) & \text{if } i = [f_{\text{face}(\mathbf{v})}]_0, \\ r_b(\mathbf{u}(\mathbf{v})) & \text{if } i = [f_{\text{face}(\mathbf{v})}]_1, \\ r_c(\mathbf{u}(\mathbf{v})) & \text{if } i = [f_{\text{face}(\mathbf{v})}]_2, \\ 0 & \text{otherwise}, \end{cases} \\ r_i(\mathbf{v}) &= \max(s_i(\mathbf{v}), s_{i+6}(\mathbf{v})) \end{aligned} \tag{5.33}$$

Note that the final representation will only predict the values r_i for the first 6 vertices, and is assigned as the larger of the pair of values s_i and s_{i+6}.

Inverse Representation Definition Next, let us invert the representation. First, we must determine which face the predicted $\mathbf{r}$ lies on. We can do this by finding the face with the largest sum of the representation value s_i for its respective vertices $i \in f_j$ after assigning the predicted values of r_i to both s_i and s_{i+6}. We get the following equation 5.34:

$$\begin{aligned} s_i(\mathbf{r}) &= r_{\{i \bmod 6\}}, \\ \text{face\_inv}(\mathbf{r}) &= j \text{ s.t. } \min_i(\sum_{k \in f_i} s_k(\mathbf{r})) = \sum_{k \in f_j} s_k(\mathbf{r}). \end{aligned} \tag{5.34}$$

We can now identify r_a, r_b, r_c on the triangle **abc** as $r_{f_{\text{face\_inv}(\mathbf{r})}}$ respectively. We can use these values to find the location of $\mathbf{u}$ in equation 5.35, which is based on equations 5.29. Note that we have to normalise the magnitude of r_a, r_b, r_c, since it

is not guaranteed that they add up to 1, due to errors in the predictions.

$$\mathbf{u}(\mathbf{r}) = \begin{bmatrix} 1 - \frac{r_a}{r_a+r_b+r_c} + \frac{r_b}{r_a+r_b+r_c} \\ \frac{\sqrt{3}\cdot r_c}{r_a+r_b+r_c} \\ 0 \end{bmatrix}. \tag{5.35}$$

Next, we need to find $\mathbf{v}$ from $\mathbf{u}$, by applying inverse transformation matrices T_j^{-1}, that map the triangle **abc** onto the vertices $\mathbf{c}_i$ on the face j, where $i \in f_j$. This will map the point $\mathbf{u}$ onto the surface of the icosahedron. To map this point onto the unit sphere and find $\mathbf{v}$, we must normalise its magnitude to be equal to 1. This is defined in the following equation 5.36, which uses equations 5.34, 5.35, 5.48:

$$\mathbf{v}(\mathbf{r}) = \mathbf{u}(\mathbf{r}) \times T^{-1}_{\text{face\_inv}(\mathbf{r})}. \tag{5.36}$$

Transformation Matrices Let us now define the transformation matrices T_j that transform the vertices $\mathbf{c}_{f_j}$ onto $\mathbf{a}, \mathbf{b}, \mathbf{c}$ on the standardised triangle seen in Figure 5.8 respectively. To facilitate these transformations we use homogeneous coordinates, which allow us to represent translations and other affine transformations as matrix multiplications. Homogeneous coordinates extend Cartesian coordinates by adding one additional dimension as follows: $\mathbf{v} = (x, y, z)$ is expressed as $(x, y, z, 1)$ in homogeneous coordinates. The transformation matrices T_j are 4x4 matrices, which are multiplied with the vector for the transformation. I will use the convention from computer graphics, where vectors $\mathbf{v}$ are expressed as row vectors and are multiplying them by the transformation matrices T_j from the right as $\mathbf{v} \cdot T_j$. From now on, when performing operations between vectors and transformation matrices, the use of homogeneous coordinates is implied. Formally, we want the transformations to

satisfy the following equations 5.37, which use equations 5.27, 5.28:

$$\begin{aligned} \forall j \in [0,9] \cap \mathbb{Z} \text{ find } T_j \text{ s.t. :} \\ \mathbf{a} = (0,0,0) = \mathbf{c}_{[f_j]_0} \times T_j, \\ \mathbf{b} = (1,0,0) = \mathbf{c}_{[f_j]_1} \times T_j, \\ \mathbf{c} = (\frac{1}{2}, \frac{\sqrt{3}}{2}, 0) = \mathbf{c}_{[f_j]_2} \times T_j. \end{aligned} \tag{5.37}$$

Finding these transformation matrices algebraically is difficult, however, we can construct them from a sequence of consecutive transformations. Let us define the transformations to perform translation and rotations in the x, y, and z axes as follows in equation 5.38:

$$\begin{aligned} T_{\text{translation}}(x,y,z) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & z & 1 \end{bmatrix} \\ T_{\text{rotation\_}x}(\theta) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ T_{\text{rotation\_}y}(\theta) &= \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ T_{\text{rotation\_}z}(\theta) &= \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{5.38}$$

We can now construct our transformation matrices T_j using these operations.

First, we perform a translation to move the first point $\mathbf{c}_{[f_j]_0}$ onto $\mathbf{a} = (0, 0, 0)$:

$$T_{\mathrm{I}}(j) = T_{\mathrm{translation}}(-[\mathbf{c}_{[f_j]_0}]_0, -[\mathbf{c}_{[f_j]_0}]_1, -[\mathbf{c}_{[f_j]_0}]_2). \tag{5.39}$$

Let us also define the inverse of this operation for later use:

$$T_{\mathrm{I}}^{-1}(j) = T_{\mathrm{translation}}([\mathbf{c}_{[f_j]_0}]_0, [\mathbf{c}_{[f_j]_0}]_1, [\mathbf{c}_{[f_j]_0}]_2). \tag{5.40}$$

Next, consider how the vertices $\mathbf{c}_{f_j}$ are transformed by $T_{\mathrm{I}}(j)$. $\mathbf{c}_{[f_j]_0} \times T_{\mathrm{I}}(j)$ is guaranteed to be $(0, 0, 0)$, while $\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j)$ and $\mathbf{c}_{[f_j]_2} \times T_{\mathrm{I}}(j)$ vary depending on f. Next, let us apply a rotation in the y-axis to align the second point (index 1) so that its z-axis component is zero.

$$T_{\mathrm{II}}(j) = T_{\mathrm{rotation\_}y}(\mathrm{atan2}([\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j)]_1, [\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j)]_0)), \tag{5.41}$$

$$T_{\mathrm{II}}^{-1}(j) = T_{\mathrm{rotation\_}y}(-\mathrm{atan2}([\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j)]_1, [\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j)]_0)). \tag{5.42}$$

Next, let us make the y-axis component of the second vertex (index 1) equal to zero, by performing a rotation in the z-axis. By performing the rotation in the z-axis, we know that the z-axis component of the vector will be unaffected, leaving at a value of zero.

$$T_{\mathrm{III}}(j) = T_{\mathrm{rotation\_}z}(\mathrm{atan2}([\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j)]_2, [\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j)]_0)), \tag{5.43}$$

$$T_{\mathrm{III}}^{-1}(j) = T_{\mathrm{rotation\_}z}(-\mathrm{atan2}([\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j)]_2, [\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j)]_0)). \tag{5.44}$$

The second vertex $\mathbf{c}_{[f_j]_1} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)$ will now be equal to $\mathbf{b} = (1, 0, 0)$. We know this because we make its y-axis component zero using $T_{\mathrm{II}}(j)$ and its z-axis component zero using $T_{\mathrm{III}}(j)$. We know that it will be a distance of 1 away from the first point $\mathbf{c}_{[f_j]_0} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)$, which was made into $\mathbf{a} = (0, 0, 0)$ using $T_{\mathrm{I}}(j)$ and it remained unchanged the rotations around the origin $T_{\mathrm{II}}(j)$ and $T_{\mathrm{II}}(j)$. We are now only left with the third vertex as a function of f. A rotation around

the x-axis will leave both the first and second vector unaffected, since they have zero values in the y-axis and z-axis components. Let us therefore use the rotation around the x-axis to make the z-axis component of the third vertex equal to zero:

$$\begin{aligned} T_{\mathrm{IV}}(j) = T_{\mathrm{rotation\_}x}(\mathrm{atan2}([\mathbf{c}_{[f_j]_2} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)]_2, \\ [\mathbf{c}_{[f_j]_2} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)]_1)), \end{aligned} \tag{5.45}$$

$$\begin{aligned} T_{\mathrm{IV}}^{-1}(j) = T_{\mathrm{rotation\_}x}(-\mathrm{atan2}([\mathbf{c}_{[f_j]_2} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)]_2, \\ [\mathbf{c}_{[f_j]_2} \times T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j)]_1)), \end{aligned} \tag{5.46}$$

With this fourth transformation, our vertices now lie on **a**, **b**, **c** respectively. We can therefore define the full transformation matrices as follows in equation 5.47:

$$T_j = T_{\mathrm{I}}(j) \times T_{\mathrm{II}}(j) \times T_{\mathrm{III}}(j) \times T_{\mathrm{IV}}(j). \tag{5.47}$$

Similarly, the inverse to the whole operation is defined as follows in equation 5.48:

$$T_j^{-1} = T_{\mathrm{IV}}^{-1}(j) \times T_{\mathrm{III}}^{-1}(j) \times T_{\mathrm{II}}^{-1}(j) \times T_{\mathrm{I}}^{-1}(j). \tag{5.48}$$

The correctness of these transformation matrices can easily be verified numerically, by ensuring that the equation 5.37 is satisfied for all faces j. I have indeed performed this check.

5.1.5 Saxena's Representation

Lastly, let us talk about an existing representation proposed by Saxena et. al. [109] as a benchmark to compare my representations against. Saxena's representation uses a 3×3 matrix to encode a vector $\mathbf{v}$ as $\mathbf{v}\cdot\mathbf{v}^T$. This is a symmetrical matrix and therefore contains 6 unique values, which we can predict using the neural network. To invert the representation, we take the eigenvector corresponding to the largest eigenvalue in the matrix. This can be done using principal component analysis (PCA) [110], however, that is an expensive operation and cannot be easily performed on a GPU.

Although this representation is promising in principle, its computation was too expensive in practice. Any models that predict orientation for every pixel/voxel in a

volume must invert the representation for every pixel/voxel. This means the use of PCA for every pixel/voxel when using Saxena's representation, which was too slow. It was only viable to use the representation on the 3D Toy Dataset.

5.2 Experiments

5.2.1 3D Dummy Dataset

Moving on from 2D to 3D, I initially performed experiments on the Dummy 3D dataset. I only spent a brief period on this dataset before it got superseded by the combination of the Toy Dataset and the Synthetic Seismic Dataset. The Toy dataset allows us to evaluate the representations in a very controlled environment that allows us to identify the specific orientations that the representations struggle with. Conversely, the synthetic seismic dataset allows to better evaluate the practical utility of the representations in a more realistic and challenging environment. This 3D Dummy Datasets sits between the two, with pixel-wise labels that require a large model, but without resembling the seismic data. As such, it provides little utility over the combination of the two datasets that superseded it and their associated experiments.

All of the experiments that I performed on this dataset were during the prototyping and debugging stage of the creation of this dataset. I cannot be certain whether there weren't bugs in how I created the labels for this dataset, in fact, I suspect that was the case. I abandoned the dataset before addressing all of my doubts about it. Although the dataset was useful to figure out the direction of my research before I had access to the Synthetic Seismic Data, I did not draw any conclusions from the experiments about the performance of the various representation. For this reason I decided to omit these results from this thesis.

5.2.2 3D Toy Dataset

The 3D Toy Dataset gives us the unique opportunity to evaluate the representations and identify any specific orientations that it struggles with. This allows us to confirm

the presence discontinuities in the representations and prove that they cause issues for neural networks. On the contrary, we can demonstrate that the four proposed representations do not suffer from these issues.

The volumes in this dataset are only $8 \times 8 \times 8$ voxels in size, with a single orientation label for each volume. It is therefore sufficient to train a very small model on this dataset. Since the dataset contains a volume for every possible orientation, we can evaluate all the possible orientations very effectively.

Since the dataset is very simple, there is no variety in the volumes apart from the varying orientation. However, we want to both train and evaluate the model on all the possible orientations. This is why we cannot split the dataset into a training and validation set. However, this does not matter, since issues with the representations should even be present in the training performance. The model should struggle with problematic representations, especially near their discontinuities. With a sufficiently small model, regardless of whether overfitting occurs or not, we should see differences in the training performance of the representations are fundamentally flawed, which we indeed observe.

Network Architecture

The model used for these experiments is shown in Figure 5.9. It is a convolutional architecture that downsamples the resolution of the volume, with a final fully connected layer. With a total of 1329 parameters, this can be considered a very small model.

Results

To visualise the error for all possible orientations, we need to create a visualisation of the surface of a sphere, or at least a hemisphere, corresponding to the possible orientations. These errors in the predictions can then be visualised on the sphere as colour coding. Rather than visualising the sphere in 3D, it is more convenient to use a projection of the sphere onto a 2D image. To do this, I use an equal area projection. These results are visualised in Figure 5.10. Depending on the random initialisation of the model and all other random elements during training,

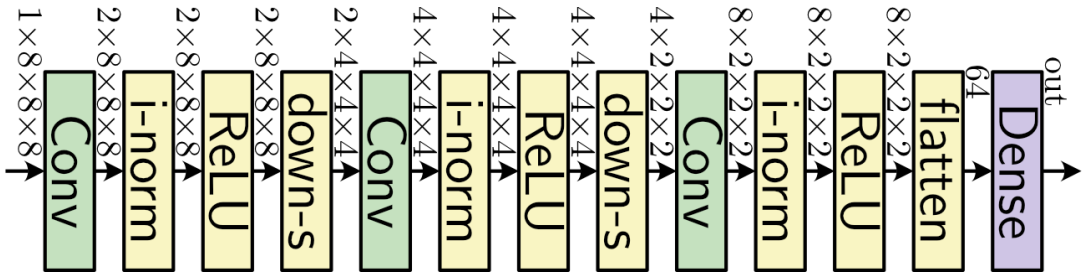


Figure 5.9: The 3D convolutional neural network architecture used on the 3D Toy Dataset and task. Conv is a 3D convolutional layer with a $3 \times 3 \times 3$ kernel, padding of 1 and a strike of 1. i-norm is instance normalisation, down-s is $2\times$ downsampling using max pooling, Dense is a fully connected layer. The dimensions shown are of the activations propagating through the model, with the first being the channel dimension.

the performance of a model varies between different runs. To minimise the variation caused by randomness, we can run the model multiple times and average the results across all said runs. This is already the case for the images in Figure 5.10.

We can also use the average error over all possible orientations as a useful metric. In Figure 5.11 we can see how this average error changes over the course of training the model. The plotted line is the average across multiple runs. In Figure 5.11 we can see the distribution across the different runs, of the average error across all orientations.

Discussion

Let us first talk about the issues with the benchmark representations and how the experimental results verify the expected issues from discontinuities that I identified in theory.

First, let us look at the Dip-Strike representations, which were described in Figure 5.2 and Equations 5.2 and 5.3.

The Dip 90° Strike 360° representation has discontinuities at $\delta = 90°$, where the strike angle suddenly changes by 180°. We observe large errors in the predictions near this discontinuity, seen as a ring in Figure 5.10. The performance of this representation is good outside of the discontinuity, however, we can see that the overall performance was negatively affected in Figure 5.12.

The Dip 180° Strike 180° representation suffers from a discontinuity at $\phi =$

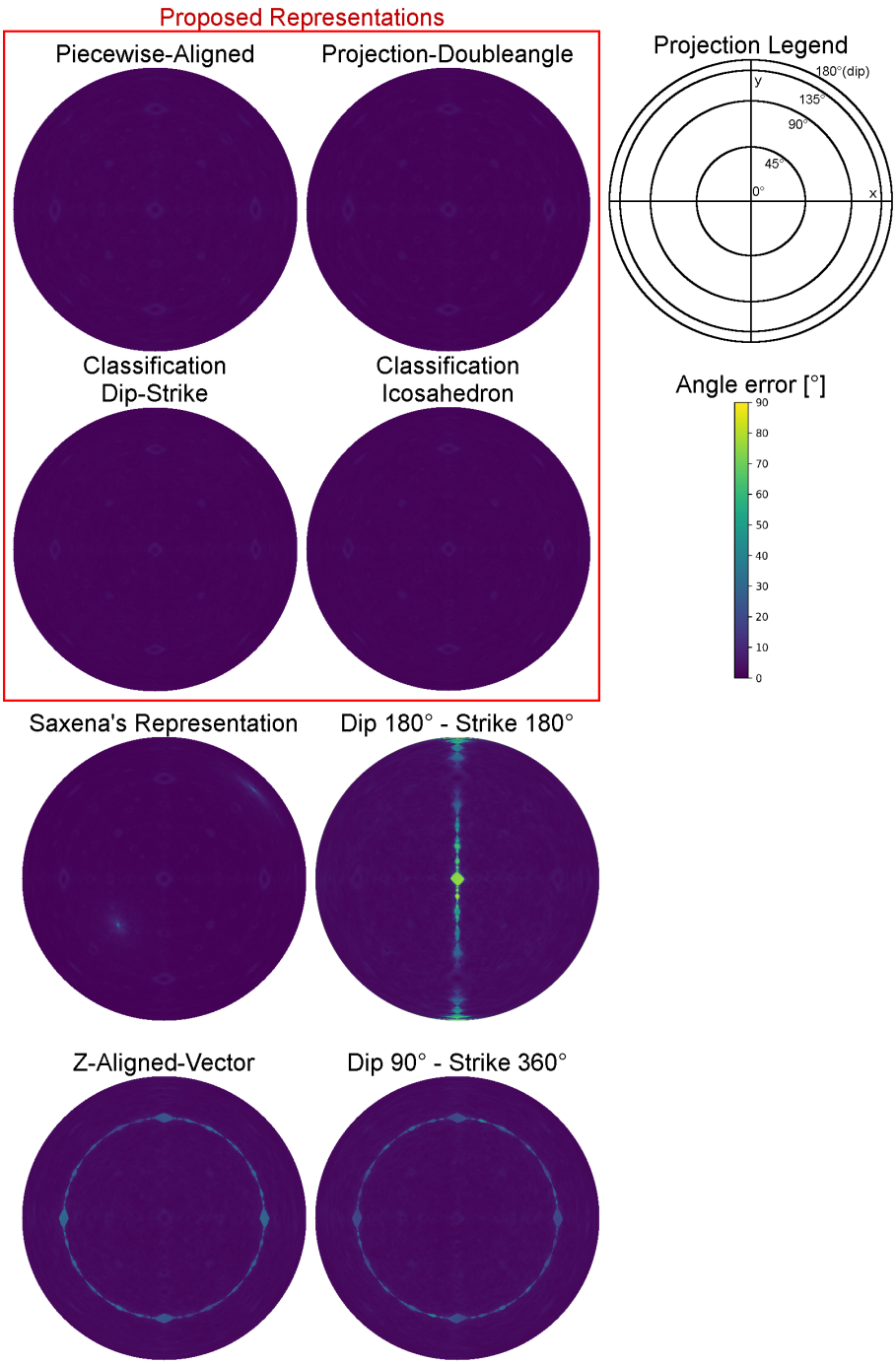


Figure 5.10: The results of the 3D Toy dataset and task. The error is visualised for every possible orientation as colour coding on an equal area projection of a sphere. Note that there is a symmetry in the images, since any direction and its negative are the same orientation. The errors shown are averaged over 10 runs of the same model.

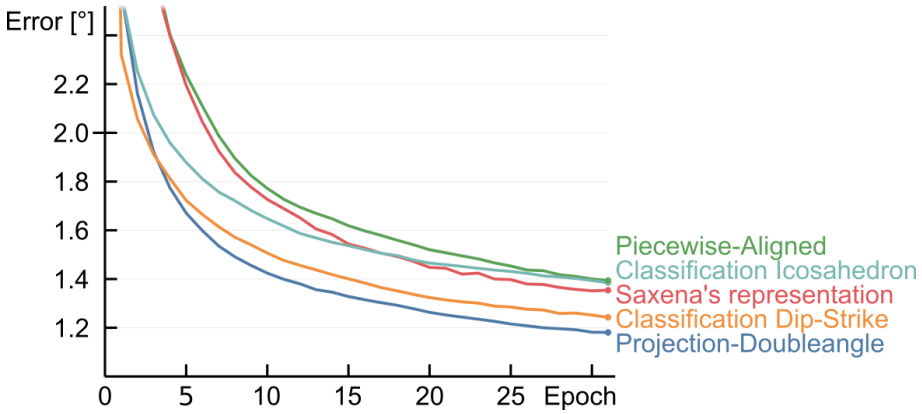


Figure 5.11: The performance during training on the 3D Toy task. The errors shown are averaged over 10 runs of the same model.

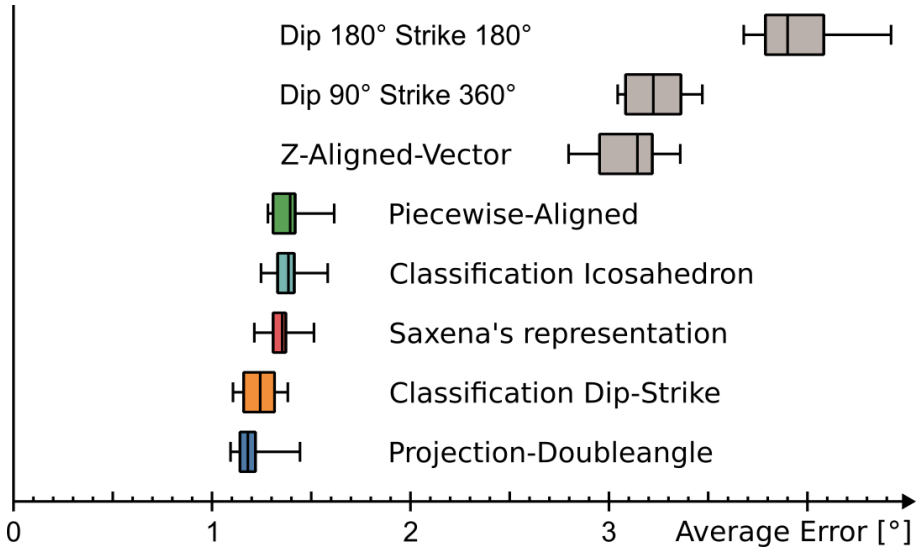


Figure 5.12: A distribution of the final performances on the 3D Toy task over the 10 runs of the same model.

$0° = 180°$. Despite using the doubleangle representation on the strike angle, which removes the discontinuity in the strike angle, the dip angle becomes discontinuous. The dip value jumps from δ to $180° - dip$ in places where $\phi = 0° = 180°$. This jump is big for dip values close to $0°$ or $180°$, but small near values of $90°$. This is precisely what we observe in Figure 5.10, where large errors occur near places where the discontinuity is big. On average over all orientations, this representation performs the worst, as seen in Figure 5.12.

The Z-Aligned-Vector representation suffers from a discontinuity at $z = 0$, which corresponds to $\delta = 90°$, which is where we observe large errors in Figure 5.10. These locations align with the Dip 90° Strike 360° representation, but on average the Z-Aligned-Vector representation performs a little better, as seen in Figure 5.12.

Next, let us look at Saxena's representation. On average it performs well, being the third best representation, as seen in Figure 5.12. However, it has errors near a specific orientation, as seen in Figure 5.10. It also trains more slowly than the proposed representations, as seen in Figure 5.11. However, its biggest problem is the computational time required to compute PCA to invert the representation. On my system, the overall training time on Saxena's representation was almost 4 times as long as the other representations.

Moving on to the proposed representations, in Figure 5.10 we can see that all 4 representations perform uniformly well across all orientations. They do not have any specific orientations for which their error is higher. That said, they do vary slightly in their overall average performance. The Piecewise representation performs worst, both training slower and plateauing at a worse performance. The classification representations train quickly, but the Classification Icosahedron plateaus earlier than the Classification Dip-Strike representation. Finally, the Projection-Doubleangle representation performs the best.

We cannot be sure whether the performance of these representations translates to other tasks. We observe differences in how quickly the representations train, which could be exacerbated on difficult tasks, and could possibly even cause the models to get stuck and plateau at different performances. What we can be certain of from this experiment, however, is that discontinuities are indeed a problem for

neural network output representations. We can also be certain that the proposed representations don't suffer from orientation-specific issues and perform uniformly well across all orientations.

5.2.3 3D Synthetic Seismic Datasets

The 3D Synthetic Seismic Dataset is the closest counterpart to real 3D seismic surveys while having ground truth labels. In fact, the synthetic data is similar enough to the real seismic data that the models trained on the synthetic data can be applied directly to the real seismic data without the need for any fine-tuning or domain adaptation techniques. I will use this principle in Chapter 6.

Unlike hand labelled real data, synthetic seismic data has the advantage that its labels are guaranteed to be correct and perfectly accurate. This is important for training deep learning models, especially if we strive to surpass human performance, since we can at best match human performance if we train on human annotations. Additionally, unlike what geologists tend to label in their interpretations, we can extract more information from the synthetic data. Specifically, we can get information about the orientation of the faults. We can therefore use this data to evaluate the effectiveness of the orientation representations on a task and dataset that is very similar to its intended deployment.

Since we are predicting the orientation of faults in the dataset, the distribution of orientations that we are evaluating determined by the orientations present in the dataset. These do not cover all the possible orientations and are restricted to dip values in the range of 10° to 35°, followed by random transformations that may rotate them further. This is why it was important to also use the 3D Toy dataset. It can be reasoned that any representation that performs well for all orientations on the 3D Toy Dataset and performs well in practise on the 3D Synthetic Seismic Dataset should also perform well in any other task.

The dataset consists of 1535 volumes of size $96 \times 96 \times 96$ voxels and is split into training, validation and testing sets in a 80%, 10% and 10% ratio respectively. I used the validation set for all of the model optimisations, while reporting the test set performances for the metrics seen when comparing the representations.

Network

Similarly to the networks used in for the 2D tasks on chromosomes, this tasks takes images as inputs and returns images of the same resolution. The only difference being that the images have an extra spatial dimension. The way to address the extra spacial dimension is to use 3D convolutions as opposed to 2D convolutions. Otherwise, the same principles of network design still hold.

The biggest difference that using 3D volumes as opposed to 2D images is the memory footprint of the data. Not only on the inputs and outputs, but also for the hidden activations in the model. The VRAM of a GPU poses a major limit to the size of the model that can be used and therefore inform the model design.

The UNet architecture [106] provides a memory efficient way of predicting pixel/voxel-wise features. Every downsample in the architecture reduces the memory footprint of the subsequent layers at the lowered resolution. In two dimensions, downsampling by $2\times$ roughly reduces the memory to a quarter, while in 3D to an eighth. However, detail is lost at these lower resolutions. It is therefore important to find a balance between the amount of computation performed at higher and at lower resolutions.

To find the best architecture, I restricted my search to models that use most of the 24GB of available VRAM memory on my Titan RTX graphics card. I always adjusted the numbers of channels/filters a.k.a. the width of the network accordingly, to use up the available memory.

Within the UNet architecture, convolutional blocks are used to perform calculations at every resolution. However, these convolutional blocks can themselves use various architectures. These can have different convolutional parameters, use different normalisation layers, or have different skip connections. In my experiments, I varied the parameters of these blocks, while also changing their surrounding UNet structure. The UNet can have different numbers of downsamples, while also having different widths or depths of the blocks within it. Because it takes a lot of time to train a single model on this dataset, the optimisation process had to be efficient. Rather than running an automated sweep over all these hyperparameters, I manually selected the next models to run.

The metrics that I paid most attention during the optimisation procedure are

capturing the performance of predicting the location of the faults and how well their orientation is represented. All of these models were optimised using the Piecewise-Vector Representation, before I had come up with the other 3D representations. The two metrics that capture these properties best are the Intersection over Union (IOU) score of the fault location and the average angle error of the orientation. Empirically, these two metrics were highly correlated in my experiments, which implies that architectural improvements apply to the general model's ability to interpret the images, rather than optimising for one part of the task or the other (fault location vs orientation prediction). Because of this, I would expect the same architecture to perform well for all the different orientation representations. I have therefore not repeated the network optimisation procedure for the different orientation representations.

Figure 5.13 shows all the various architecture types that I experimented with and optimised. Figure 5.14 shows the final optimised architecture. The details and performances of all the models that I trained in the process of optimising the architecture can be found in the Appendix Tables A.1, A.2, and A.3. My general findings from the optimisation process are summarised as follows:

1. Using concatenation instead of addition as the Combine operation yields better performance.

2. Using Instance Normalisation [2] as the Norm Layer yields similar performance to Batch Normalisation [1], while avoiding a floating point overflow on the deeper architectures when using mixed (16 and 32 bit) floating point precision.

3. We order the blocks in increasing order of performance as follows: "Conv Block", "HarDAddNet Block", "ResNet Block", "HarDNet Block"

The optimised architecture in Figure 5.14 has the following architectural details: UNet with 2 down/up samples, using max pooling with a kernel and stride of $2 \times 2 \times 2$ for down-sampling, and trilinear interpolation for up-sampling. The model uses instance normalisation [2] for the Norm Layer, concatenation for the Combine operation, a batch size of 4, the Adam optimiser [111] with a learning rate of 1.0e-3. The model uses HarDNet Blocks for the encoders, decoders and the backbone, all of

which use a value of $m = 1.7$. The width of the hidden layers is: $h1 = 72$, $h2 = 104$; the encoder block 1 and decoder block 2 have a depth of 8 and $k = 11$; the encoder block 2 and decoder block 1 have a depth of 8 and $k = 16$; the backbone has a depth of 64 and $k = 24$.

Results

Figures 5.15 and 5.16 are counterparts to Figures 5.11 and 5.12 respectively. Figure 5.15 plots the average performance of the model across 4 separate training runs, while 5.16 shows the distribution of the best performances achieved in each run. These Figures and results have been published in [5].

In addition to predicting the orientation, the models also predict the location of the faults. This performance should be similar for all models. Table 5.1 shows all the metrics for the different representations, while their averages are shown in Table 5.2.

Discussion

Unlike the 3D Toy dataset, we cannot easily visualise the error for different orientations on the Synthetic Seismic Dataset. The orientations are not uniformly distributed within the dataset, rather, they are made to resemble the distribution of orientation within real seismic surveys. In this way, the average performance on the synthetic data should more closely resemble the performance we might expect on the real data.

We observe a similar pattern of performances between the Toy Dataset and the Synthetic Seismic Dataset across the 4 suggested orientation representations, when looking at the final distribution of their performances in Figure 5.16 for the synthetic seismic data compared to the toy task in Figure 5.16. The Projection-Doubleangle representation clearly performs best, followed by the two classification representations, and lastly the Piecewise-Aligned representation. We do, however, observe some differences in the performances. Most notably, the average error is much higher for all the representations on the synthetic seismic data than on the toy task. This is not surprising, since the task is much more difficult in general. It

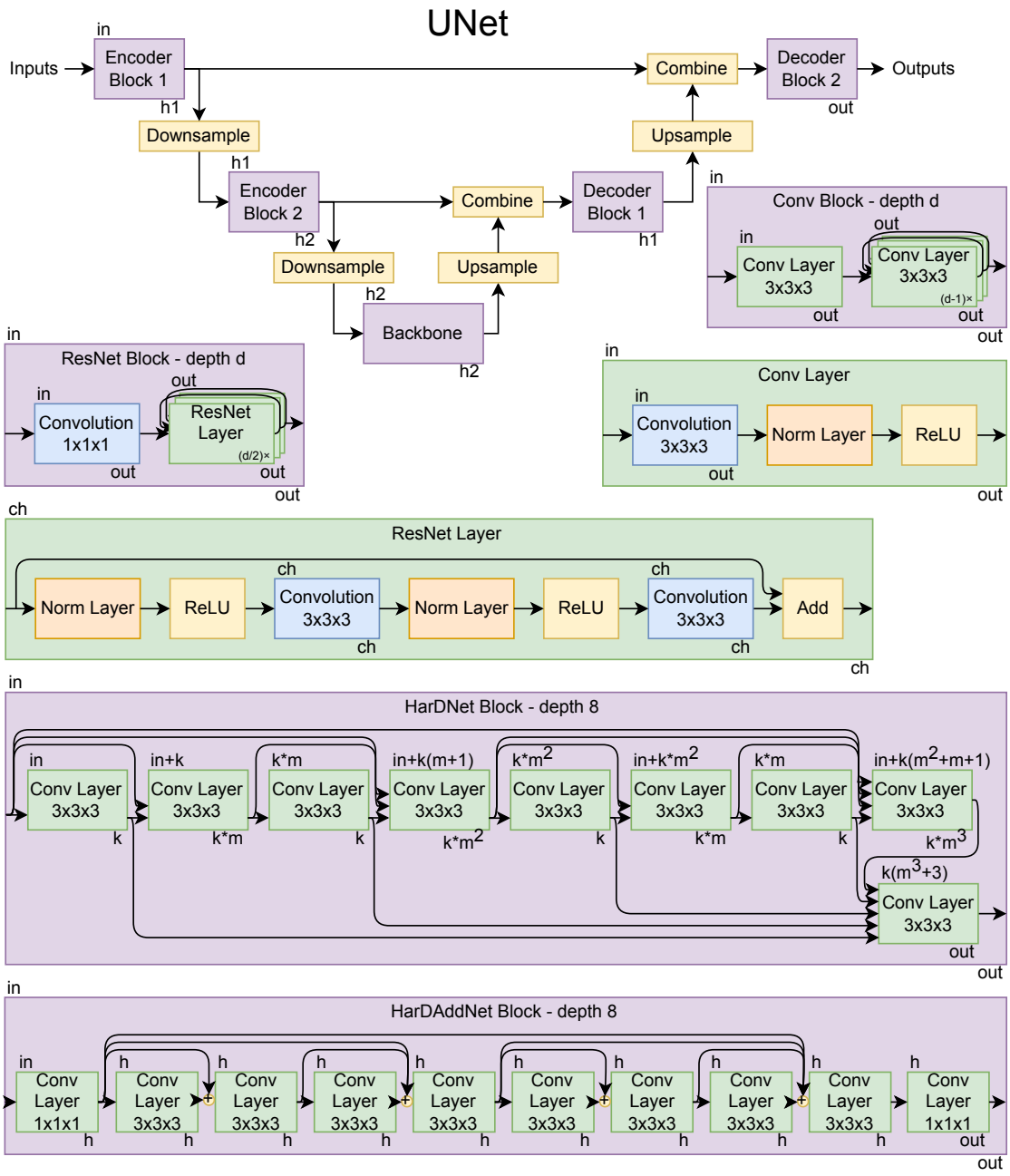


Figure 5.13: The model architectures that I used in my experiments on 3D seismic data. The UNet is the template that contains placeholders for encoder/decoder blocks, downsample/upsample operations and combine blocks, as well as the norm layers used within the various blocks. These placeholders are then populated. The options for the blocks are the Conv Block, ResNet Block, HarDNet Block and HarDAddNet Block. The UNet can also vary with the number blocks and up/down sample operations, while each block can be made with different widths/depths (or other hyperparameters), that also have to be specified. The numbers above a box in the top left represent the number of input channels passed into the block, while the number of output channels is on the bottom right.

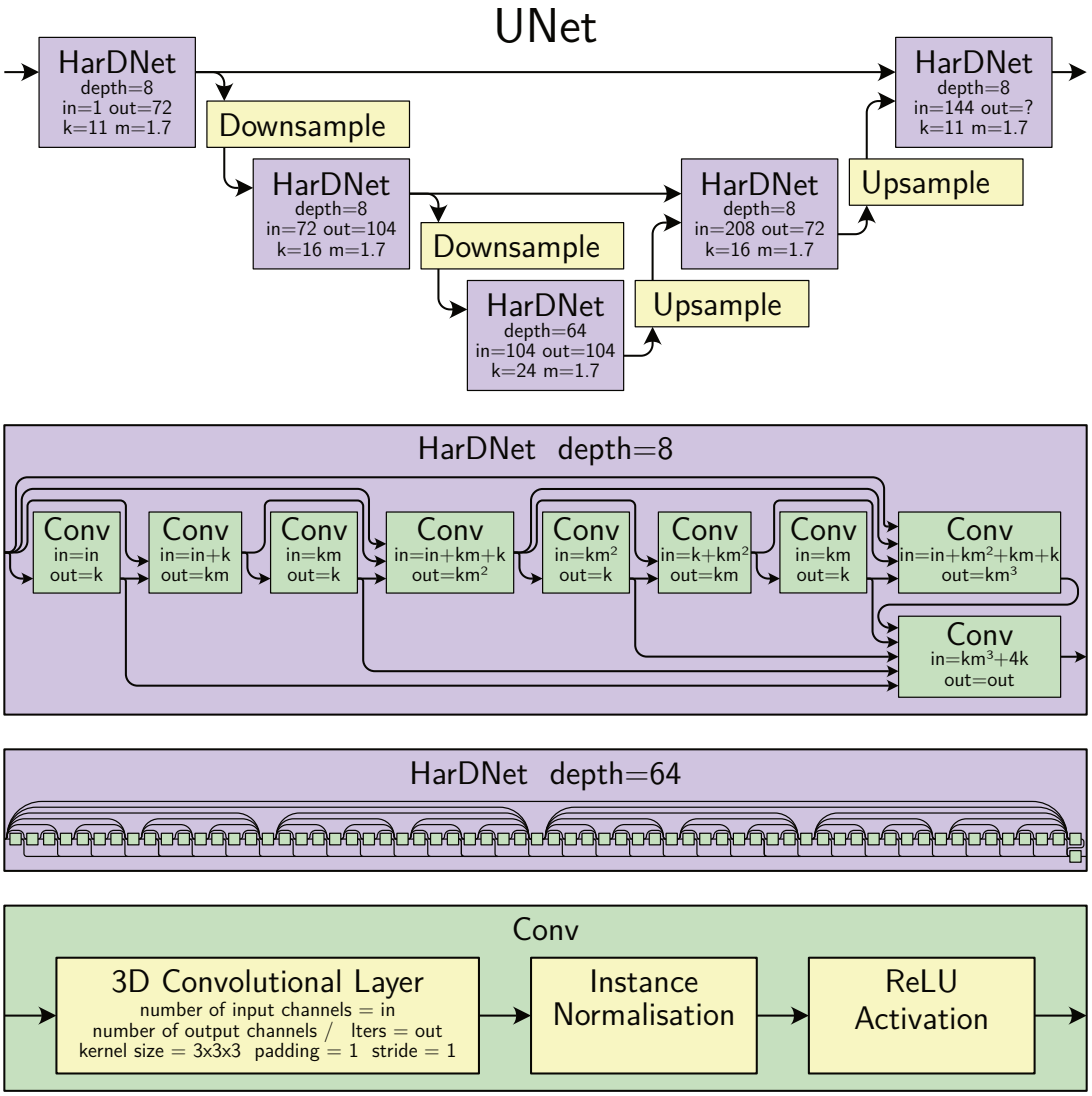


Figure 5.14: The final optimised architecture is shown in this diagram. It corresponds to row 59 in Table A.3 and is used for all subsequent experiments where a model trained on synthetic seismic data is deployed.

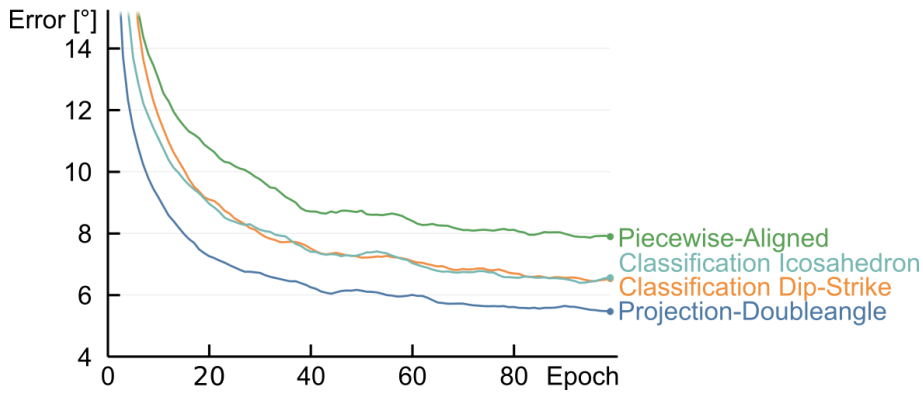


Figure 5.15: The performance during training on the 3D Synthetic Seismic dataset. The errors shown are averaged over 4 runs of the same model.

Orientation Representation	Fault IOU	Average Orientation Error	Max Orientation Error
Projection Doubleangle	66.5%	5.28°	67.6°
Projection Doubleangle	65.0%	5.64°	68.1°
Projection Doubleangle	66.1%	5.71°	68.7°
Projection Doubleangle	66.9%	5.13°	66.7°
Classification Dip-Strike	65.3%	6.74°	73.8°
Classification Dip-Strike	66.4%	6.58°	73.6°
Classification Dip-Strike	65.5%	6.59°	73.8°
Classification Dip-Strike	66.8%	6.00°	70.2°
Classification Icosahedron	65.1%	6.76°	73.5°
Classification Icosahedron	65.8%	6.39°	70.5°
Classification Icosahedron	66.1%	6.61°	73.5°
Classification Icosahedron	66.4%	6.52°	71.0°
Piecewise-Vector	65.6%	8.42°	77.2°
Piecewise-Vector	66.0%	7.77°	77.7°
Piecewise-Vector	66.0%	7.38°	75.6°
Piecewise-Vector	65.5%	7.92°	78.2°
Z-Aligned Vector	65.4%	8.13°	74.8°
Z-Aligned Vector	66.5%	7.99°	74.6°
Z-Aligned Vector	66.9%	7.37°	74.7°
Z-Aligned Vector	66.2%	7.35°	72.7°
Dip 90 Strike 360°	65.5%	7.94°	73.7°
Dip 90 Strike 360°	67.0%	7.26°	73.4°
Dip 90 Strike 360°	65.4%	7.82°	73.9°
Dip 90 Strike 360°	65.9%	8.02°	75.5°
Dip 180° Strike 180°	66.7%	7.98°	66.9°
Dip 180° Strike 180°	67.2%	7.33°	66.3°
Dip 180° Strike 180°	66.6%	7.57°	66.1°
Dip 180° Strike 180°	65.2%	8.47°	68.3°

Table 5.1: The results for the Validation set of the Synthetic Seismic Dataset. Each row represents a separate model that was trained from scratch.

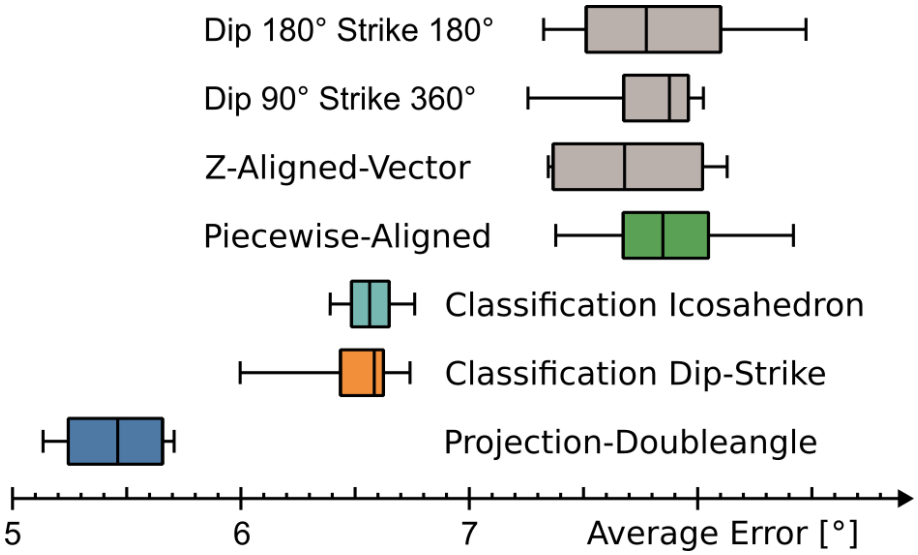


Figure 5.16: A distribution of the final performances on the 3D Synthetic Seismic dataset over the 4 runs of the same model.

Orientation Representation	Fault IOU	Average Orientation Error	Max Orientation Error
Projection Doubleangle	66.1%	5.44°	67.8°
Classification Dip-Strike	66.0%	6.48°	72.8°
Classification Icosahedron	65.9%	6.57°	72.1°
Piecewise-Vector	65.8%	7.87°	77.2°
Z-Aligned Vector	66.3%	7.71°	74.2°
Dip 90° Strike 360°	66.0%	7.76°	74.2°
Dip 180° Strike 180°	66.4%	7.84°	66.9°

Table 5.2: The results for the Validation set of the Synthetic Seismic Dataset. Each row represents the average performance of the 4 models with the specified orientation representation.

does however shift the emphasis from the requirements on the representations. It is less important to very finely distinguish similar orientations, and more important to make it easy for the neural network to predict the representation effectively. The following are my observations and thoughts about the individual representations' results.

When looking at the imperfect representations (Piecewise-Aligned, Dip90-Strike360, Dip180-Strike180), they perform similarly well to each other and to the Piecewise-Aligned representation, which was not the case on the toy task. I suspect this is because not all orientations are equally represented in the synthetic dataset. Specifically, neither completely vertical nor completely horizontal faults are represented in the dataset. The orientations that are most problematic with these orientations are thus avoided, making the orientations more effective. At the same time, the three imperfect orientations are most similar to the Piecewise-Aligned representation in nature, which would explain why they all perform similarly well.

The Classification Icosahedron and Classification Dip-Strike representations perform similarly well on the synthetic seismic data, unlike on the toy dataset, where the Classification Dip-Strike representation was superior. Notably, in Figure 5.11, we can see that for the first few epochs of training on the toy task, the two classification representations performed more similarly. A possible explanation for these performances is that the Classification Dip-Strike representation can be better optimised for distinguishing very similar orientations. This is not very beneficial on the synthetic seismic data, which is a much more difficult task where we never reached such small orientation errors, leading to similar performance for both representations on the synthetic seismic dataset.

Lastly, the Projection-Doubleangle representation is superior across the board. It both trains quickly, as seen in Figure 5.15 and also reaches the best performance for both tasks. It uses the unique approach that generalises the doubleangle representation from 2D into 3D which seems to be very effective.

A potential shortcoming of this experiment is that the model architecture was optimised using the Piecewise-Aligned representation. However, we see similar performances for the fault location in all runs in Table 5.1. This suggests that the

architecture is similarly effective regardless of the representation used. Also, the representation that it was optimised for is not the best performing representation, which is why we do not have to be concerned about biasing the results for the best representation. We can also observe very high maximum orientation errors for all representations, but this could just be a feature of the synthetic seismic task, which is difficult. The fact that the maximum orientation error is lower for the runs with lower average orientation errors suggests that we could equivalently use it as a metric and come to similar conclusions.

CHAPTER 6

Instance Segmentation

When performing seismic interpretations, one of the advantages that algorithms have over people is, that they can easily work with large amounts of data at the same high precision and fidelity as they would with smaller tasks. This is especially true when working with 3D data that is difficult for people to work with and visualise. This is because 3D seismic data is volumetric, with a scalar amplitude value for every voxel in the volume. As such, without first interpreting the data, either using automated tools or with hand-drawn annotations, there are no surfaces that can be rendered in 3D. Instead, we must look at slices through the volume to inspect the data in the middle of a volume. However, what complicates things further is the way that faults are seen in the volume, as a discontinuity in the seismic reflectors, which can be seen as lines in the images. These can only be seen effectively if a slice through the volume is made that is perpendicular to the fault plane. It is therefore important to slice the volume in a variety of directions to identify all faults. This is further complicated with faults not being straight planes, but having curvature and complicated geometries, meaning that the optimal direction to identify a fault changes depending on the location of a fault. The algorithms therefore have a clear advantage over the human approach of looking at slices through the volume, since

they can effectively interact with the 3D data directly, in a 3D context.

When hand annotating faults, the geologist must go through the entire volume of interest and highlight the location of faults slice after slice, over the whole volume. Many times they must also consider varying slicing directions over the same volume, to accurately capture all faults. This is a very tedious task. Instead, to save time, geologists commonly annotate faults at a lower precision than voxel precision, annotating a handful of points on the fault and drawing straight lines between them. This also allows them to skip multiple slices. On the contrary, the deep learning models that I train create predictions for every individual voxel in the volume.

The voxel-wise fidelity of the deep learning is both a blessing and a curse though. While offering a high fidelity, information about which voxels belong to which fault is getting lost. This means that the prediction is not separated into distinct faults that are 2D planes in 3D. This makes it very difficult to visualise the predictions of a deep learning model, since a lot of occlusion may occur between different faults. One effective strategy is to look at slices through the volume. However, this way we lose interesting high fidelity detail about the faults, such as their orientation or curvature. In turn we lose one of the advantages that the deep learning model offers over human interpretations and annotations.

Instead, if we were able to separate individual faults from a model's prediction, we could visualise them separately in 3D and avoid issues with occlusion. To achieve this prediction, we could look at instance segmentation techniques. We could treat the faults in the volume as different instances of a fault within a binary classification context.

There are many existing deep learning instance segmentation techniques that are most commonly trained on 2D images. However, they fundamentally rely on ground truth labels as annotated data for supervised learning. These are even more difficult to source than annotations of faults in seismic data in the first place. Additionally, due to memory constraints, when working with 3D data, the volumes analysed in a single pass must be quite small. However, faults are thin planar structures that span very large parts of the volume. We can therefore not identify a whole fault instance with deep learning based instance segmentation techniques directly, since

the volume that spans the fault could not fit into memory of a GPU. Instead, we must stitch together many parts of a volume that are analysed directly, using a broader post processing approach.

I must therefore use a post-processing algorithm on top of a deep learning model's predictions that can separate faults into instances. Additionally, the post processing algorithms could also make use of the fact that faults are planes in 3D. Specifically, I should be able to separate fault instances using the orientation of faults from Chapter 5, together with some other features in the form of pixel-wise binary labels akin to a semantic segmentation. In fact, one of the purposes of designing the orientation representations from Chapter 5 was for performing this instance separation.

A property that we can exploit of fault planes in 3D is that if they are parallel, then they cannot intersect. In other words, the orientation of two intersecting planes must be different. If we train a model to predict the orientation of fault planes together with the location of the fault planes and with places of intersection, we can make use of the orientation predictions to refine where intersections occur as well as match or distinguish different faults.

Similarly to orientation representations, the chromosome dataset can be used as a 2D simplification of the 3D problem of separating seismic fault planes. However, unlike the orientation representations, there are bigger differences between these two instance separation tasks with their unique challenges. They do share the same key principles though, most importantly the use of orientation to distinguish the separate instances.

6.1 2D - Separating Overlapping Chromosomes

Karyotyping is a process in which an image of chromosomes is taken from a cell and the chromosomes are laid out and sorted according to their size. It is used to visually identify numerous defects, such as genetic abnormalities [112] or cancer [113]. One of the difficulties with karyotyping is that the chromosomes are positioned randomly in the image and can touch or overlap. It is often simpler to prepare multiple images of chromosomes from the same patient and to look for ones without overlaps [114, 115]

than to segment overlapping chromosomes. Numerous image processing algorithms have been proposed to separate overlapping chromosomes [116–121], however, the task remains an open problem.

Using standard instance segmentation approaches proves difficult when dealing with overlapping chromosomes for a similar reason to separating intersecting fault planes: the overlapping features are not sufficiently differently localised in space. Of particular difficulty is the fact that the overlapping chromosomes can have the exact same centre-point and a lot of the visible area of the chromosomes may be overlapping, which makes the use of bounding-box based methods difficult. Additionally, approaches developed on vision tasks don't have to be concerned with the shape of an object that is obscured by another object, unlike on the chromosomes, where we must accurately reconstruct the entire shape of both chromosomes.

Instead, existing deep learning approaches for overlapping chromosome separation focus on the use of semantic segmentation. This can be effectively used to segment the chromosomes into three semantic categories: the background, pixels where multiple chromosomes overlap, pixels with a single chromosome. Some existing methods [122, 123] attempt to use semantic segmentation approaches to also separate the two overlapping chromosomes as different semantic classes: chromosome 1 and chromosome 2. I prove that this is a problematic approach and that there are issues with Pommier's dataset 3.1.2 that they used. It allowed undetected overfitting to falsely inflate performance metrics, which also means that the method does not generalise.

Instead of using semantic segmentation to distinguish the chromosome instances, I propose predicting the orientation of the chromosomes together with a 3-class semantic segmentation which are used by a post-processing algorithm do distinguish the chromosome instances.

The model architecture used in all the experiments with chromosomes is the same as the one used in Chapter 5 and can be seen in Figure 4.8.

6.1.1 Semantic Segmentation

Semantic Segmentation is an integral part in my orientation-based segmentation approach. However, this section explores the use of segmentation to directly distinguish chromosomes instances as semantic categories. This approach has been used to separate Pommier's dataset in existing literature [122, 123]. I argue that this approach is fundamentally flawed, even with modification that address some of its issues.

The semantic segmentation approach discussed in this section takes images of pairs of chromosomes that are overlapping. These two chromosomes are then labelled as distinct semantic classes: "chromosome 1"("ch1") and "chromosome 2"("ch2"), together with a "background" and an "overlap" class. The first issue with this approach is that it is unclear which of the two chromosomes should be labelled as "ch1" and which as "ch2". This ambiguity is the fundamental source of the issues with this approach.

Additionally, this semantic segmentation approach also suffers from another downside. It requires the input images to already be cropped to precisely two chromosomes that are overlapping. It is common to use object detection models, such as the YOLO models [124], to identify bounding boxes around the chromosomes. However, in addition to this extra step required in the pipeline, there is another issue with cropping the images to have exactly pairs of chromosomes present: we cannot guarantee that other chromosomes won't be present in the cropped image. A lot of the difficult cases are clusters of chromosomes rather than just pairs of chromosomes overlapping. The model may consequently struggle to distinguish which chromosomes are the intended targets of the semantic segmentation and which should be classed as the background.

What is most confusing, however, is that this semantic segmentation approach is seemingly effective when used on Pommier's dataset [122, 123], generalising well from the Training set to both Validation and Testing sets. However, as previously mentioned in Chapter 3.1.2, the entire dataset of 13434 images is created synthetically from only 46 source images of chromosomes, overlapping different pairs of chromosomes in different orientations. This means that the images in the dataset

are not truly independent and overfitting on the training set could improve performance on the validation and testing sets. In order to demonstrate that this is indeed happening, I created a separate dataset from Pommier's source data, as described in Chapter 3.1.2, which I will refer to as the synthetic chromosome dataset. I also created a small dataset of real overlapping chromosome images from Pommier's source data, which I will refer to as the real chromosome dataset. Experiments on these datasets create a more realistic baseline for my orientation-based segmentation to improve upon.

In order to demonstrate the problems with both the approach and Pommier's dataset, we need to compare model performance between the different datasets. If models trained on Pommier's dataset perform well when testing on Pommier's dataset, but do not translate well to the Synthetic dataset, then we can see that the domain adaptation from one dataset to the other could be the issue. However, these issues with domain adaptation should also cause similar issues when training on the Synthetic dataset and testing on Pommier's dataset. If we don't observe a similar drop in performance in this case, then it is highly likely that domain adaptation isn't the main cause of the drop in performance and that the undetected overfitting is the issue.

Furthermore, we still have to address which chromosome to label as "ch1" and which "ch2". Training the model using arbitrary labels would only confuse the model. Instead, we could introduce comparison rules that determine the labels. I use the following in my experiments: length-wise (longer/shorter), orientation-wise (more/less vertical), position-wise (rightmost/leftmost), and randomly as a control. These could alleviate the issue with treating the two chromosome instances as different semantic categories. However, note that at evaluation we do not care which chromosome is assigned which class, as long as it is consistent within the same image. We therefore try both possible assignments and pick whichever has the better metrics for each image at evaluation. To determine the effectiveness of the comparison rules, we can compare the performance of the model on the individual classes "ch1" and "ch2" with the performance on the union of the two classes "ch1+ch2". Any discrepancy between the IOU score on these classes is

Table 6.1: Semantic segmentation results, shown as IOU % scores. The individual semantic categories are: background, chromosome 1, chromosome 2 and overlap. ch1+ch2 represents the area covered by merging the ch1 and ch2 categories.

Training Dataset	**Testing Dataset**	**Average IOU**	**Individual IOUs** back.	ch1	ch2	over.	**ch1+ch2**
Pommier's	Pommier's	**92.4**	100	90.7	96.4	82.6	97.7
	Synthetic	43.2	86.6	26.1	25.2	34.9	44.4
	Real	35.9	66.5	25.8	22.3	29.0	51.4
Synthetic Length-wise	Pommier's	51.1	92.4	33.1	48.3	30.5	55.2
	Synthetic	71.2	98.8	69.5	56.9	59.8	88.1
	Real	50.2	84.3	42.1	38.7	35.8	67.3
Synthetic Orientation-wise	Pommier's	60.0	97.3	44.8	58.8	39.2	63.5
	Synthetic	75.3	98.8	72.2	70.5	59.7	87.7
	Real	**56.8**	84.2	49.8	53.4	39.8	67.7
Synthetic Position-wise	Pommier's	55.2	97.5	38.7	47.9	36.6	62.2
	Synthetic	**77.3**	98.8	74.2	76.2	60.0	88.0
	Real	52.7	84.5	43.9	43.3	39.2	67.9
Synthetic Random	Pommier's	50.9	95.5	29.5	46.3	32.1	61.3
	Synthetic	64.6	98.7	53.4	52.1	54.0	87.5
	Real	46.5	82.3	35.6	34.9	33.5	65.1

caused solely by the model confusing which of the two chromosome classes to assign to a pixel.

Results

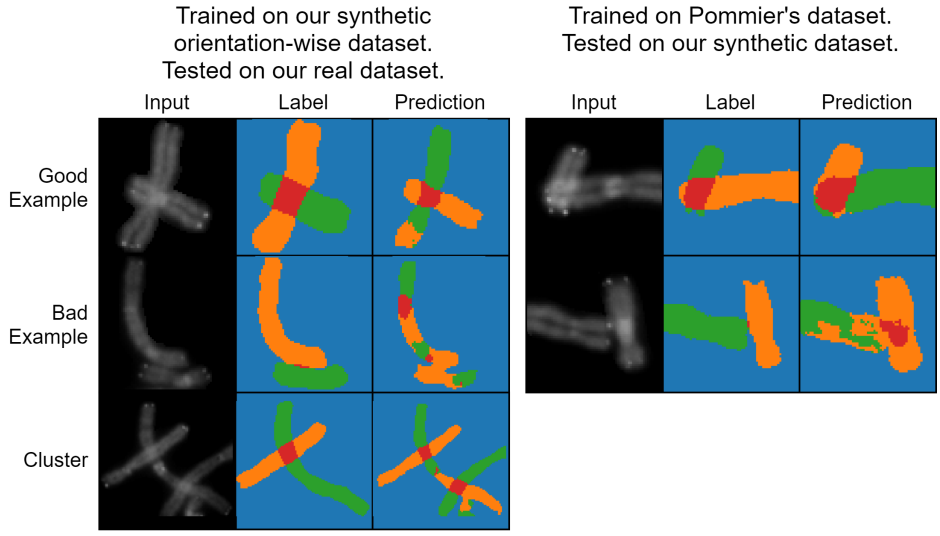
Table 6.1 shows the metrics from my experiments, while Figure 6.1 shows examples of images and their segmentations.

Discussion

Firstly, let us compare my results on Pommier's datasets with the relevant results in literature. My performance of 92.4% is comparable to the 94.7% reported by [122] and the range of 90.63% - 99.94% reported by [123]. This justifies that criticisms of my models trained on Pommier's dataset most probably apply to the cited approaches as well.

Next, let us compare the drop in performance caused by the domain shift between the datasets. For the model trained on Pommier's dataset, there is a drop

Figure 6.1: Semantic segmentation result examples.



of 49.2% IOU between the testing on Pommier's dataset (92.4%) to testing on the Synthetic dataset (43.2%). When training on the synthetic dataset, the drop in performance lies between 13.7%(Random) and 22.1%(Orientation-wise). The drop is therefore almost 2x bigger in the best case and over 3x bigger in the worst case when transferring from Pommier's dataset to the synthetic dataset than the other way around. This highly suggests that there are issues with Pommier's dataset and that it does not actually generalise. We also see that the model trained on Pommier's dataset generalises more poorly to the Real dataset (35.9%) as opposed to the models trained on the Synthetic dataset (46.5% - 56.8%). Notably, even the model trained using random assignment of "ch1" and "ch2" classes performs better on the Real dataset than the models trained on Pommier's dataset. This may be due to the larger size of the synthetic dataset in comparison to Pommier's dataset.

Let us compare the different class assignment rules. Orientation-wise assignment yielded the best performance on the real dataset, while position-wise assignment yielded the best performance on the synthetic dataset. Notably, all three rules performed better than the control with random assignment, with an improvement of 6.6%, 10.7% and 12.7% for the synthetic dataset and an improvement of 3.7%, 10.3% and 6.2% for the real dataset, respectively for length-wise, orientation-wise and position-wise assignment. This suggests that the ambiguity in the random assignment is indeed problematic, and that the assignment rules address the issue,

at least partially.

Lastly let us look at the difference between the performances on the "ch1" and "ch2" classes separately as opposed to the union "ch1+ch2" of the two classes. For the synthetic dataset we find a difference of 16.6%, 9.4%, 6.2% and 25.2%, and for the real data 20.3%, 10.7%, 17.6% and 23.4%, respectively for length-wise, orientation-wise, position-wise and random assignment. Larger values suggest that the model is more confused with which of the two chromosome labels to assign. These results similarly show that position-wise assignment performed best on the synthetic dataset, while orientation-wise assignment performed best on the real dataset. Nevertheless, the difference between these scores is significant, which suggests that the models still struggle to distinguish the two classes and that the semantic segmentation approach is still flawed.

Conclusion

Despite the improvement that the class assignment rules offer, the approach is still fundamentally problematic. Firstly, there are instances where the properties used for the assignment rules are very similar for both chromosomes. Two chromosomes can have the same length, orientation or position. In these cases the assignment is still a source of uncertainty. Secondly, using a semantic segmentation in this way requires the image to be cropped around the pair of overlapping chromosomes. This could be done with a separate model, however it also causes the issues with larger clusters, where more than two chromosomes are visible in the cropped image, which the model wasn't trained to deal with. This can also be seen in Figure 6.1, where the model is expected to only highlight two chromosomes in the cluster, but struggles to tell which chromosomes cropped around. The best way to use semantic segmentation is to distinguish the semantic categories: background, unique-chromosome and overlap, which can be applied to images with any amount of chromosomes, and rely on other methods to separate chromosome instances.

6.1.2 Orientation-based Separation

My proposed approach for separating overlapping chromosomes is to predict the location of the chromosomes using semantic segmentation without distinguishing the chromosome instances. In addition to the semantic segmentation, we can use the representations discussed in Chapter 4, specifically the Doubleangle representation 4.1.1, to predict the orientation of the chromosomes. I then use all of these predictions in a post processing algorithm to separate the chromosome instances in the image.

The semantic segmentation is used to distinguish the following three semantic classes: "unique chromosome", "chromosome overlap", and "background". For the pixels that lie on a unique chromosome, we also predict the orientation using the Doubleangle representation. Since neural network must predict the same features for each pixel in the outputs, we ignore the orientation prediction in the pixels that are classified as "background" or "overlap".

Orientation is a useful distinguishing feature for the chromosomes, since most overlapping chromosomes will have different orientations. The only exception being parallel chromosomes that partially overlap, either side-to-side or end-to-end. In this case the overlapping area should fully separate the two chromosome instances. The key utility provided by knowing the orientation of the chromosomes is, that it can be used to match multiple separate chromosome areas that are divided by overlapping areas. Assuming that chromosomes don't bend very sharply, the orientation of the chromosome before and after an overlap will be mostly the same.

Predicting the exact pixels that lie on the chromosome will always be very problematic though. We must expect errors, especially near the boundary between semantic classes. This is especially problematic near the quadripoint where the "background" meets the "overlap" area and the two chromosomes. Because we cannot distinguish the two chromosomes from each other using the 3 semantic classes, the two distinct chromosome regions can merge together and be difficult to tell apart. To distinguish these, we also predict another region, which is the "overlap" region, but is dilated by two pixels.

Overall, the model predicts a total of six channels: two for the Double-Angle

representation, one for the dilated overlap, and the last three for the 3-class semantic segmentation. These features are then used by the post-processing algorithm, which is detailed in Algorithm 17.

In essence, the algorithm first analyses the prediction and identifies unique chromosome segments. These segments are then merged to create the predicted chromosome instances based on their orientation and relative location to the areas of overlap.

To avoid issues with noisy network outputs, we remove any areas of overlap that are too small. We then subtract the dilated overlap prediction from the chromosome class to spatially separate chromosome segments from each other. Nonetheless, the segments may not be fully disjointed, so we perform the following operation to identify them. First we determine seeding points as the maxima of a distance transform applied to the segments. These will be the points in which the chromosome segments are the widest. We then iteratively grow these segments to cover the whole segment area. Crucially, due to fluctuations in chromosome widths, multiple seeding points may exist in a single chromosome segment. We therefore merge two segments if either of their seeding points was barely a local maximum, raised only 1 pixel over the neighbouring area. Finally we grow the segments over the areas of the dilated overlap until they cover the whole chromosome areas. We further consider merging two adjacent segments if their orientation is sufficiently similar in the area where they touch. We do not apply this operation near areas of overlap where two distinct chromosomes must be present. Next we determine which chromosome segments belong to the same chromosome. We assume that every area of overlap is created by overlapping exactly two chromosomes. We therefore merge the chromosome segments with the most similar orientation until only two chromosome instances are present near every overlap. Finally we add the adjacent overlap areas to all chromosome instances.

Results

The trained deep learning model was able to achieve the following results: background IOU: 98.8%, chromosome IOU: 88.3%, overlap IOU: 58.6%, dilated overlap

Algorithm 1: Orientation-Based Instance Segmentation

Input: *orientation*, *dilated_overlap*, *background*, *chromosome*, *overlap*

remove small areas from *overlap*
distance_image ← distance transform of (*chromosome* - *dilated_overlap*)
segments ← local maxima of (*distance_image*)
for *distance* in max(*distance_image*) to 0 **do**
    dilate *segments* until they fill the area where *distance_image* ≥ *distance*
    if two *segments* touch and maximum distance in either *segment* ≤ *distance*+1 **then**
        merge the two *segments*
dilate *segments* until they fill the area of *chromosome*
for all pairs of *segments* **do**
    neighbouring_pixels ← intersection of (pair of *segments* dilated by 1 pixel)
    if *neighbouring_pixels* are not near an overlap and their *orientation* is sufficiently similar **then**
        merge the two *segments*
remove small *segments*
for all *overlaps* **do**
    while there are more than two *segments* adjacent to the *overlap* **do**
        merge the *segments* with the most similar *orientation*
separate_chromosomes ← *segments* merged with adjacent areas of *overlap*

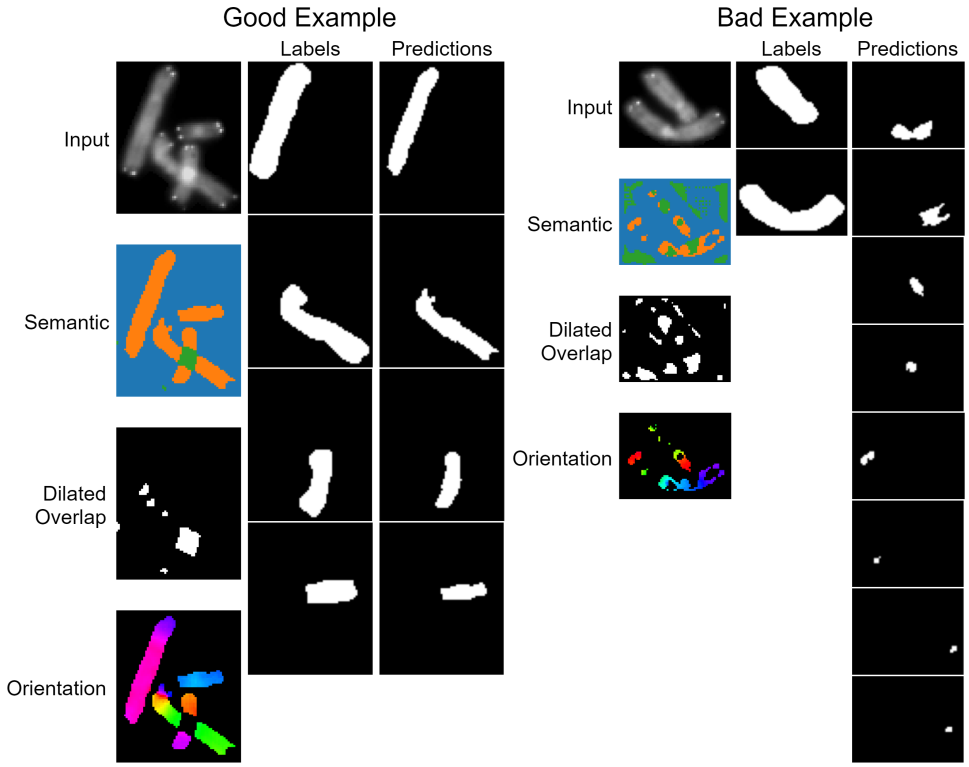
IOU: 70.4%, average orientation difference: 70.4%.

After separating the chromosomes using the post-processing algorithm, we compare the labelled chromosomes with the best matching predicted chromosome and average their IOU values. Note that any extra predicted chromosomes that did not get matched are ignored. These metrics can be seen in Table 6.2.

Discussion

The instance segmentation performs 3.6% better on the synthetic data and 20.8% better on the real data than the semantic segmentation trained on the orientation-wise synthetic dataset. When qualitatively evaluating the results in Figure 6.2, we find that the model can correctly separate some examples, but fails on others. When it does fail, the network prediction is very noisy, suggesting that the neural network failed to generalise well. The most likely cause of this is the use of the synthetic dataset for training, which does not fully resemble the real overlapping chromosomes. The dataset is also of limited size and sample variety, owing to its

Figure 6.2: Orientation-based segmentation result examples. The Labels and Predictions are the separated chromosomes, while Semantic, Dilated Overlap and Orientation are the network outputs.



source of only 15 human metaphase images, nine of which were used for training.

Ultimately, the work on the chromosomes proved a very useful test bed for the principles of using orientation as a useful feature to distinguish overlapping object instances. It leads directly to the 3D counterpart of separating faults, while also proving the utility of predicting orientation in its own right.

6.2 3D - Separating Faults

In the real world, faults occur in various orientations and often also intersect. If we visualised all the faults in 3D from a seismic survey, they would look like a big web of flat planes. Separating them into separate fault segments would make visualisations and further analysis of individual faults much easier. It is this very environment that the instance separation approach must cater to. In comparison, the synthetic data used for training my models is extremely simple. The synthetic data consists of at most 3 planes that are completely flat. Nonetheless, some models trained on

Table 6.2: The average IOU scores over the best-matching separated chromosomes.

Testing Dataset	**Semantic Segmentation (orientation-wise)**	**Orientation-Based Segmentation**
Pommier's	61.5	66.4
Synthetic	78.3	85.4
Real	57.0	77.8

synthetic data translate very well to real seismic surveys, especially convolutional models that perform localised analysis and aren't as good at taking long distance relations into account, such as the broad fault geometry. In our case this may even be advantageous due to the lack of realism in the fault geometry of the synthetic volumes.

Although the synthetic data lacks complex geometry, we can take models that were trained on synthetic data that can predict the location and orientation of faults effectively. We can then hard-code the post-processing algorithm and optimise it for the real seismic surveys. Specifically, I worked with the Laminaria seismic survey, as described in Chapter 3.2.4. Note that due to the lack of annotated data, there are no metrics that can be used to evaluate the performance of the post-processing algorithm. Instead we have to rely on subjective human evaluation of the outputs.

We can use the same models from Chapter 5.2.3 that were trained on the Synthetic Seismic dataset to predict the location and orientation of faults in the real seismic data. The networks are only modified to output additional features that are required for the post-processing algorithm. The following is a list of the predicted features:

1. At least one fault is present (fault probability)
2. More than one fault is present (intersection probability)
3. Exactly one fault is present (segment probability)
4. The orientation of the fault normal (only if 3 is True)
5. The orientation of the intersection, which is perpendicular to the intersecting faults' normals (only if 2 is True)

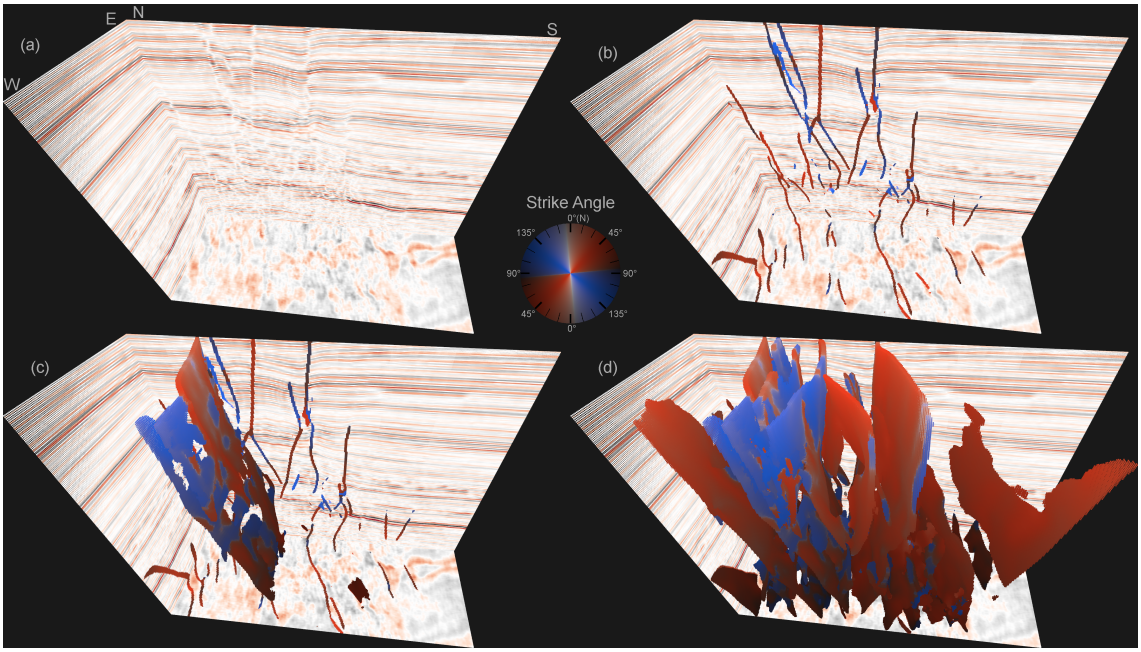


Figure 6.3: 3D visualisation of shallow faults from the Laminaria 3D dataset bordering slice (b) in Fig. 6.6. The strike angle of the faults is visualised using colour, with additional shading making deeper voxels darker. In part (a) we visualise the seismic data only as slices on the boundary of the volume. Part (b) adds the predicted faults to the slices. Part (c) adds a single fault segment in 3D, which was separated by the post-processing algorithm in Section 6.2. Part (d) visualises all of the predicted faults in 3D.

The first three features: fault, intersection, and segment probability, are predicted independently as binary classifiers,also known as a multi-label classification, and are trained using the dice loss [125]. The orientations are predicted using the Projection Doubleangle representations from Chapter 5.1.3. Figure 6.3 shows an example of a region from the Laminaria volume and the model's predictions therein.

These predictions are then fed to the post-processing algorithm which annotates all the fault voxels using unique integers, where all voxels belonging to the same fault segment are marked with the same integer. The predicted fault segments are intended to be sections of faults that do not intersect with any other faults. The post-processing algorithm takes the predicted fault locations and removes areas from the volume until its fault segments are disjointed in space. These are then enumerated and grown out to cover the original predicted volume. The steps of the algorithm are visualised in Figure 6.4 and explained below.

1. Threshold the fault probability, intersection probability, and segment proba-

bility values. Morphologically dilate the intersection volume and subtract it from the fault segment volume. The resultant binary volume of segments will further be refined until the segments are separated in space.

2. Subtract areas with inconsistent fault orientation from the segments, measured as the absolute value of the dot product between normals in a neighbourhood.

3. Connected smoothing: Set voxels as segment voxels if and only if a sufficient number of neighbouring voxels are segment voxels. This removes noisy parts of the volume and smooths out the segments.

4. Shrink the fault segments along the fault plane, which avoids thinning them. For every segment voxel, consider the points in its neighbourhood that are perpendicular to the orientation expressed as a normal vector to the fault plane. Of the considered points, count the number labelled as a segment and only keep the points where this count is higher than a threshold. Repeat this process 3 times.

5. Connected smoothing: Repeat the same operation from point 3.

6. We proceed to label each distinct segment with a unique integer. To do this in a GPU parallelisable manner, we assign every segment voxel in the volume a unique integer and assign non-segment voxels a very large integer. We then iteratively apply the min operation over segment voxels in a local neighbourhood until no more changes occur. Finally, we count which unique integer values are present in the volume and map them to consecutive values starting at 2. The background is set to 0 and segments that are too small are assigned a value of 1.

7. The segments are now uniquely labelled and returned. However, they are smaller than the original predicted segments.

8. The segments are iteratively dilated until they cover the volume of the original segment probability volume. To encourage the dilating segments to grow in a meaningful manner, an iteration of connected smoothing is applied between every iteration of dilation until no more changes occur, and return the output.

9. Finally, we repeat the dilation process, but cover the whole faults including the intersections. This is used as the third and final output.

This algorithm returns its predicted fault segments in 3 levels. The first being the core points that were used to separate the fault segments, while the second covers the whole fault segments (voxels with exactly 1 fault present) as predicted by the neural network. The third covers all the faults (at least 1 fault present), which includes areas of intersection which will be allocated to the nearest fault rather than all the faults present in the intersection area. One of the major advantages of this algorithm is that all its operations are performed in a local neighbourhood. They can be expressed as stencils and are therefore easily parallelisable on a GPU.

Due to GPU VRAM memory constraints, it is not possible to work with the whole Laminaria 3D volume at once, which has a resolution of $1444 \times 3964 \times 751$. Instead, we can turn to a sliding window approach for both the model prediction and post processing. With the model having a fully convolutional architecture, it can be applied to any resolution. With 24GB of VRAM I was able to deploy the model on volumes of shape $176 \times 176 \times 176$, and moved the sliding windows with a stride of 44. I perform a weighted average of all the overlapping sliding windows, where the weight is highest in the centre of a window and 0 at the edge. Points near the edge cannot see the full context past the edge of the window to make the prediction, which is why they have a lesser weight. The weight changes sinusoidally from zero at the edge to 1 in the centre. To average the orientations we cannot simply average their vectors, since we don't know whether to multiply them by -1. Instead, I make use of our Piecewise-Aligned representation, where taking a mean of multiple orientations in this representation yields a meaningful average orientation. To do this I translate the orientation output of the model from any trained representation into the Piecewise-Aligned representation, average the values and invert the representation back into a vector.

When performing the post-processing on the Laminaria volume, I use a sliding window of size $608 \times 608 \times 751$. Note that 751 is the full height of the Laminaria 3D volume. We slide the window with a stride of 304, where we compare the fault instances in overlapping windows. If two fault instances share more than 5 voxels

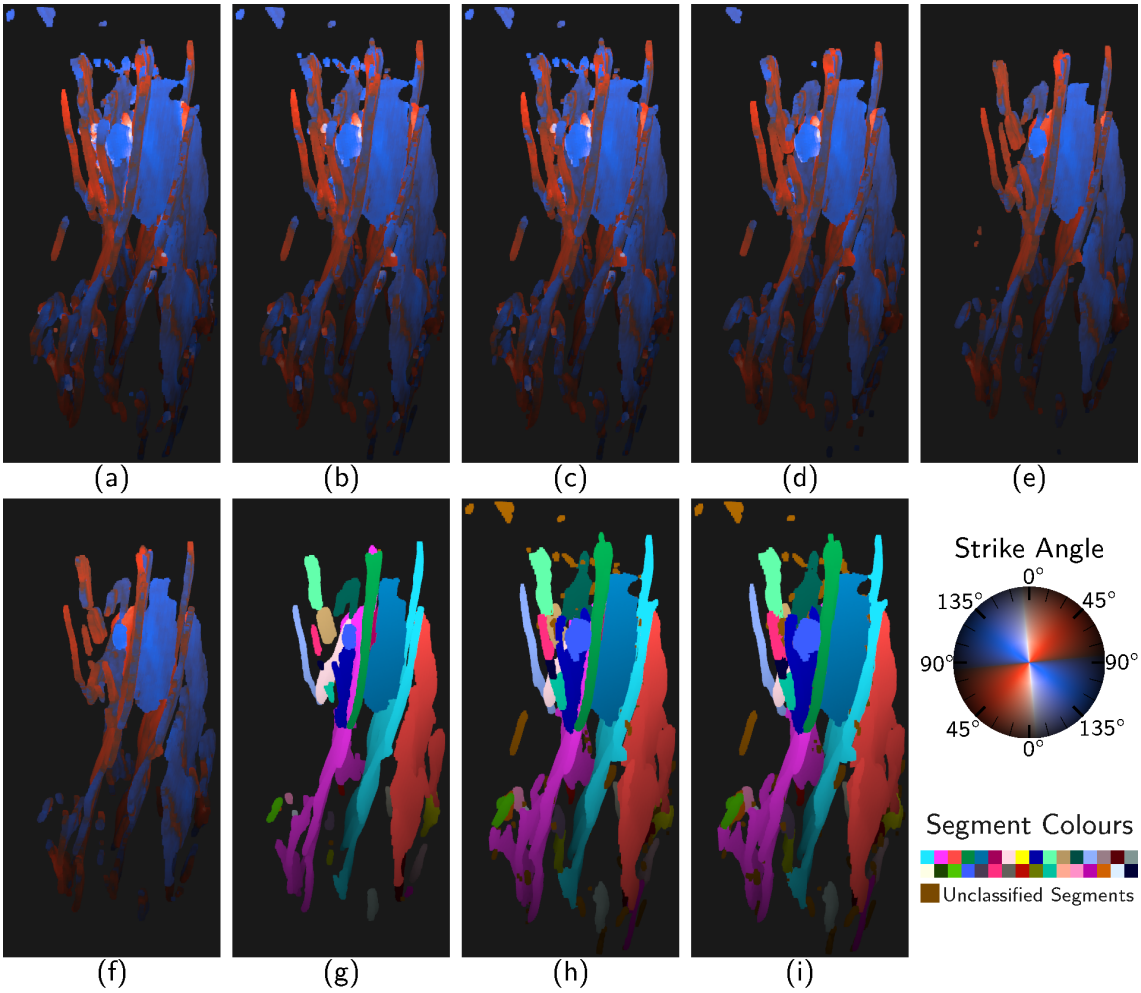


Figure 6.4: 3D visualisation of shallow faults from the Laminaria 3D dataset at different stages of the post-processing algorithm. (a) - (f) visualise the strike angle using colour, while (g) - (i) visualise distinct fault segments in different colours. Additional shading is applied making deeper voxels darker. (a) visualises voxels with a high fault probability. (b) - (f) visualise the fault segments after steps 1 - 5 of the post processing algorithm respectively. (g), (h), and (i) visualise the result of steps 7, 8, and 9 respectively.

in the core points volume (1st level output), we unify them as the same instance.

6.2.1 Results

The performance of the deep learning model that predicts the location and orientation of the faults plays the most important role. The post-processing algorithm only enriches those results by separating the predicted faults into fault segments, however, their location in the first place is of highest importance. For this reason, I spent considerable effort evaluating the quality of the fault location predictions on the Laminaria volume. I achieved the most comprehensive results with the simplest approach of directly applying the model trained on the synthetic data to the real volumes. All the results in this section were achieved with this method.

First, let us visualise the deep learning model's predictions on the Laminaria volume on three horizons through the volume in Figure 6.5. We can compare these to the interpretation by Phillips et al. [6] and Çiftçi and Langhi [126]. We can also compare the orientation of the faults on the horizons, as well as in areas around the Corallina, Laminaria and Vidalia wells in Fig. 6.7.

To evaluate how well the model distinguishes the faults, we analyse two slices through the volume seen in Fig. 6.6, where Ken McCaffrey, a geologist, annotated the predicted faults as true positives or false positives. He also annotated faults that the model missed as false negatives. To get quantitative measures, we count the number of faults in each category and calculate the following metrics:

$$\text{F1 Score: } \frac{2 * TP}{2 * TP + FP + FN} \ , \tag{6.1}$$

$$\text{Jaccard Index (IOU): } \frac{TP}{TP + FP + FN} \ . \tag{6.2}$$

The results can be seen in Tables 6.3 and 6.4. For comparison, on the synthetic test set, the model achieved an F1 of 98% and an IOU of 96%. Volumetrically when treating pixels independently, it got an IOU of 67%. The volumetric IOU is a much more stringent metric, which heavily penalises even small positional inaccuracies of the prediction, especially since the faults are very thin features. This is why its value is much lower than the fault counting IOU.

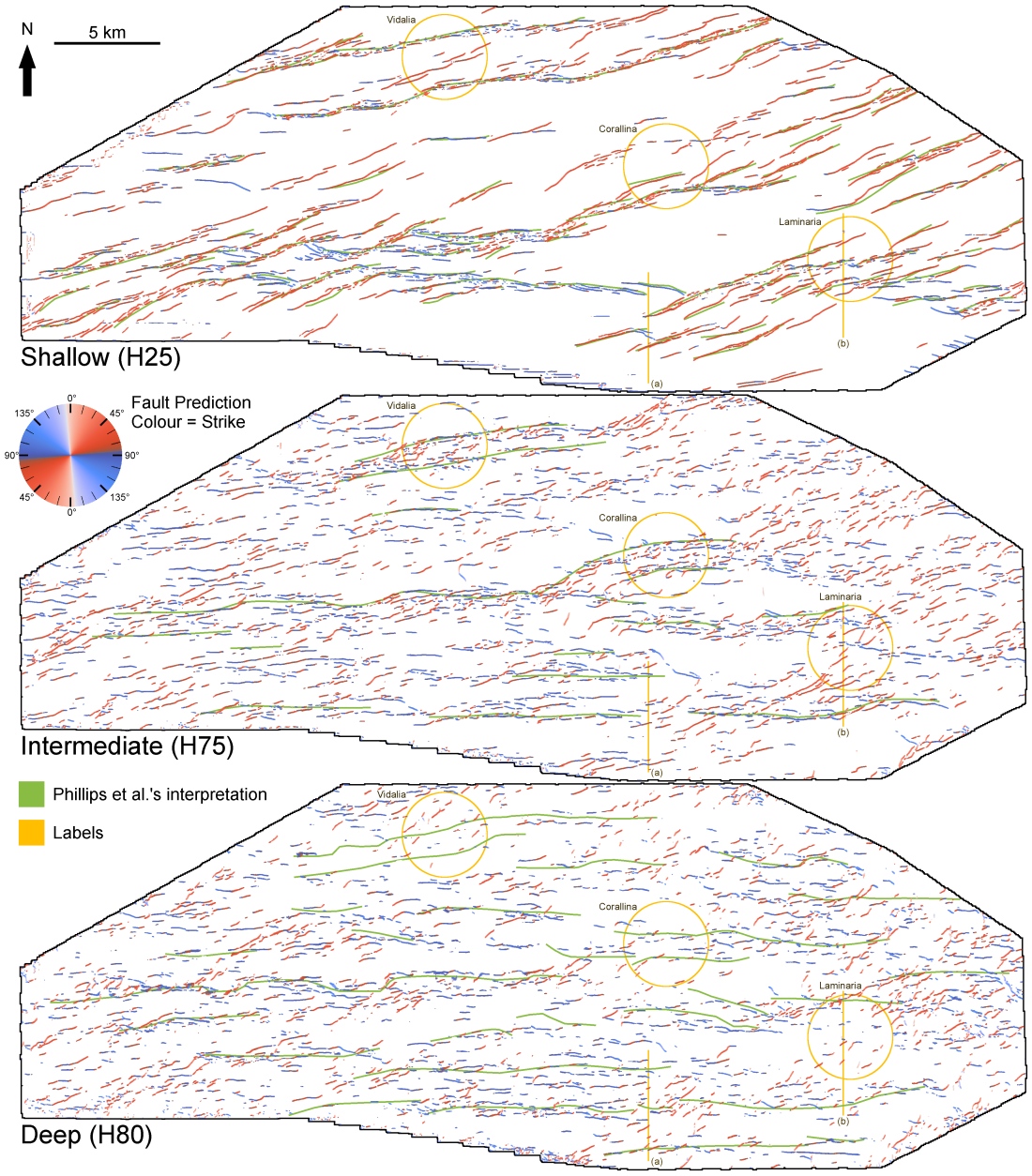
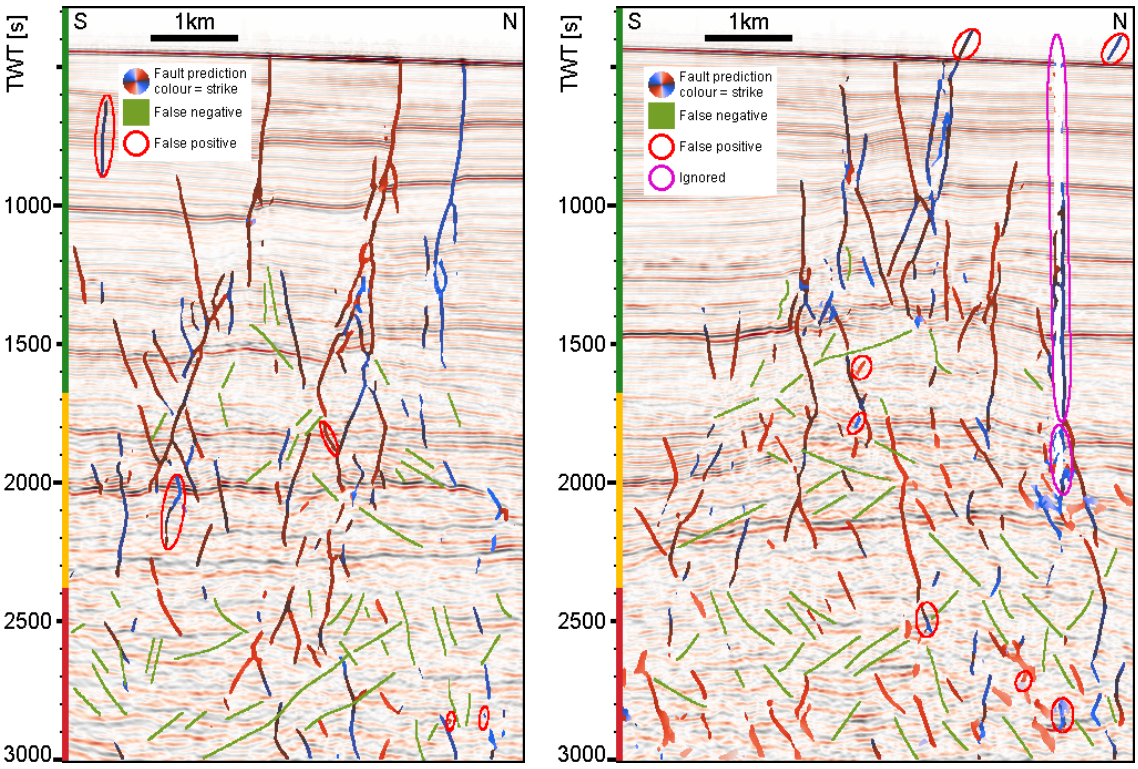


Figure 6.5: 2D map comparison between modelled fault traces and those interpreted by Phillips et al. [6]. There is generally good correspondence between the model and interpreter at shallow (H25 level, c. 1s TWT). At intermediate (H75, c. 2.3 s TWT) the level of agreement drops. The structures identified by Phillips et al. correspond to the main east-west fault traces. The Corallina, Laminaria and Vidalia (yellow circles) are where the wells were drilled in the area of this seismic volume. The model shows a lot of additional detail in the form of short E-W and ENE-WSW trending fault segments that are probably secondary faults to the main structures. At deep (H80, c. 2.6 s TWT) levels there is very little correspondence between the modelled and interpreter structures other than a broad agreement in the trend and location of some of the bigger faults. Lines (a) and (b) correspond to the two slices through the data shown in Fig. 6.6.



(a) (b)

Figure 6.6: Slice (a) and (b) taken across the Laminaria High (locations are shown on Fig. 6.5). The model fault picks are shown in red and blue. False positive structures (where the interpreter (McCaffrey) does not agree with a model pick) are shown and are overall quite low in number. False negatives (model has missed faults the interpreter would pick) increase dramatically with depth. The faults are grouped into shallow between 400-1680 on TWT scale (green), intermediate between 1680-2400 (yellow) and deep level for those greater than 2400 TWT (red) which corresponds to the levels identified by Phillips et al. [6]) and the metrics presented in Table 6.3 and 6.4.

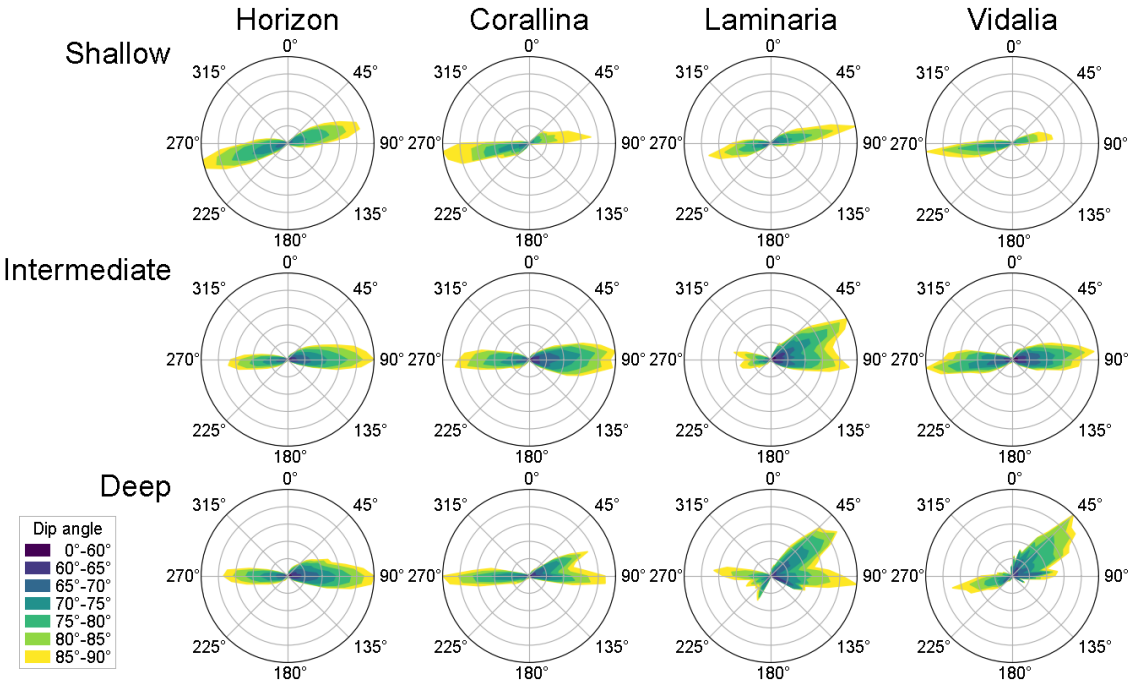


Figure 6.7: Rose diagrams showing the distribution of the strike and dip angles in the following areas: Horizon spans the whole area of the volume at shallow horizon H25 level, intermediate horizon H75 and deep horizon H80 respectively. Corallina, Laminaria and Vidalia span an area with a radius of 2km and ±100ms from the respective horizon in TWT, as seen in Fig. 6.5 in yellow.

Table 6.3: Metrics on slice (a) in Fig. 6.6.

	Shallow	Intermediate	Deep	All
Total Faults	39	52	63	154
True Positives	32	38	30	100
False Positives	2	2	2	6
False Negatives	5	12	31	48
F1 Score	90.1%	84.4%	64.5%	78.7%
Jaccard Index	82.1%	73.1%	47.6%	64.9%

Table 6.4: Metrics on slice (b) in Fig. 6.6.

	Shallow	Intermediate	Deep	All
Total Faults	55	64	68	187
True Positives	42	50	38	130
False Positives	5	1	3	9
False Negatives	8	13	27	48
F1 Score	86.6%	87.7%	71.7%	82.0%
Jaccard Index	76.4%	78.1%	55.9%	69.5%

With the help of the Department of Earth Sciences at Durham University, I collected annotations from 8 volunteers of 4 slices through the volume, which can serve as a ground truth in these limited regions. These can be seen in Figures 6.8, 6.9, 6.10. The volunteers were qualified geoscientists ranging from postgraduate students to the head of the department. They were provided with the slices as images, together with an adjacent slice from the volume for context. The each used their own tools and were asked to identify all faults present in the slices. The volunteers returned images with drawn-on lines in the locations of the faults. However, due to the variety of their preferred tools, the annotated lines were of varying widths, and some images were interpolated to a lower resolution. Because of this, I normalised the interpretations, to the same resolution and the same line thickness.

Table 6.5 visualises the IOU scores for the agreement between the interpreters and the model. Due to the high disagreement between the interpreters, we must decide what to treat as the ground truth. First, how many interpreters must agree on a fault location for us to treat it as being present as ground truth. Second, we must decide how to determine whether the interpreters and the model point to the same

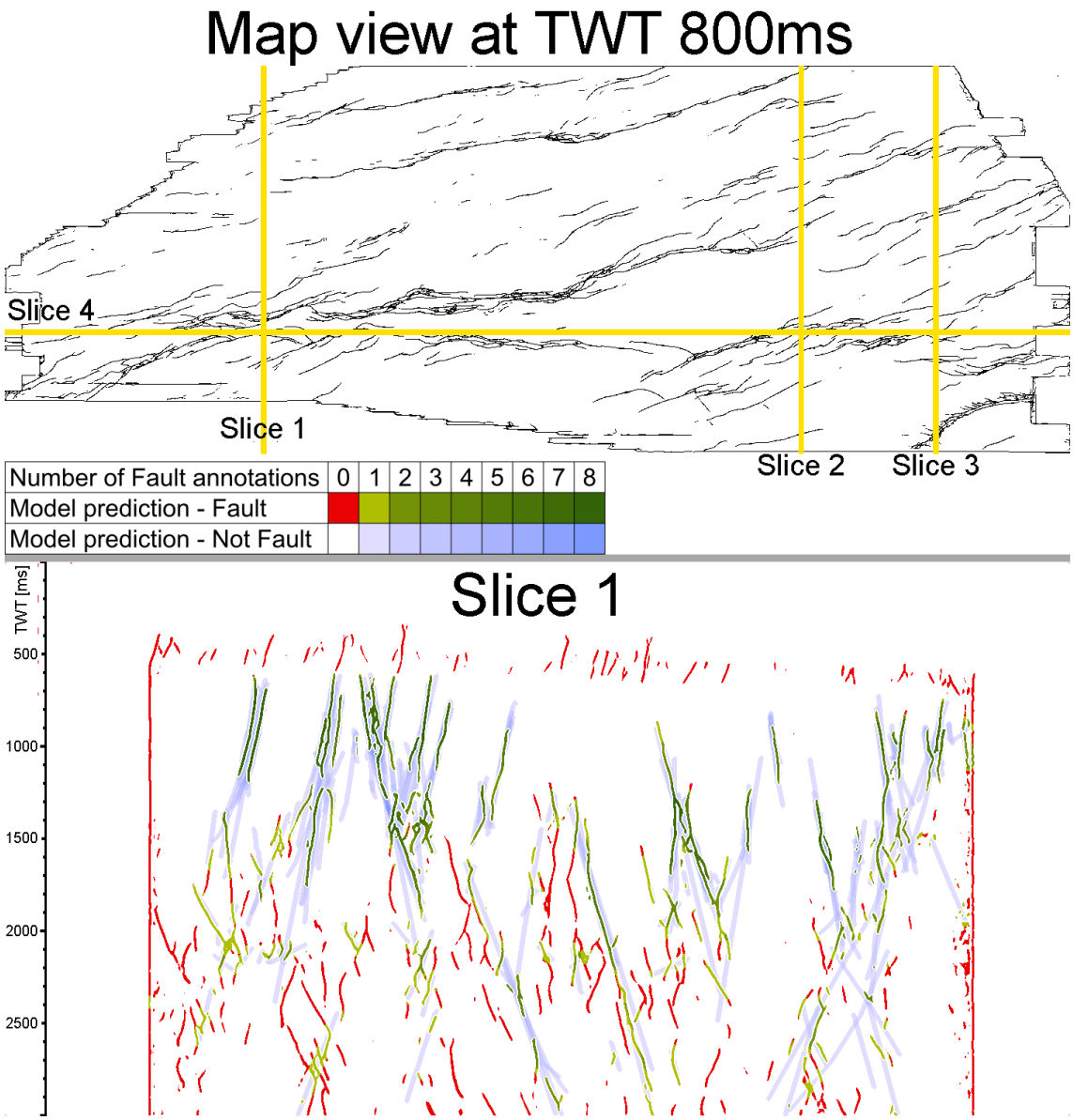


Figure 6.8: Continued in Figures 6.9 and 6.10

fault. We could treat pixels independently, however, in this way small inaccuracies in the placement of the faults can lead to complete disagreement. Instead, we could allow for any fault interpreter's fault pixels that are drawn within a fixed radius of the model's prediction to be treated as matching. Table 6.5 shows the values with a tolerance of a radius of 4 pixels.

Lastly, to evaluate the post-processing algorithm, we can look at visualisations of how the faults were separated into segments. Figure 6.11 shows small sections of the Laminaria volume with interesting geometries where faults of different orientations

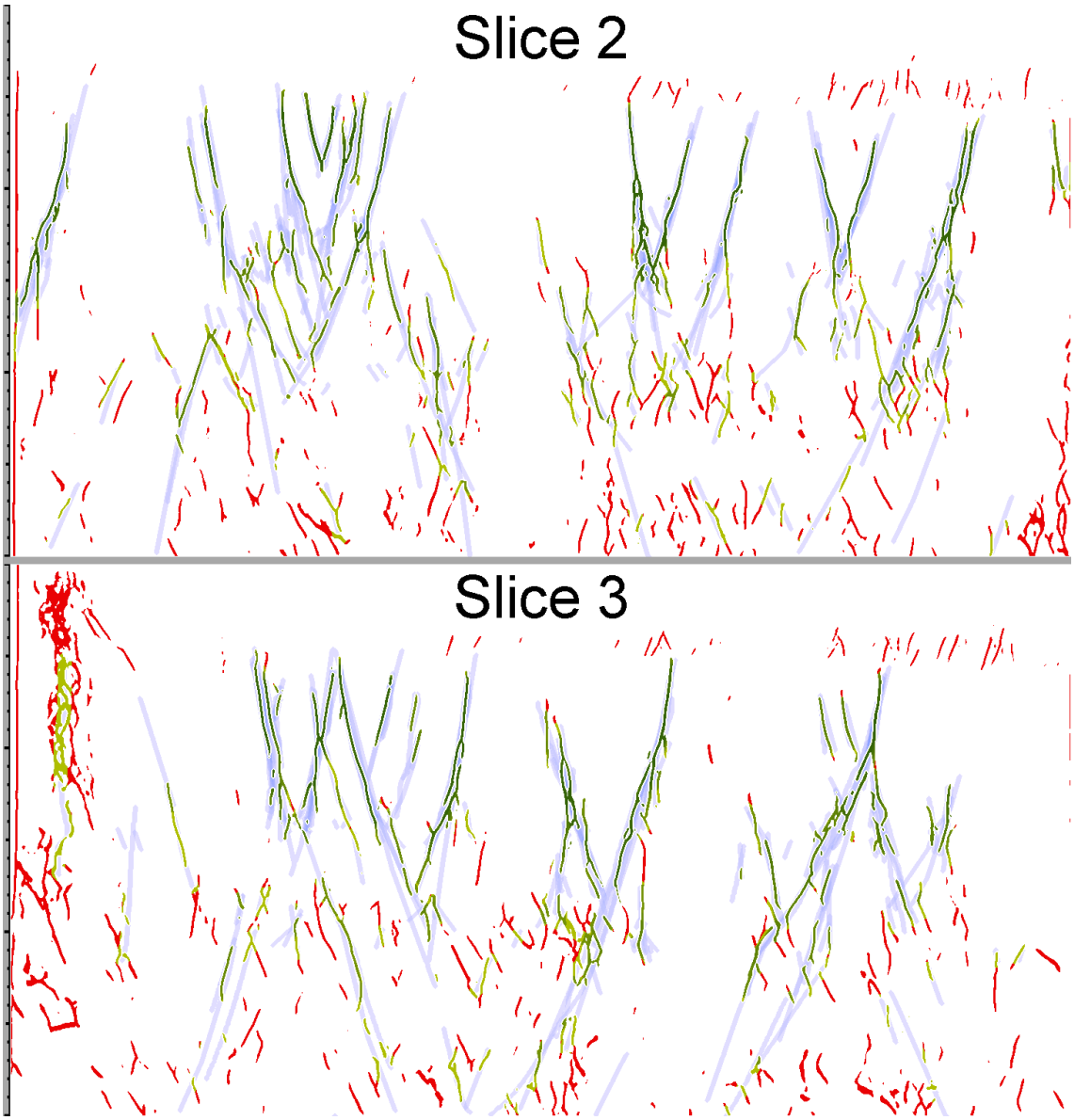


Figure 6.9: Continuation of Figure 6.8, continued in Figure 6.10

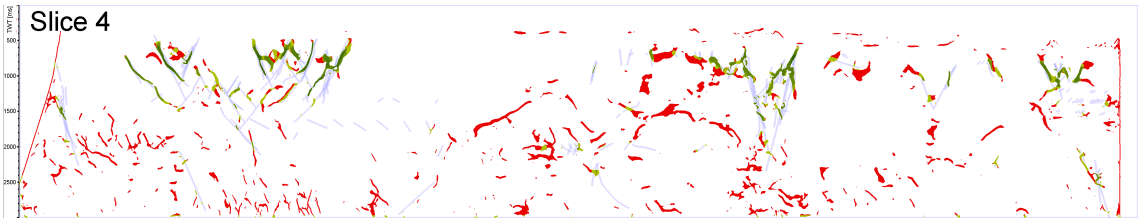


Figure 6.10: Continuation of Figure 6.8 and 6.9

# interpr.	1	2	3	4	5	6	7	8
pixel-wise								
Slice 1	14.1	13.7	9.2	5.1	2.0	0.4	0.0	0.0
Slice 2	17.6	21.2	17.1	10.9	5.6	2.6	1.0	0.1
Slice 3	14.5	17.0	13.3	8.0	3.7	1.5	0.3	0.0
Slice 4	9.6	5.5	2.4	0.8	0.3	0.0	0.0	0.0
with matching								
Slice 1	26.8	30.6	27.3	23.1	18.5	13.8	8.5	3.7
Slice 2	33.7	47.7	42.6	35.4	30.2	25.9	19.9	10.3
Slice 3	27.4	34.9	32.6	29.3	24.1	16.5	10.1	4.0
Slice 4	21.8	19.8	13.9	8.8	5.0	2.5	0.8	0.0
subjective matching								
Slice 1	27.9	31.9	28.6	24.7	17.7	13.7	10.6	6.6
Slice 2	33.8	47.7	42.6	37.1	30.4	26.9	22.1	15.1
Slice 3	28.1	36.1	31.9	29.3	23.5	15.8	7.9	2.7
Slice 4	22.2	21.0	14.5	10.1	6.6	4.1	2.1	0.8

Table 6.5: Agreement between human annotators and the model on 4 slices through the Laminaria volume as seen in Figures 6.8, 6.9, 6.10. The values are volumetric (pixel-wise) Intersection Over Union (IOU) values as percentages. The columns represent the number of interpreters that must agree on a fault to be considered as the ground truth, out of 8 interpretations. Pixels are either treated independently as seen in the upper half of the table, or we perform matching with a tolerance of 4 pixels in the bottom half of the table.

intersect. I mostly used these volumes to develop the post-processing algorithm and adjusting various parameters of the algorithm. Figure 6.4 shows a 3D visualisation of faults at various stages of the algorithm. The parameters of the algorithm were optimised to achieved the desired effect on the volume at each of these steps. It is however difficult to view the final segmentation of faults from this visualisation. Figures 6.12 and 6.13 visualise faults around near and including slice (b) as seen in Figure 6.5. These visualisations are limited to the largest faults and shallow faults respectively, since it is very difficult to visualise the whole volume in an image without an interactive 3D application. In Figure 6.14 we get an idea of what the interpretation looks like at all depths and the ability the system offers to individually visualise faults.

6.2.2 Discussion

Ken McCaffrey's Insights

We can qualitatively evaluate the predictions of the model as well as compare it against existing literature. The following two paragraphs are the analysis that was kindly provided by Ken McCaffrey and published in [5]:

We further analyse the prediction on the Laminaria 3D seismic volume where we observe that the model performs well at shallow levels. There is good agreement with a high F1 score and Jaccard Index at the shallow levels in Table 6.3 and 6.4 from Fig. 6.6. In 2D map view, the structures match well in terms of location (Fig. 6.5) and orientation (Fig. 6.7) with previous manual interpretations by Phillips et al. [6] and Çiftçi and Langhi [126]. The trends in Fig. 6.7 for the Corallina, Laminaria and Vidalia areas at the shallow level are the same ENE-WSW trends as shown in the previous work. At the deep level both E-W trending and a large number of ENE-WSW fault major structures were identified by the model, however only E-W structures were identified in previous work. This is possibly because the ENE-WSW structures are short and were considered to be secondary faults by the authors. Note the prominent structural feature where the two trends meet, as seen in Fig. 6.5 in horizon Shallow H25, which was labelled as 'intersection point' and

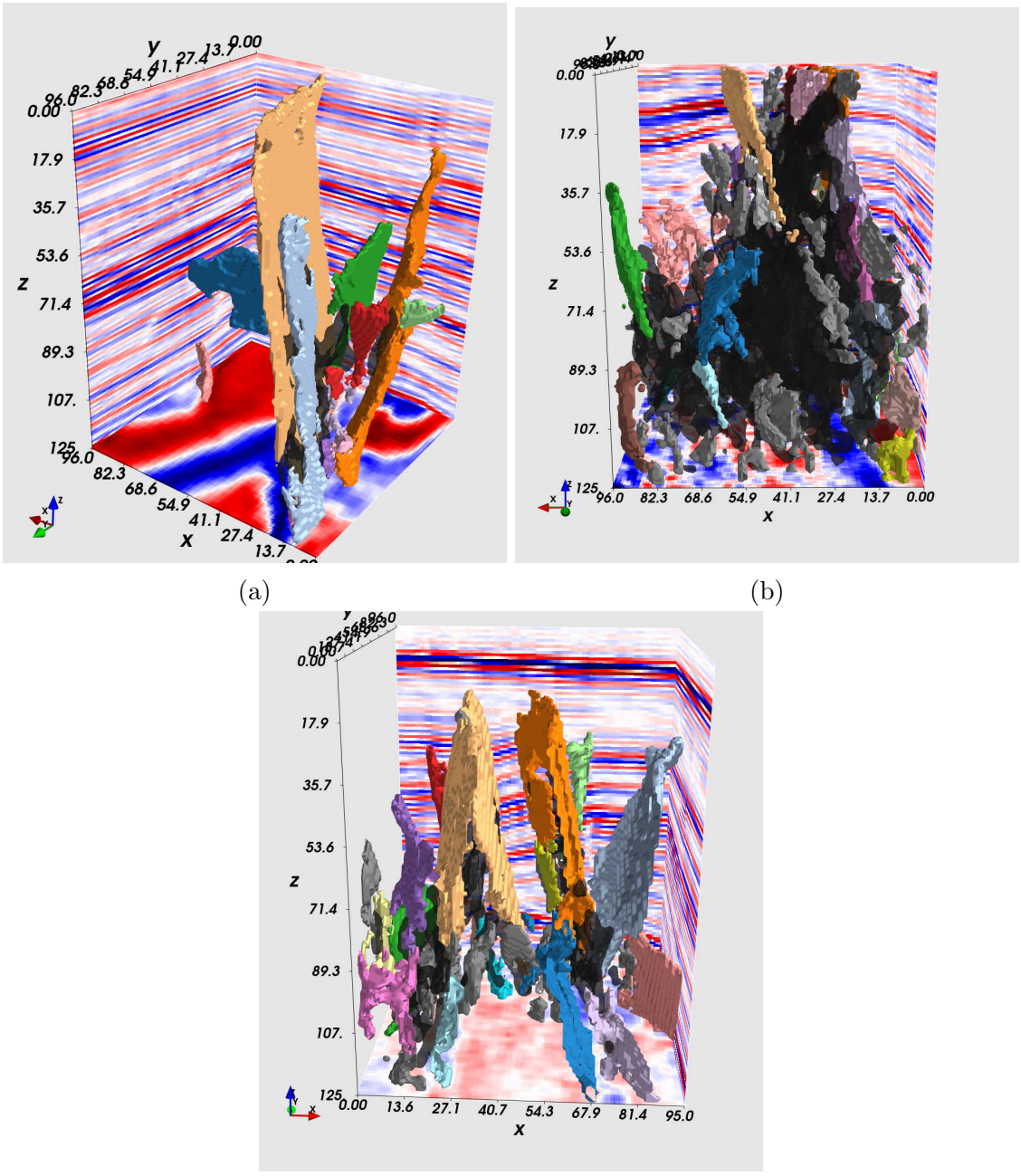


Figure 6.11: Small real seismic volumes, with the largest 20 fault segments visualised in different colours. The remaining fault segments are visualised in grey and the fault intersections are drawn in translucent black.

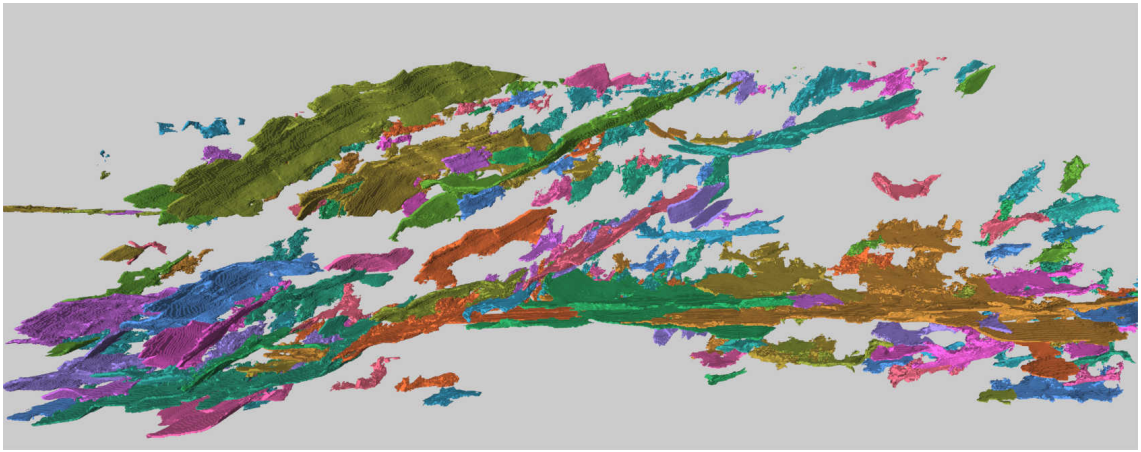


Figure 6.12: The 200 largest faults in the volume drawn in cyclically repeating colours.

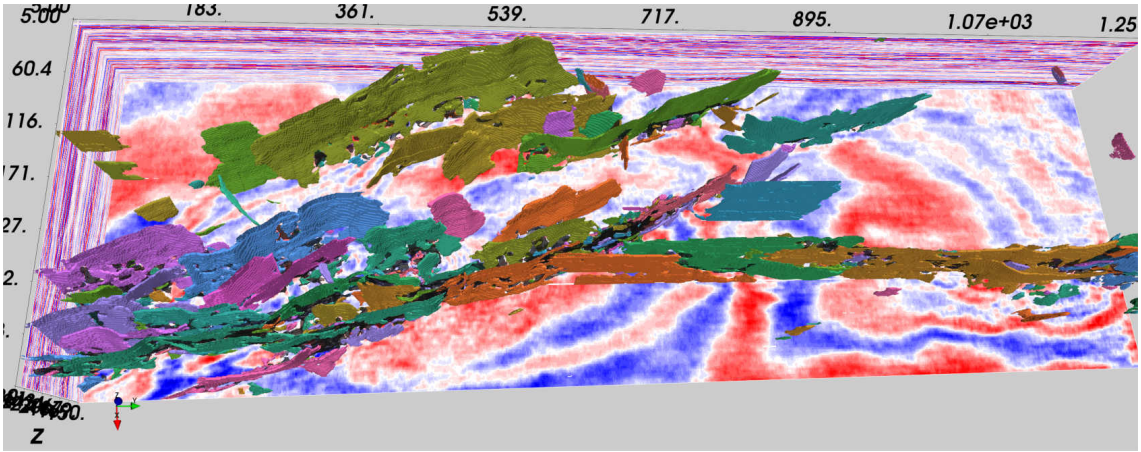


Figure 6.13: Fault segments from the upper part of the large real seismic volume drawn in cyclically repeating colours, with intersections drawn in translucent black.

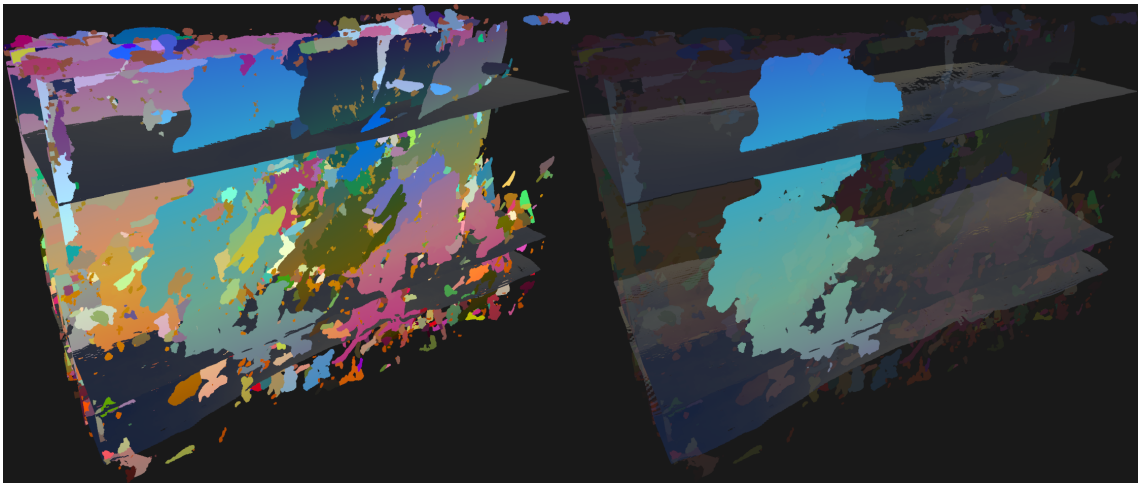


Figure 6.14: A visualisation from the Laminaria 3D volume shown at the full depth, visualising individual fault segments in different colors, as predicted by the model and post-processing algorithm. A single fault is also selected for individual visualisation, which is the main benefit of annotating separate fault segments.

'outboard en-echelon faults' by Phillips et al. [6].

There is some drop off in performance between the model and interpreter at intermediate level (Fig. 6.6, Table 6.3, 6.4), but overall the agreement remains relatively high. At deep levels, the amount of correspondence between the interpreter and the model is relatively low (Fig. 6.6, Table 6.3, 6.4). The main reason for the poorer model performance is the number of false negatives, i.e. structures that the model has not recognised that the interpreter thinks should have been there. Possible explanations for this poorer performance at deep levels might have to do with the high levels of ambient noise at depth in the Laminaria 3D seismic volume and the presence of more low dip angle structures, both of which are missing from the synthetic training data set. We can see from our modelled dip values that there are more gently dipping faults at the deep and intermediate levels compared to the shallow level. In contrast, at the deep and intermediate levels many more faults are picked by the model than shown in the published interpretations. We do not, however, consider these to be false positives because they were not identified as such in our own interpretation (Fig. 6.6). It is likely that in previous studies [6] the fault interpretation has been implicitly simplified at intermediate and deep levels to focus on primary structures, i.e., those that bound the main high structures. Our model has picked out both the major and minor (secondary) faults. This illustrates

the difficulty in comparing Deep Learning representations of faults with published interpretations.

Volunteer Annotated Data

Next, let us look at the ground truth results collected from 8 volunteers in the Department of Earth Sciences at Durham University in Figures 6.8, 6.9, 6.10 and the metrics in Table 6.5. We observe that there is a lot of disagreement between the interpreters, both in fine fault location and in fault presence. There is much more agreement in shallow faults, where many interpreters draw faults in similar, but not identical locations, resulting in a fuzzy heatmap of many parallel lines. It is not possible to distinguish whether these are all annotating the same fault or if they show the presence of multiple faults. In intermediate and deep regions, there is very little agreement between the interpreters. Many faults are only drawn in by a single interpreter, which do not always agree with faults predicted by the model. There are also many faults that the model predicted but were not annotated by any interpreters. Slice 4 in Figure 6.10 crosses many faults at very acute / obtuse angles (not perpendicular), making them much more difficult to see in the slices. This might explain why there are many more faults that were not annotated by the interpreters but seen by the 3D model.

We can quantify the degree of agreement between the model and interpreters by looking at Table 6.5. The highest agreement is achieved when treating the areas with at least 2 interpreters as true faults. Nevertheless, the pixel-wise agreement is only between 5.5% and 21.2% IOU depending on the slice. These values are very low, compared to both the results achieved on the synthetic dataset with and the slices annotated by Ken McCaffrey in Figure 6.6 and Tables 6.3 and 6.4. The volumetric pixel-wise metrics are the same as the synthetic dataset which achieved 67%. The other type of metric used on the synthetic dataset and the slices in Figure 6.6 doesn't match pixels, but rather counts entire faults as true positives, false positives or false negatives. Notably, the notion of true negatives does not exist in this setting. This is a much easier metric to achieve. It is, however, not possible to match the ground truth annotations with the model predictions using these metrics, since it is not

possible to accurately determine whether the annotated faults that are adjacent to a model's fault refer to the same fault or not. Moreover, faults can be drawn with different lengths and only partially match, or be drawn as a single fault as opposed to multiple shorter faults in different interpretations. Instead, using the 4 pixel radius of tolerance allows for a metric that sits in-between the two approaches. It is still pixel-wise in nature, but allows for a degree of inaccuracy to be tolerated. Nevertheless, the metrics that we observe with 4-pixel matching in Table 6.5 are between 19.8% and 34.9% IOU, which is still much lower than the synthetic pixel-wise 67% and especially lower than the fault-counting IOU of 96% on the synthetic dataset and 78.7% and 82.0% on the slices from Table 6.3 and 6.4, respectively.

Given these problems with the evaluation method, I argue that it is not a viable approach. The biggest reason is the extent of disagreement between the interpreters, both in terms of precise fault positioning and especially fault presence. I can see three reasons for the large extent of disagreement. First, the task is simply difficult. In Chapter 2.2.1 we have seen that uncertainty in seismic interpretations is a big problem. Second, the slices through the volume are very large, making the workload of interpreting them large. Such interpretations could be the subject of entire papers or a thesis. In comparison, the amount of time spent on the interpretations by the volunteers was much shorter than what would be required to get their best interpretations. Third, the expected amount of detail in the interpretations was not specified. Most interpreters highlighted the shallow faults that are the clearest, while only some annotated deeper faults that are much more difficult to distinguish. Overall, we can only rely on the annotations for the presence of the shallow faults with a great deal of agreement between the interpreters. We must suspend our judgement on the intermediate and deeper layers and remain sceptical about the faults that were predicted by the model but not by the interpreters. We must also be sceptical of the faults that were only drawn by a few interpreters, where the interpreters may have been overzealous with their interpretations and where the faults may not be truly present.

My Personal Insights

Outside of the metrics and comparisons from the previous sections, there are notable features of the deep learning model's predictions that should be pointed out. Most notably, the model struggles with missing data. The Laminaria volume does not cover a precise rectangle. The areas with missing data are filled in with zeros. The synthetic data does not contain such regions, which explains why the model struggles to understand the regions, labelling them as faults on the boundary of the data. This is not necessarily problematic, since the boundary around the volume can easily be identified and any predictions removed. However, what is more problematic is that the Laminaria data is missing some lines throughout the volume, in the inline direction. These are also predicted as faults by the model. Although these predictions can easily be removed, they may have an effect on the predictions in its local neighbourhood that cannot be avoided.

Post-processing Algorithm

Lastly, let us look at the results of the post-processing algorithm. We can see that the algorithm is generally effective at separating faults into segments. In Figure 6.11 we can see that it is able to distinguish segments of faults that intersect in various orientations. Figures 6.12 and 6.13 show that the approach is also effective in a broader context where patching is required to stitch the predictions of multiple smaller volumes together, due to computational memory constraints. The only instance where the post-processing algorithm struggles to distinguish fault segments is when multiple faults are highly parallel, and especially when two parallel faults merge into one at different depths. These faults can be labelled as a single fault segment in some instances.

6.3 Conclusion

Separating overlapping instances of objects is no easy task. 3D Fault separation offers some unique challenges in the field, some of which are shared in the 2D context of separating overlapping chromosomes. Predicting and using the orientation of the

faults/chromosomes to separate their individual instances proves to be an effective approach that is able to leverage the power of deep learning when restricted to training on synthetic data.

When looking at chromosome separation, the proposed orientation-based segmentation offers improvements over the semantic segmentation approach when closely scrutinised. My work proved issues with existing approaches in the field that artificially inflated the reported performance. After addressing these issues, both the existing and my proposed approach are not robust enough for deployment in industry yet, mostly due to the quality of the data available for training. However, the experiments did pave the way for the orientation-based separation and to further use a similar method for seismic interpretation.

When deploying my system for seismic interpretation on the Laminaria volume it achieves useful results. By annotating individual faults, the system is able to create a great first pass for geologists, who can then scrutinise the results rather than drawing faults from scratch. This saves on both the time required to find the faults and to enter their locations into the system. Additionally, by separately annotating fault segments as instances, small faults can be omitted from visualisations, as well as areas of intersections. When focusing on individual faults, multiple small fault segments could be selected to form a fault to be visualised in its entirety, as well as the selection of faults based on their orientation. This could prove to be very beneficial if introduced into the workflow of interpreting faults in software, such as GeoTeric.

CHAPTER 7

Discussion and Conclusion

The most significant scientific contribution of this thesis is the development of representations for orientation, or negation-invariant vectors, in both 2D and 3D. These representations are not only directly applicable to chromosome segmentation and seismic fault identification but also stand as versatile mathematical formulations with potential applications across various fields. Beyond orientation representation, they could be applied to other deep learning tasks, such as pose estimation. These representations are fundamental building blocks, not limited to any specific dataset, task, domain, or even sub-field of deep learning.

The first application explored in this thesis is chromosome karyotyping, with a focus on segmenting overlapping chromosomes. By predicting the orientation of chromosomes alongside semantic segmentation, the developed algorithms can accurately determine which segments of overlapping chromosomes belong to which individual chromosome instance. This work also identified issues with existing published methods and datasets in the field, where certain combinations led to artificially inflated metrics, allowing these issues to go unnoticed. I improved the published dataset, re-evaluated the problematic methods, and tested proposed improvements, which ultimately proved insufficient. As an alternative, I propose the use of orientation-

based instance segmentation. This application also served as a test bed for developing and refining orientation representations in 2D, laying the groundwork for a similar approach in 3D.

The second application focuses on the identification of faults in 3D seismic volumes. Building on semantic segmentation, my models can pinpoint the locations of faults within seismic images, predict their orientations, and separate them into distinct, non-intersecting fault segments. This process is highly scalable, making it suitable for large seismic surveys and efficient GPU processing. The resulting fault segments enable selective visualization in 3D, significantly improving the usability and inspection of the model's interpretations. The synthetic seismic images also provided a platform for evaluating the effectiveness of the 3D orientation representation, with the trained models subsequently deployed on a real seismic survey. The findings from this survey are compared with existing publications on the same geological site, offering new insights and a qualitative assessment of the model's ability to generalize from synthetic data to real seismic surveys.

7.1 Seismic Interpretation

Existing deep learning approaches for fault interpretation in seismic images primarily focus on detecting fault locations, which also serves as the foundation for my approach. The selection and optimization of my deep learning model were aimed at maximizing fault prediction performance and served as the benchmark for evaluating domain adaptation techniques. However, my research extends beyond fault location prediction by introducing a novel method: predicting fault orientation directly through the neural network. This orientation prediction can further refine the model's fault location predictions, offering a new dimension of accuracy and insight in seismic interpretation.

The novelty of my research is a significant strength, yet it also presents challenges. The absence of competing approaches means there are no direct comparisons to be made with other methods for predicting fault orientation. Additionally, the lack of standardized datasets in this field leads to comparisons between existing methods

that are often qualitative rather than quantitative. This challenge applies to both automated interpretation methods and those performed by geoscientists.

The 3D orientation representations developed in this research were tested in deployment scenarios specifically on seismic images. While this focused testing raises questions about the generalizability of these representations to other data domains, I argue that the extensive theoretical exploration and the experiments conducted on the toy dataset provide a strong justification for their broader applicability. Although slight variations in performance were observed among the four proposed representations—such as differences in their effectiveness on the toy dataset versus the synthetic seismic dataset—these variations should not detract from their overall qualitative utility.

7.2 Chromosome Karyotyping

A significant portion of my research was dedicated to exploring 2D orientation representations and the principles required to make them effective. I also focused on developing representations that can be scaled to 3D. Despite the limitations of the dataset on which the models were trained—such as its small size—the findings regarding the effectiveness of these representations remain robust. These findings validate the theoretical challenges associated with various representations, as outlined in this thesis, and demonstrate their practical effectiveness.

My critique of existing techniques and the dataset used in published studies constitutes a relevant scientific contribution. This critique identifies specific issues in the methods employed by previous works, highlighting gaps in the field that my research addresses.

The orientation-based approach for separating overlapping chromosomes stands as a valuable contribution in its own right. It successfully outperformed the criticized semantic segmentation method and, despite the limited size of the available dataset, demonstrated superior performance over existing alternatives. This method not only provides a practical solution for the immediate problem but also lays the groundwork for future research that could refine the approach and potentially lead to its adoption

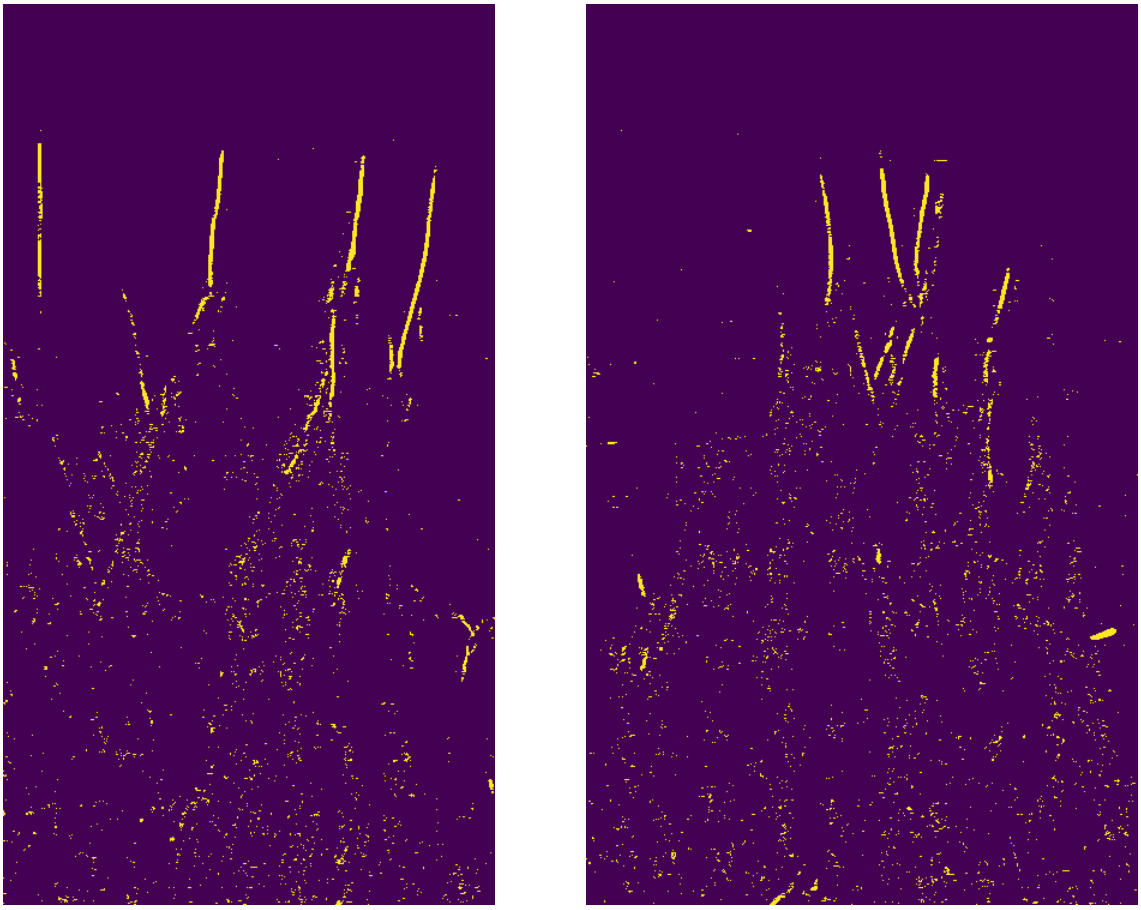
in industry.

7.3 Domain Adaptation

In this thesis, many models were trained on synthetic data, which were then deployed on real data. This raises the question of whether other domain adaptation approaches could improve the model's ability to generalize to real datasets. I explored several approaches to mitigate this issue in the seismic imaging application, but mostly encountered negative results. It appears that the limitations of the synthetic seismic data and of the convolutional architectures counterbalance each other, leading to exceptionally good transfer performance to the real seismic survey. In this section, I will outline the approaches I explored and my findings.

The approach I explored in the most detail is masking self-supervised learning. Despite the lack of annotated real seismic data, unlabelled data is plentiful. We can, therefore, train the model in a self-supervised manner on this data. I trained the model to recreate the input data in its outputs and to fill in masked areas of the input data. The model was then trained on the synthetic data to perform both fault location and orientation prediction, while also recreating the masked areas of the inputs. During training, it was simultaneously trained to recreate the inputs on the real data, while its predictions on fault locations remained unaffected by the real data. This approach improved the model's ability to predict fault locations by 2.5% on the synthetic data. However, the model almost completely lost its ability to predict faults on the real data, with only some of the most obvious and clean faults in the shallow parts of the volume being detected.

I hypothesize that the differences between the synthetic and real data were exacerbated in the model. Since the synthetic data does not fully resemble the real data, the model may have learned to treat the self-supervised learning task on the real data differently from the synthetic data, recognizing that they are different. This could have led to the model's ability to predict fault locations and orientations being limited to data that more closely resembles the synthetic data, such as the shallow regions, and hindered its ability to detect faults in deeper and more chaotic



(a) (b)

Figure 7.1: Slices (a) and (b) are the same from Figure 6.6, visualising the prediction of fault locations from the self-supervised model that was fine-tuned on the annotations from Figure 6.6. As such, they serve as training-set performance visualisations.

regions. Moreover, the self-supervised learning forces the model to deduce information from the surroundings of the masked region, which means it is compelled to observe broader fault geometry—one of the aspects that differs significantly between the synthetic and real data.

Next, I attempted to fine-tune the self-supervised model. The two slices from Figure 6.6 contain high-quality annotations, which I used to further train the model. Unfortunately, this did not lead to substantial improvements, implying that the problems with self-supervised learning are not easily overcome in this model. Figure 7.1 shows the resulting predictions on the slices used for fine-tuning. More fine-tuning with additional data could still lead to larger improvements.

The next approach I considered was the use of GANs, specifically starting with

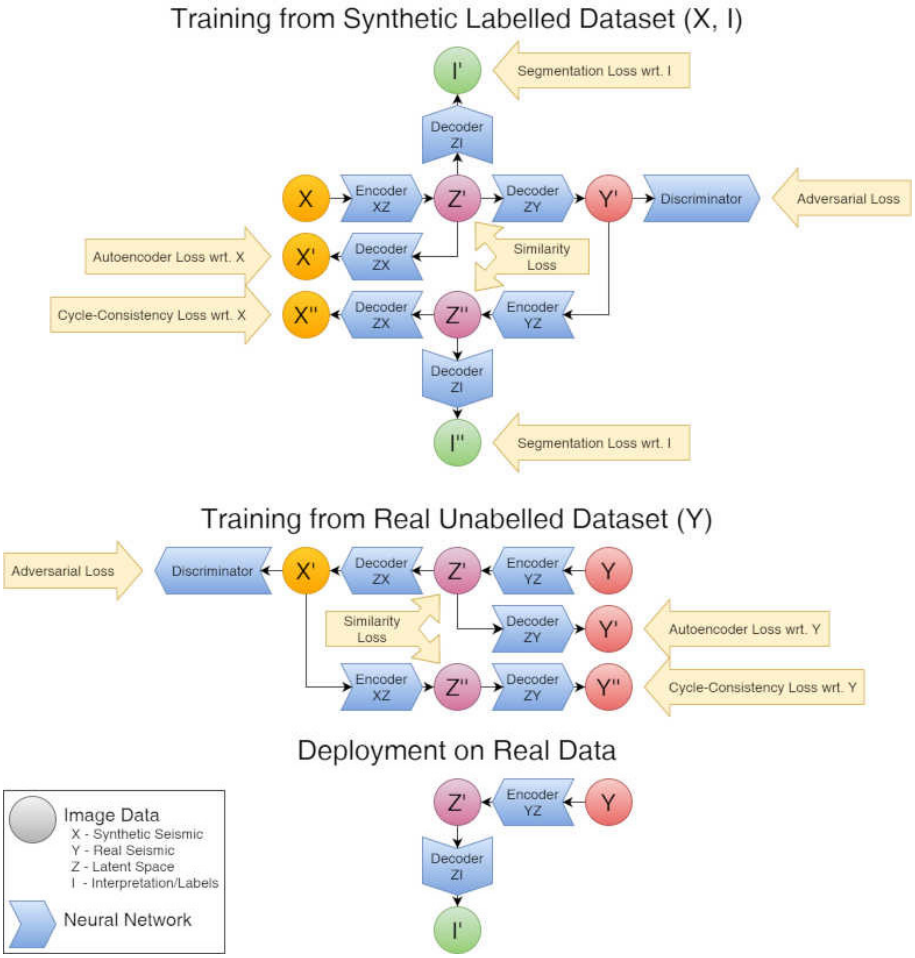


Figure 7.2: My proposed GAN architecture for training a model with labelled synthetic data and unlabelled real data.

a Cycle-GAN that could learn the domain shift between the synthetic and real data. My proposed architecture, visualized in Figure 7.2, intercepts the process of shifting from one domain to the next and feeds this in-between space into the model for identifying faults. Although potentially promising, the architecture suffers from a fundamental issue: it does not guarantee that the location of the faults is maintained when transferring from one domain to the other. Due to this limitation, I did not implement or experiment with the architecture.

Ultimately, the naive approach of training on synthetic data and deploying on real data is exceptionally appropriate for its purpose, especially given the nature of these particular datasets and the architectures used. Although there is still potential to exploit the available unlabelled data from real seismic surveys as part of model training, a significant breakthrough in future work would likely be required to achieve

substantial progress in this area.

7.4 Future Work

Most of my work is wrapped up well, especially from the point of view of what it can offer to the theme of this thesis, namely seismic interpretation. Although there are multiple areas that would benefit from further work, they would not have a major impact on the conclusions drawn from this thesis, since they would either benefit fields other than seismic interpretation, or merely offer incremental improvements. Domain Adaptation remains as the primary area with potential for a breakthrough that would benefit seismic interpretation.

The models that I trained for chromosome karyotyping are capable of separating overlapping chromosomes. For true utility to karyotyping in industry, the model would have to be integrated into a workflow that works with whole images and multiple chromosomes. Although my proposed method is capable of working on the whole images, the neural networks powering the method are only trained on the synthetic data cropped around pairs of chromosomes. The most beneficial future work of this method would be improving the quality of this model and making it robust to the types of features and noise present in whole images. This would mostly require the collection of data for the task.

My work on predicting the orientation of 3D vectors is complete. The representations are explored and tested, both theoretically and in practice. A potential direction is predicting other features for seismic interpretation that may benefit from the representations. Similarly, some features may not benefit from negation invariant representations, but may need specialised representations developed. However, this is conditioned on having annotated features within the training data, which is something I don't have in my synthetic dataset.

The instance separation of faults could benefit from future work. It is difficult to quantitatively evaluate the effectiveness of the method due to the lack of ground truth annotations of real seismic surveys. Future work focusing on gathering datasets for evaluation could benefit in the refinement of the algorithms and

their hyperparameters. Nevertheless, the approach stands as useful in its current state, which is demonstrated by the qualitative evaluation on small examples from the real seismic dataset.

Similarly to the instance separation, improving the architecture of the models that perform the seismic interpretation would not change the conclusions of this thesis. Deep Learning is a field that moves very quickly, as well as the available computational resources. A slightly better model is always around the corner, and it is important to keep up with the field for any future research. Therefore, I don't think the pursuit of the best available models is justified at this time, beyond what I have performed in this thesis.

7.5 Conclusion

In conclusion, this thesis provided a thorough exploration of orientation representation and how neural networks can predict it, for both 2D and 3D orientation. I proposed novel representations and demonstrated their performance when predicting the orientation of chromosomes in 2D and of seismic faults in 3D. Subsequently, I demonstrated the utility of predicting orientation by using it for separating overlapping chromosomes and separating the predicted faults into separate fault segments that do not intersect. My novel orientation representations can be used in a broad set of fields that use Deep Learning and could benefit from predicting negation invariant vectors (orientation). My trained models for seismic interpretation and my method for separating faults into fault segments directly brought novel functionality and utility to the field of seismic interpretation. My work on chromosome segmentation uncovered issues in published approaches and a dataset, while proposing an alternative method to segment overlapping chromosomes.

Bibliography

[1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015. (document), 2.1, 2.1.1, 2.4, 2

[2] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016. (document), 2.1.1, 2.4, 2, 5.2.3

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. (document), 2.4, 2.1.1

[4] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018. (document), 2.4, 2.1.1

[5] D. Kluvanec, K. J. W. McCaffrey, T. B. Phillips, and N. Al Moubayed, "Negation invariant representations of 3-d vectors for deep learning models applied to fault geometry mapping in 3-d seismic reflection data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–16, 2023. (document), 1, 3.5, 3.6, 5.1.2, 5.2.3, 6.2.2

[6] T. B. Phillips, K. McCaffrey, and L. Magarinos, "Influence of variable decoupling between vertically separated fault populations on structural inheritance–the laminaria high, nw shelf of australia," *Basin Research*, vol. 34, no. 1, pp. 440–456, 2022. (document), 3.2.4, 6.2.1, 6.5, 6.6, 6.2.2

[7] D. Kluvanec, T. B. Phillips, K. J. W. McCaffrey, and N. Al Moubayed, "Using orientation to distinguish overlapping chromosomes," in *Artificial Neural Networks and Machine Learning – ICANN 2022* (E. Pimenidis, P. Angelov, C. Jayne, A. Papaleonidas, and M. Aydin, eds.), (Cham), pp. 391–403, Springer International Publishing, 2022. 1, 4.2.2

[8] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelli-*

gence and statistics, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011. 2.1

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 2.1

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. 2.1, 2.1.5

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. 2.1

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010. 2.1.1

[13] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, PMLR, 2016. 2.1.2

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 2.1.4, 2.1.5

[15] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018. 2.1.5

[16] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert, "Attention u-net: Learning where to look for the pancreas," 2018. 2.1.5

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. 2.1.5

[18] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," 2021. 2.1.5

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016. 2.1.6

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018. 2.1.6

[21] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," 2018. 2.1.6

[22] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," 2019. 2.1.6

[23] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li, "Solo: A simple framework for instance segmentation," 2021. 2.1.6

[24] R. Frodeman, "Geological reasoning: Geology as an interpretive and historical science," *GSA Bulletin*, vol. 107, pp. 960–968, 08 1995. 2.2.1

[25] W. Walker, P. Harremoës, J. Rotmans, J. Sluijs, M. Asselt, P. Janssen, and M. Kraus, "Defining uncertainty: A conceptual basis for uncertainty management in model-based decision support," *Integrated Assessment*, vol. 4, 03 2003. 2.2.1

[26] C. E. Bond, "Uncertainty in structural interpretation: Lessons to be learnt," *Journal of Structural Geology*, vol. 74, pp. 185 – 200, 2015. 2.2.1

[27] C. Bond, A. Gibbs, Z. Shipton, and S. Jones, "What do you think this is? "conceptual uncertainty" in geoscience interpretation," *GSA Today*, vol. 17, 11 2007. 2.2.1

[28] C. Bond, R. Lunn, Z. Shipton, and A. Lunn, "What makes an expert effective at interpreting seismic images?," *Geology*, vol. 40, pp. 75–78, 01 2012. 2.2.1

[29] E. C. Rankey and J. C. Mitchell, "That's why it's called interpretation : Impact of horizon uncertainty on seismic attribute analysis," *The Leading Edge*, vol. 22, pp. 820–828, 09 2003. 2.2.1

[30] D. Polson and A. Curtis, "Dynamics of uncertainty in geological interpretation," *Journal of The Geological Society - J GEOL SOC*, vol. 167, pp. 5–10, 01 2010. 2.2.1

[31] A. Curtis, "The science of subjectivity," *Geology*, vol. 40, pp. 95–96, 01 2012. 2.2.1

[32] P. Rowbotham, P. Kane, and M. Bentley, "Bias in geophysical interpretation–the case for multiple deterministic scenarios," *The Leading Edge*, vol. 29, no. 5, pp. 590–595, 2010. 2.2.1

[33] Z. Wang, H. Di, A. Shafiq, Y. Alaudah, and G. Alregib, "Successful leveraging of image processing and machine learning in seismic structural interpretation: A review," *The Leading Edge*, vol. 37, pp. 451–461, 06 2018. 2.2.2

[34] S. Chopra and K. J. Marfurt, "Seismic attributes — a historical perspective," *GEOPHYSICS*, vol. 70, no. 5, pp. 3SO–28SO, 2005. 2.2.2

[35] M. S. Bahorich and S. L. Farmer, *3-D seismic discontinuity for faults and stratigraphic features: The coherence cube*, pp. 93–96. 2005. 2.2.2

[36] A. Roberts, "Curvature attributes and their application to 3d interpreted horizons," *First Break*, vol. 19, no. 2, pp. 85–100, 2001. 2.2.2

[37] K. M. Tingdahl and M. De Rooij, "Semi-automatic detection of faults in 3d seismic data," *Geophysical Prospecting*, vol. 53, no. 4, pp. 533–542, 2005. 2.2.2

[38] I. Cohen, N. Coult, and A. A. Vassiliou, "Detection and extraction of fault surfaces in 3d seismic data," *GEOPHYSICS*, vol. 71, no. 4, pp. P21–P27, 2006. 2.2.2

[39] H. Di and D. Gao, "3d seismic flexure analysis for subsurface fault detection and fracture characterization," *Pure and Applied Geophysics*, vol. 174, no. 3, pp. 747–761, 2017. 2.2.2

[40] S. I. Pedersen, T. Randen, L. Sønneland, and Ø. Steen, "Automatic fault extraction using artificial ants," in *2002 SEG Annual Meeting*, OnePetro, 2002. 2.2.2

[41] S. Purves, J. Lowell, D. Norton, J. Henderson, G. Paton, and N. McArdle, "Us 10,429,529 b2 - adaptive fault tracking," Oct 2019. 2.2.2

[42] T. Pavlidis, "A thinning algorithm for discrete binary images," *Computer graphics and image processing*, vol. 13, no. 2, pp. 142–157, 1980. 2.2.2

[43] D. Hale, "Methods to compute fault images, extract fault surfaces, and estimate fault throws from 3d seismic images," *Geophysics*, vol. 78, no. 2, pp. O33–O43, 2013. 2.2.2

[44] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988. 2.2.2

[45] Y. An, H. Du, S. Ma, Y. Niu, D. Liu, J. Wang, Y. Du, C. Childs, J. Walsh, and R. Dong, "Current state and future directions for deep learning based automatic seismic fault interpretation: A systematic review," *Earth-Science Reviews*, vol. 243, p. 104509, 2023. 2.2.2

[46] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015. 2.2.2

[47] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989. 2.2.2

[48] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017. PMID: 28599112. 2.2.2

[49] Y. Zhang, Y. Liu, H. Zhang, and H. Xue, "Seismic facies analysis based on deep learning," *IEEE Geoscience and Remote Sensing Letters*, pp. 1–5, 2019. 2.2.2

[50] J. S. Dramsch and M. Lüthje, *Deep-learning seismic facies on state-of-the-art CNN architectures*, pp. 2036–2040. 2018. 2.2.2

[51] D. Salles Chevitarese, D. Szwarcman, R. Silva, and E. Vital Brazil, "Seismic facies segmentation using deep learning," 05 2018. 2.2.2

[52] Z. Wang, F. Li, T. R. Taha, and H. R. Arabnia, "Improved automating seismic facies analysis using deep dilated attention autoencoders," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 2.2.2

[53] T. Wrona, I. Pan, R. L. Gawthorpe, and H. Fossen, "Seismic facies analysis using machine learning," *GEOPHYSICS*, vol. 83, no. 5, pp. O83–O95, 2018. 2.2.2

[54] M. Alfarhan, A. Maalej, and M. Deriche, "Concurrent detection of salt domes and faults using resnet with u-net," in *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, pp. 118–122, 2020. 2.2.2

[55] A. U. Waldeland, A. C. Jensen, L.-J. Gelius, and A. H. S. Solberg, "Convolutional neural networks for automated seismic interpretation," *The Leading Edge*, vol. 37, no. 7, pp. 529–537, 2018. 2.2.2

[56] Y. Alaudah, S. Gao, and G. AlRegib, *Learning to label seismic structures with deconvolution networks and weak labels*, pp. 2121–2125. 2018. 2.2.2

[57] H. Di, Z. Wang, and G. Alregib, "Deep convolutional neural networks for seismic salt-body delineation," in *AAPG ACE 2018*. 2.2.2

[58] S. Li, B. Liu, Y. Ren, Y. Chen, S. Yang, Y. Wang, and P. Jiang, "Deep-learning inversion of seismic data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 3, pp. 2135–2149, 2020. 2.2.2

[59] W. Lewis and D. Vigh, *Deep learning prior models from seismic images for full-waveform inversion*, pp. 1512–1517. 2017. 2.2.2

[60] L. Huang, M. Polanco, and T. E. Clee, "Initial experiments on improving seismic data inversion with deep learning," in *2018 New York Scientific Data Summit (NYSDS)*, pp. 1–3, 2018. 2.2.2

[61] F. Yang and J. Ma, "Deep-learning inversion: a next generation seismic velocity-model building method," *ArXiv*, vol. abs/1902.06267, 2019. 2.2.2

[62] V. Tschannen, M. Delescluse, N. Ettrich, and J. Keuper, "Extracting horizon surfaces from 3d seismic data using deep learning," *GEOPHYSICS*, vol. 85, no. 3, pp. N17–N26, 2020. 2.2.2

[63] L. Yang and S. Z. Sun, "Seismic horizon tracking using a deep convolutional neural network," *Journal of Petroleum Science and Engineering*, vol. 187, p. 106709, 2020. 2.2.2

[64] H. Wu and B. Zhang, "A deep convolutional encoder-decoder neural network in assisting seismic horizon tracking," 04 2018. 2.2.2

[65] J. Lowell and G. Paton, *Application of deep learning for seismic horizon interpretation*, pp. 1976–1980. 2018. 2.2.2

[66] X. Wu, Y. Shi, S. Fomel, L. Liang, Q. Zhang, and A. Z. Yusifov, "Faultnet3d: Predicting fault probabilities, strikes, and dips with a single convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 9138–9155, 2019. 2.2.2, 2.2.3

[67] X. Wu, L. Liang, Y. Shi, and S. Fomel, "Faultseg3d: Using synthetic data sets to train an end-to-end convolutional neural network for 3d seismic fault segmentation," *Geophysics*, vol. 84, p. IM35–IM45, 02 2019. 2.2.2

[68] X. Wu, L. Liang, Y. Shi, Z. Geng, and S. Fomel, *Deep learning for local seismic image processing: Fault detection, structure-oriented smoothing with edge-preserving, and slope estimation by using a single convolutional neural network*, pp. 2222–2226. 2019. 2.2.2

[69] A. Cunha, A. Pochet, H. Lopes, and M. Gattass, "Seismic fault detection in real data using transfer learning from a convolutional neural network pre-trained with synthetic seismic data," *Computers & Geosciences*, vol. 135, p. 104344, 11 2019. 2.2.2, 2.2.3

[70] W. Xiong, X. Ji, Y. Ma, Y. Wang, N. M. AlBinHassan, M. N. Ali, and Y. Luo, "Seismic fault detection with convolutional neural network," *GEOPHYSICS*, vol. 83, no. 5, pp. O97–O103, 2018. 2.2.2

[71] T. Zhao and P. Mukhopadhyay, *A fault-detection workflow using deep learning and image processing*, pp. 1966–1970. 2018. 2.2.2

[72] P. Lu, M. Morris, S. Brazell, C. Comiskey, and Y. Xiao, "Using generative adversarial networks to improve deep-learning fault interpretation networks," *The Leading Edge*, vol. 37, pp. 578–583, 08 2018. 2.2.2

[73] M. Araya-Polo, T. Dahlke, C. Frogner, C. Zhang, T. Poggio, and D. Hohl, "Automated fault detection without seismic processing," *The Leading Edge*, vol. 36, no. 3, pp. 208–214, 2017. 2.2.2

[74] T. Dahlke, M. Araya-Polo, C. Zhang, C. Frogner, and T. Poggio, "Predicting geological features in 3d seismic data," *Advances in Neural Information Processing Systems (NIPS)*, vol. 29, 2016. 2.2.2

[75] A. Pochet, P. H. Diniz, H. Lopes, and M. Gattass, "Seismic fault detection using convolutional neural networks trained on synthetic poststacked amplitude maps," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 3, pp. 352–356, 2018. 2.2.2, 2.2.3

[76] Y. Ma, X. Ji, N. M. BenHassan, Y. Luo, *et al.*, "A deep learning method for automatic fault detection," in *2018 SEG International Exposition and Annual Meeting*, Society of Exploration Geophysicists, 2018. 2.2.2

[77] L. Huang, X. Dong, and T. E. Clee, "A scalable deep learning platform for identifying geologic features from seismic attributes," *The Leading Edge*, vol. 36, no. 3, pp. 249–256, 2017. 2.2.2

[78] J. Heaton, "Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618," *Genetic programming and evolvable machines*, vol. 19, no. 1, pp. 305–307, 2018. 2.2.3

[79] X. Wu, Y. Shi, S. Fomel, and L. Liang, "Convolutional neural networks for fault interpretation in seismic images," 08 2018. 2.2.3

[80] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135 – 153, 2018. 2.2.3

[81] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," in *Advances in neural information processing systems*, pp. 136–144, 2016. 2.2.3

[82] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *European conference on computer vision*, pp. 443–450, Springer, 2016. 2.2.3

[83] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He, "Supervised representation learning: Transfer learning with deep autoencoders," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. 2.2.3

[84] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, no. 1-2, pp. 151–175, 2010. 2.2.3

[85] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, "Revisiting batch normalization for practical domain adaptation," *arXiv preprint arXiv:1603.04779*, 2016. 2.2.3

[86] A. Rozantsev, M. Salzmann, and P. Fua, "Beyond sharing weights for deep domain adaptation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 4, pp. 801–814, 2018. 2.2.3

[87] T. Xiao, H. Li, W. Ouyang, and X. Wang, "Learning deep feature representations with domain guided dropout for person re-identification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1249–1258, 2016. 2.2.3

[88] S.-W. Huang, C.-T. Lin, S.-P. Chen, Y.-Y. Wu, P.-H. Hsu, and S.-H. Lai, "Auggan: Cross domain adaptation with gan-based data augmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 718–731, 2018. 2.2.3

[89] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017. 2.2.3

[90] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4068–4076, 2015. 2.2.3

[91] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning*, pp. 1180–1189, 2015. 2.2.3

[92] Z. Yi, H. Zhang, P. Tan, and M. Gong, "Dualgan: Unsupervised dual learning for image-to-image translation," in *Proceedings of the IEEE international conference on computer vision*, pp. 2849–2857, 2017. 2.2.3

[93] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017. 2.2.3

[94] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," *arXiv preprint arXiv:1703.05192*, 2017. 2.2.3

[95] J. Hoffman, D. Wang, F. Yu, and T. Darrell, "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation," *arXiv preprint arXiv:1612.02649*, 2016. 2.2.3

[96] Y. Chen, W. Li, and L. Van Gool, "Road: Reality oriented adaptation for semantic segmentation of urban scenes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2.2.3

[97] S. Sankaranarayanan, Y. Balaji, A. Jain, S. Nam Lim, and R. Chellappa, "Learning from synthetic data: Addressing domain shift for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3752–3761, 2018. 2.2.3

[98] J. Choi, T. Kim, and C. Kim, "Self-ensembling with gan-based data augmentation for domain adaptation in semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, pp. 6830–6840, 2019. 2.2.3

[99] P. M. Lansdorp, N. P. Verwoerd, F. M. Van De Rijke, V. Dragowska, M.-T. Little, R. W. Dirks, A. K. Raap, and H. J. Tanke, "Heterogeneity in telomere length of human chromosomes," *Human molecular genetics*, vol. 5, no. 5, pp. 685–691, 1996. 3.1.2

[100] X. Wu and D. Hale, "3d seismic image processing for faults," *Geophysics*, vol. 81, 2016. 3.2.3

[101] X. Wu, Z. Geng, Y. Shi, N. Pham, S. Fomel, and G. Caumon, "Building realistic structure models to train convolutional neural networks for seismic structural interpretation," *Geophysics*, vol. 85, 2020. 3.2.3

[102] Australian Geological Survey Organisation (AGSO) North West Shelf Study Group, "Deep reflections on the north west shelf: changing perspectives of basin formation," 1994. 3.2.4

[103] Y. An, J. Guo, Q. Ye, C. Childs, J. Walsh, and R. Dong, "A gigabyte interpreted seismic dataset for automatic fault recognition," *Data in Brief*, vol. 37, 2021. 3.2.4

[104] Y. An, J. Guo, Q. Ye, C. Childs, J. Walsh, and R. Dong, "Deep convolutional neural network for automatic fault recognition from 3d seismic datasets," *Computers & Geosciences*, vol. 153, p. 104776, 2021. 3.2.4

[105] X. Wu, L. Liang, Y. Shi, and S. Fomel, "Faultseg3d: Using synthetic data sets to train an end-to-end convolutional neural network for 3d seismic fault segmentation," *GEOPHYSICS*, 2019. 3.2.4

[106] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 4.2.1, 5.2.3

[107] I. 80000-2:2019, "Quantities and units — part 2: Mathematics," tech. rep., International Organization for Standardization, 2019. 5.1.1

[108] X. Wu, Y. Shi, S. Fomel, L. Liang, Q. Zhang, and A. Yusifov, "Faultnet3d: Predicting fault probabilities, strikes, and dips with a single convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, pp. 9138–9155, 2019. 5.1.4

[109] A. Saxena, J. Driemeyer, and A. Y. Ng, "Learning 3-d object orientation from images," in *2009 IEEE International conference on robotics and automation*, pp. 794–800, IEEE, 2009. 5.1.5

[110] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011. 5.1.5

[111] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 5.2.3

[112] R. M. Gardner and D. J. Amor, *Chromosome abnormalities and genetic counseling*. Oxford University Press, 2018. 6.1

[113] C.-H. Pui, W. M. Crist, and A. T. Look, "Biology and clinical significance of cytogenetic abnormalities in childhood acute lymphoblastic leukemia.," *Blood*, vol. 76 8, pp. 1449–63, 1990. 6.1

[114] T. Arora and R. Dhir, "A review of metaphase chromosome image selection techniques for automatic karyotype generation," *Medical & biological engineering & computing*, vol. 54, no. 8, pp. 1147–1157, 2016. 6.1

[115] X. Shen, Y. Qi, T. Ma, and Z. Zhou, "A dicentric chromosome identification method based on clustering and watershed algorithm," *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019. 6.1

[116] H. Cao, H.-W. Deng, and Y.-P. Wang, "Segmentation of m-fish images for improved classification of chromosomes with an adaptive fuzzy c-means clustering algorithm," *IEEE Transactions on fuzzy systems*, vol. 20, no. 1, pp. 1–8, 2011. 6.1

[117] L. Ji, "Fully automatic chromosome segmentation," *Cytometry: The Journal of the International Society for Analytical Cytology*, vol. 17, no. 3, pp. 196–208, 1994. 6.1

[118] Y. Li, J. H. Knoll, R. C. Wilkins, F. N. Flegal, and P. K. Rogan, "Automated discrimination of dicentric and monocentric chromosomes by machine learning-based image processing," *Microscopy research and technique*, vol. 79, no. 5, pp. 393–402, 2016. 6.1

[119] R. M. Nair, R. Remya, and K. Sabeena, "Karyotyping techniques of chromosomes: a survey," *International Journal of Computer Trends and Technology*, vol. 22, no. 1, pp. 30–34, 2015. 6.1

[120] M. Popescu, P. Gader, J. Keller, C. Klein, J. Stanley, and C. Caldwell, "Automatic karyotyping of metaphase cells with overlapping chromosomes," *Computers in biology and medicine*, vol. 29, no. 1, pp. 61–82, 1999. 6.1

[121] D. Somasundaram and V. V. Kumar, "Separation of overlapped chromosomes and pairing of similar chromosomes for karyotyping analysis," *Measurement*, vol. 48, pp. 274–281, 2014. 6.1

[122] R. L. Hu, J. Karnowski, R. Fadely, and J.-P. Pommier, "Image segmentation to distinguish between overlapping human chromosomes," *arXiv preprint arXiv:1712.07639*, 2017. 6.1, 6.1.1, 6.1.1

[123] H. M. Saleh, N. H. Saad, and N. A. M. Isa, "Overlapping chromosome segmentation using u-net: Convolutional networks with test time augmentation," *Procedia Computer Science*, vol. 159, pp. 524–533, 2019. 6.1, 6.1.1, 6.1.1

[124] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. 6.1.1

[125] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 fourth international conference on 3D vision (3DV)*, pp. 565–571, Ieee, 2016. 6.2

[126] N. B. Çiftçi and L. Langhi, "Evolution of the hourglass structures in the laminaria high, timor sea: Implications for hydrocarbon traps," *Journal of Structural Geology*, vol. 36, pp. 55–70, 2012. 6.2.1, 6.2.2

APPENDIX A

Basic and Auxiliary Results

A.1 Orientation Representation

A.1.1 3D Network Optimisation

	model: widths × depths	norm	combine	other	fault IOU	intersection IOU	max angle	mean angle
1	HarDAddUNet 60×16	batch			52%	9%	82°	12.9°
2	HarDUNet (k=16)×16	batch			54%	7%	82°	12.2°
3	ResUNet 80×16	batch			56%	11%	80°	11.4°
4	UNet 92×16	batch			54%	9%	81°	11.6°
5	HarDAddUNet 30×32	batch			54%	6%	82°	11.9°
6	HarDUNet (k=8)×32	batch			51%	5%	83°	16.2°
7	ResUNet 42×32	batch			58%	12%	83°	13.8°
8	UNet 46×32	batch			55%	9%	82°	11.9°
9	HarDAddUNet 16×64	batch			51%	0%	84°	29.2°
10	HarDUNet (k=3)×64	batch			54%	7%	83°	13.2°
11	ResUNet 22×64	batch			59%	13%	84°	18.6°
12	UNet 23×64	batch			50%	5%	84°	13.6°
13	HarDAddUNet 32×8 ↓ 44×64 ↑ 32×8	batch	add		61%	16%	81°	12.6°
14	HarDAddUNet 32×8 ↓ 88×32 ↑ 32×8	batch	add		61%	16%	81°	12.2°
15	HarDAddUNet 32×8 ↓ 64×8 ↓ 160×64 ↑ 64×8 ↑ 32×8	batch	add		63%	17%	82°	11.8°
16	HarDAddUNet 32×8 ↓ 64×16 ↓ 128×32 ↑ 64×16 ↑ 32×8	batch	add		59%	14%	82°	13.5°
17	HarDAddUNet 64×4 ↓ 128×4 ↓ 128×32 ↑ 128×4 ↑ 64×4	batch	add		61%	14%	80°	10.2°
18	HarDAddUNet 96×2 ↓ 192×2 ↓ 192×32 ↑ 192×2 ↑ 96×2	batch	add		58%	16%	81°	10.7°
19	HarDAddUNet 32×8 ↓ 64×8 ↓ 160×64 ↑ 64×8 ↑ 32×8	batch	add		61%	16%	82°	12.2°
20	HarDUNet 100(k=11)×8 ↓ 160(k=16)×8 ↓ (k=24)×64 ↑ 160(k=16)×8 ↑ 100(k=11)×8	batch	add		61%	17%	76°	9.0°

Table A.1: Is extended in Tables A.2 and A.3: All of the neural network architectures trained on the synthetic seismic dataset and tested on its validation set. The second line of each row in the table shows the width×depth of the blocks used in the UNet architecture, with ↑ and ↓ representing the downsampling/upsampling in the UNet architecture. Refer to Figure 5.13 for architecture details.

	model: widths × depths	norm	combine	other	fault IOU	intersection IOU	max angle	mean angle
21	ResUNet 40×8 ↓ 80×8 ↓ 224×64 ↑ 80×8 ↑ 40×8	batch	add		63%	18%	80°	10.1°
22	UNet 52×8 ↓ 104×8 ↓ 240×64 ↑ 104×8 ↑ 52×8	batch	add		Failed on float16			
23	HarDUNet 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16×8 ↑ 72(k=11)×8	batch	cat		Failed on float16			
24	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 288×64 ↑ 128×8 ↑ 64×8 ↑ 32×8	batch	add		Failed on float16			
25	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 256×8 ↓ 320×64 ↑ 256×8 ↑ 128×8 ↑ 64×8 ↑ 32×8	batch	add		Failed on float16			
26	HarDAddUNet 32×8 ↓ 64×8 ↓ 160×64 ↑ 64×8 ↑ 32×8	layer	add		Out of memory			
27	HarDAddUNet 32×8 ↓ 64×8 ↓ 160×64 ↑ 64×8 ↑ 32×8	instance	add		64%	16%	81°	11.9°
28	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 288×64 ↑ 128×8 ↑ 64×8 ↑ 32×8	layer	add		Out of memory			
29	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 288×64 ↑ 128×8 ↑ 64×8 ↑ 32×8	instance	add		62%	17%	83°	12.4°
30	HarDUNet 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16×8 ↑ 72(k=11)×8	instance	cat		64%	18%	76°	9.0°
31	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 256×8 ↓ 320×64 ↑ 256×8 ↑ 128×8 ↑ 64×8 ↑ 32×8	instance	add		63%	16%	82°	13.1°
32	HarDUNet 48(k=6)×8 ↓ 80(k=10)×8 ↓ (k=22)×64 ↑ 80(k=10)×8 ↑ 48(k=6)×8	layer	cat		61%	15%	79°	9.4°
33	HarDUNet 80(k=10)×8 ↓ 112(k=14)×8 ↓ 144(k=18)×8 ↓ (k=22)×64 ↑ 144(k=18)×8 ↑ 112(k=14)×8 ↑ 80(k=10)×8	instance	cat		64%	17%	80°	9.7°
34	HarDAddUNet 32×8 ↓ 64×8 ↓ 128×8 ↓ 240×8 ↓ 288×64 ↑ 240×8 ↑ 128×8 ↑ 64×8 ↑ 32×8	instance	cat		63%	17%	82°	13°
35	HarDAddUNet 32×8 ↓ 64×8 ↓ 96×8 ↓ 160×8 ↓ 224×8 ↓ 288×64 ↑ 224×8 ↑ 160×8 ↑ 96×8 ↑ 64×8 ↑ 32×8	instance	add		60%	18%	83°	15.3°
36	HarDUNet 96(k=10)×8 ↓ 128(14)×8 ↓ 160(18)×8 ↓ 196(22)×8 ↓ (26)×64 ↑ 196(22)×8 ↑ 160(18)×8 ↑ 128(14)×8 ↑ 96(10)×8	instance	cat		61%	16%	81°	11.6°
37	HarDUNet 80(k=10)×8 ↓ 112(k=14)×8 ↓ 144(k=18)×8 ↓ (k=22)×32 ↑ 144(k=18)×8 ↑ 112(k=14)×8 ↑ 80(k=10)×8	instance	cat		64%	18%	77°	9.3°
38	HarDUNet 80(k=10)×4 ↓ 112(k=14)×4 ↓ 144(k=18)×4 ↓ (k=22)×64 ↑ 144(k=18)×8 ↑ 112(k=14)×8 ↑ 80(k=10)×8	instance	cat		58%	15%	81°	11.5°
39	HarDUNet 80(k=10)×8 ↓ 112(k=14)×8 ↓ 144(k=18)×8 ↓ (k=22)×64 ↑ 144(k=18)×4 ↑ 112(k=14)×4 ↑ 80(k=10)×4	instance	cat		61%	16%	78°	11.2°
40	ResUNet 40×8 ↓ 80×8 ↓ 224×64 ↑ 80×8 ↑ 40×8	instance	add		65%	17%	80°	10.9°
41	ResUNet 40×8 ↓ 80×8 ↓ 160×8 ↓ 320×64 ↑ 160×8 ↑ 80×8 ↑ 40×8	instance	add		62%	15%	82°	13.9°
42	HarDUNet 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8	instance	cat	lr=3e-3	64%	17%	80°	10.5°
43	HarDUNet 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8	instance	cat	lr=3e-4	63%	17%	79°	9.4°
44	HarDUNet 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8	instance	cat	lr=1e-4	62%	17%	81°	10.1°

Table A.2: Continuation of Table A.1. Is extended in Table A.3.

	model: widths × depths norm	combine	other	fault IOU	intersection IOU	max angle	mean angle
45	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8			63%	16%	81°	11.4°
46	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×32 ↑ 104(k=16)×8 ↑ 72(k=11)×8			64%	19%	74°	9.1°
47	HarDUNet instance cat 52(k=13)×8 ↓ 88(k=16)×8 ↓ (k=24)×64 ↑ 88(k=16)×8 ↑ 52(k=13)×8			63%	17%	79°	9.3°
48	HarDUNet instance cat 128(k=6)×8 ↓ 160(k=10)×8 ↓ (k=18)×64 ↑ 160(k=10)×8 ↑ 128(k=6)×8			63%	17%	79°	10.9°
49	HarDUNet instance cat 84(k=15)×8 ↓ 112(k=22)×8 ↓ (k=30)×64 ↑ 112(k=22)×8 ↑ 84(k=15)×8		m=1.2	63%	17%	80°	10.8°
50	HarDUNet instance cat 72(k=10)×8 ↓ 128(k=12)×16 ↓ (k=18)×64 ↑ 128(k=12)×16 ↑ 72(k=10)×8			61%	16%	81°	10.0°
51	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×32 ↑ 104(k=16)×8 ↑ 72(k=11)×8			61%	15%	80°	9.8°
52	ResUNet instance add 40×8 ↓ 80×8 ↓ 224×64 ↑ 80×8 ↑ 40×8			63%	16%	82°	12.4°
53	ResUNet instance add 46×8 ↓ 92×8 ↓ 138×64 ↑ 92×8 ↑ 46×8			62%	17%	81°	10.4°
54	ResUNet instance add 40×8 ↓ 80×16 ↓ 224×32 ↑ 80×16 ↑ 40×8			63%	17%	82°	10.3°
55	ResUNet instance cat 40×8 ↓ 80×8 ↓ 216×64 ↑ 80×8 ↑ 40×8			63%	17%	81°	10.9°
56	ResUNet instance add 40×8 ↓ 80×8 ↓ 224×64 ↑ 80×8 ↑ 40×8		bias	64%	18%	81°	9.96°
57	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=4.4e-4	63%	17%	80°	10.4°
58	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=6.7e-4	64%	18%	78°	8.6°
59	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=1e-3	65%	19%	76°	8.1°
60	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=1.5e-3	63%	19%	77°	9.1°
61	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=2.25e-3	64%	18%	76°	8.5°
62	HarDUNet instance cat 72(k=11)×8 ↓ 104(k=16)×8 ↓ (k=24)×64 ↑ 104(k=16)×8 ↑ 72(k=11)×8		lr=3.375e-3	64%	18%	78°	9.0°
63	HarDUNet instance cat 288(k=11)×8 ↓ 416(k=16)×8 ↓ (k=24)×64 ↑ 416(k=16)×8 ↑ 288(k=11)×8		lr=1e-3 batch=1	60%	12%	79°	10.0°
64	HarDUNet instance cat 288(k=11)×8 ↓ 416(k=16)×8 ↓ (k=24)×64 ↑ 416(k=16)×8 ↑ 288(k=11)×8		lr=3e-4 batch=1	60%	12%	79°	10.0°
64	HarDUNet instance cat 288(k=11)×8 ↓ 416(k=16)×8 ↓ (k=24)×64 ↑ 416(k=16)×8 ↑ 288(k=11)×8		lr=1e-4 batch=1	61%	11%	81°	9.7°
65	HarDUNet instance cat 240(k=32)×8 ↓ 336(k=44)×8 ↓ 432(k=56)×8 ↓ (k=68)×64 ↑ 432(k=56)×8 ↑ 336(k=44)×8 ↑ 240(k=32)×8		lr=1e-3 batch=1	59%	10%	80°	10.0°
66	HarDUNet instance cat 240(k=32)×8 ↓ 336(k=44)×8 ↓ 432(k=56)×8 ↓ (k=68)×64 ↑ 432(k=56)×8 ↑ 336(k=44)×8 ↑ 240(k=32)×8		lr=3e-4 batch=1	63%	14%	78°	9.1°
67	HarDUNet instance cat 240(k=32)×8 ↓ 336(k=44)×8 ↓ 432(k=56)×8 ↓ (k=68)×64 ↑ 432(k=56)×8 ↑ 336(k=44)×8 ↑ 240(k=32)×8		lr=1e-4 batch=1	62%	14%	80°	9.5°

Table A.3: Continuation of Table A.1 and A.2