Universidade Estadual de Campinas
Instituto de Computação

Gabriel B. Gutierrez

An Evaluation of Transformer Models for Seismic Facies Segmentation

CAMPINAS
2025

Gabriel B. Gutierrez

An Evaluation of Transformer Models for Seismic Facies Segmentation

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Edson Borin

Este exemplar corresponde à versão final da Dissertação defendida por Gabriel B. Gutierrez e orientada pelo Prof. Dr. Edson Borin.

CAMPINAS
2025

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Gutierrez, Gabriel Borges, 2000-
G985e An evaluation of transformer models for seismic facies segmentation / Gabriel Borges Gutierrez. – Campinas, SP : [s.n.], 2025.

Orientador: Edson Borin.
Dissertação (mestrado) – Universidade Estadual de Campinas (UNICAMP), Instituto de Computação.

1. Aprendizado de máquina. 2. Transformers (Arquitetura de computador). 3. Fácies sísmicas. 4. Segmentação semântica. I. Borin, Edson, 1979-. II. Universidade Estadual de Campinas (UNICAMP). Instituto de Computação. III. Título.

Informações complementares

Título em outro idioma: Uma avaliação de modelos transformers para segmentação de fácies sísmicas
Palavras-chave em inglês:
Machine learning
Transformers (Computer architecture)
Seismic facies
Semantic segmentation
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Edson Borin [Orientador]
Marcelo da Silva Reis
Alexandro José Baldassin
Data de defesa: 24-02-2025
Programa de Pós-Graduação: Ciência da Computação

Objetivos de Desenvolvimento Sustentável (ODS)
ODS: 9. Inovação e infraestrutura

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0008-4200-8963
- Currículo Lattes do autor: https://lattes.cnpq.br/6015137863078458

- Prof. Dr. Marcelo da Silva Reis
  Universidade Estadual de Campinas
- Prof. Dr. Alexandro José Baldassin
  Universidade Estadual Paulista

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Acknowledgements

I would like to express my gratitude to my adviser, Professor Edson Borin, for his invaluable guidance, advice, and constant support throughout my master's journey. His encouragement has given me the space to grow, not only as a student but also as a professional. I extend my thanks to the members of my thesis committee—Professor Marcelo da Silva Reis, Professor Edson Takashi Matsubara, Professor Esther Luna Colombini, and Professor Alexandro José Baldassin—for their constructive critiques and invaluable support.

I am deeply grateful to my colleagues at Discovery Lab for their collaboration, insights, and support. I also extend my thanks to our counterparts at Petrobras and NVIDIA for their continuous guidance and contributions.

This research would not have been possible without the generous support of Petrobras, CNPq, and FAPESP, which provided critical resources and funding. I am also thankful to SENAI/CIMATEC for granting access to supercomputing resources and to New Zealand Petroleum and Minerals (NZPM) for providing the PARIHAKA-3D public dataset.

I would like to express my deepest gratitude to my family and friends for their unwavering encouragement and understanding during the challenges of this process. Their support and companionship have been indispensable in helping me reach this milestone.

Finally, as the son of two professors, I am profoundly thankful to every teacher and professor I have had along the way. Their dedication and work have laid the foundation for all my accomplishments. I am eternally grateful for their impact on my life. To all my teachers and professors—thank you.

Resumo

A segmentação de fácies sísmicas é uma tarefa crucial na análise do subsolo, auxiliando geólogos na identificação de estruturas geológicas a partir de dados sísmicos. Redes Neurais Convolucionais (CNNs) têm sido amplamente utilizadas para essa tarefa, mas apresentam limitações intrínsecas na captura de dependências de longo alcance. Recentemente, os Transformers Visuais (ViTs) surgiram como uma alternativa promissora, aproveitando mecanismos de autoatenção para modelar relações espaciais de forma mais eficaz.

Neste trabalho, investigamos o uso de modelos de segmentação baseados em Transformers para a segmentação de fácies sísmicas. Especificamente, comparamos o desempenho de cinco modelos de segmentação, sendo dois baseados em CNNs e três baseados em Transformers. Todos os modelos são treinados e avaliados nos conjuntos de dados F3 e Seam AI sob regimes de treinamento idênticos, garantindo uma comparação justa. Além das métricas quantitativas de desempenho, conduzimos uma análise qualitativa para evidenciar diferenças sutis nas previsões dos modelos.

Também implementamos um pipeline de otimização de parâmetros para o modelo com melhor desempenho na comparação. No entanto, devido a restrições técnicas e de tempo, o modelo ajustado não alcançou os ganhos de desempenho esperados, apresentando apenas melhorias marginais. Uma configuração mais robusta é necessária para avaliar adequadamente o impacto do ajuste de hiperparâmetros.

Abstract

Seismic facies segmentation is a critical task in subsurface analysis, assisting geologists in identifying geological structures from seismic data. Convolutional Neural Networks (CNNs) have been widely used for this task, but they are inherently limited in capturing long-range dependencies. Recently, Vision Transformers (ViTs) have emerged as a powerful alternative, leveraging self-attention mechanisms to model spatial relationships more effectively.

In this work, we investigate the use of Transformer-based segmentation models for seismic facies segmentation. Specifically, we compare the performance of five segmentation models, two CNN-based and three Transformer-based. All models are trained and evaluated on the F3 and Seam AI datasets under identical training regimens to ensure a fair comparison. In addition to quantitative performance metrics, we conduct a qualitative analysis to highlight nuanced differences in the models' predictions.

We also implement a parameter optimization pipeline for the best-performing model from the comparison. However, due to technical and time constraints, the tuned model did not achieve the desired performance improvements, yielding only marginal gains. A more robust setup is necessary to properly evaluate the impact of hyperparameter tuning.

List of Figures

List of Tables

List of Abbreviations and Acronyms

- Atrous Spatial Pyramid Pooling (ASPP)
- Convolutional Neural Network (CNN)
- European Association of Geophysicists and Engineers (EAGE)
- Generative Adversarial Networks (GANs)
- Intersection Over Union (IoU)
- Mean Class Accuracy (MCA)
- Mean Intersection Over Union (mIoU)
- Multi-Layer Perceptron (MLP)
- Multi-Level Feature Aggregation (MLA)
- Natural Language Processing (NLP)
- Pixel Accuracy (PA)
- Progressive Upscale (PUP)
- Semantic Transformer (SETR)
- Vision Transformer (ViT)

Contents

Chapter 1

Introduction

Seismic Facies, described in more detail in Chapter 2, is a sedimentary unity with singular characteristics. They are usually imaged through seismic wave emission, ether on or off shore. The segmentation and classification of such seismic facies is a essential tasks for characterizing subsurface features and structures in a given area [18]. When performed by a human specialist, these tasks are highly time-consuming and demand significant expertise to ensure accuracy [4]. For this reason, machine learning models hold significant potential to both reduce the time required for these tasks and allow specialists to focus on more valuable activities.

Over the years, numerous studies have explored the application of semantic segmentation models to seismic facies, with a particular focus on CNNs [1, 2, 18, 27, 28]. However, there is a noticeable lack of direct comparative analyses between models under identical conditions. Typically, studies train a model and compare its performance against previously reported results on the same dataset. Moreover, as we discuss later, it is common for these studies to use varying data splits within the same dataset, further complicating meaningful performance comparisons.

Recently, transformer architectures have gained significant prominence in both semantic segmentation of natural images and the broader field of machine learning [26, 31, 35]. Building on these advancements, this study aims to evaluate the performance of transformer-based models for the semantic segmentation of seismic facies. To achieve this, we perform a fair comparison using identical parameters, datasets, and data splits across all models, including CNNs known for strong performance in segmenting seismic facies and transformer-based models that achieve state-of-the-art results in natural image segmentation. Beyond quantitative analysis, we also carry out a qualitative investigation to examine the strengths and weaknesses of the models' predictions. Building on this, we selected our best-performing transformer model to undergo a hyperparameter search, aiming to determine whether its performance could be improved beyond the established baseline.

The objective of this work is to assess the viability of segmentation models based on the Transformer architecture for the task of semantic segmentation of seismic facies and to explore their performance in practical scenarios. To this end, we compare five segmentation models: three Transformer-based and two CNN-based. Additionally, we offer a critique of certain practices employed in state-of-the-art approaches that affect

both the reliability of reported results and the comparability and reproducibility of these works.

This work contributes to the field by providing a comparative analysis of the architectures Deeplab V3+ [8], Deeplab V3 [7], SegFormer [31], Segmenter [26], and SETR-PUP [35]. All models are evaluated using identical parameters, the same dataset, and consistent data splits, enabling a direct performance comparison. Additionally, we compare the results of our experiments against the current state of the art, offering a reference point to our models against the best-performing models in the field. As part of this analysis, we also highlight key differences between the methods used in this work and those reported in related studies. Our results were published at the 2024 annual EAGE conference [13].

In addition, we present a hyperparameter search performed on the best model identified during the comparative analysis, aiming to extract its optimal performance. To facilitate this process, we developed a new machine learning framework designed to address specific challenges encountered during our research, providing greater flexibility and control over the experimental setup.

The remainder of the text is structured as follows: First, on Chapter 2, we present the fundamental concepts necessary for a comprehensive understanding of the study, followed by Chapter 3, with a discussion of the seismic facies segmentation field. Next, on Chapter 4, we describe the methods and materials employed in this work, followed by an overview of the results obtained on Chapter 5 and Chapter 6. Finally, we summarize the conclusions on Chapter 7, highlight the challenges and solutions encountered during the execution of the study, and suggest future directions to complement and expand upon the findings.

Chapter 2

Fundamental Concepts

This section is dedicated to the understanding of the fundamental concepts explored on this work. Here we will define important terms, clarify key concepts, and provide the theoretical foundation necessary to understand the methodologies and analyses presented. This includes an overview of seismic facies and their significance in geoscience, the basics of semantic segmentation as a machine learning task, and an introduction to the main architecture and techniques utilized in this study, the transformer. By establishing this groundwork, we aim to ensure a clear and comprehensive understanding of the principles guiding the work.

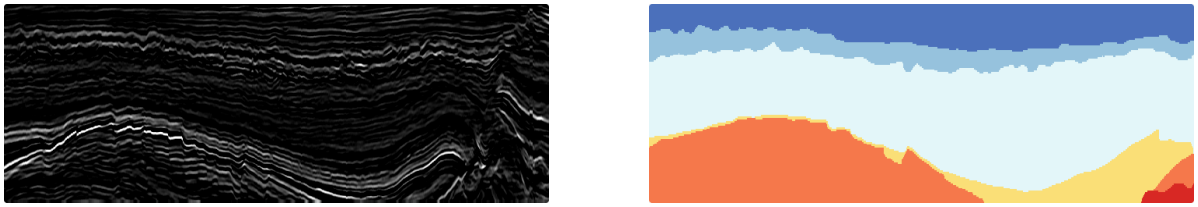
2.1 Semantic Segmentation

Semantic Segmentation is a sub-area of computer vision in which a class label is attributed to each pixel in an image [19, 20]. In Figure 2.1 we can see some examples of natural images, on top, and their respective segmentation masks from the ADA20K segmentation dataset [36]. While it shares similarities with other detection and delineation techniques, semantic segmentation stands out for its ability to identify entities at the pixel level rather than approximating their location with methods like bounding boxes. This enhanced precision is critical for tasks requiring fine-grained detail, such as autonomous vehicle navigation, medical diagnostics, and precision farming, among others. This precision advantage has brought semantic segmentation to the forefront of computer vision, significantly boosting its popularity and driving rapid advancements in the field.

Figure 2.1: Examples of images and segmentation masks form the dataset ADA20K. Image adapted from Zhou *et al.* [36]

2.2 Seismic Facies Segmentation

A seismic facies unit refers to a sedimentary unit characterized by a seismic signature distinct from the surrounding units [32]. Seismic facies segmentation involves applying semantic segmentation techniques to seismic data to differentiate between the various facies present in an image. Figure 2.2 shows and example of a seismic image and its segmentation mask.



(a) Example of seismic image.

(b) Example of labels mask.

Figure 2.2: Example of seismic image and its corresponding labels mask.

This process begins with seismic data collection, which typically involves generating seismic waves using a controlled source at a survey location. The waves propagate through the subsurface and are captured by receivers, such as geophones or hydrophones, positioned on the surface [33]. The raw data undergoes extensive processing to produce a final seismic volume, which serves as the input for segmentation tasks. Figure 2.3 shows the processes of collecting seismic data at sea. Considering this collection at sea, an inline refers to a panel from the seismic volume that runs parallel to the path of the survey ship, while a crossline is a panel that runs perpendicular to the inline, as shown in Figure 2.4.

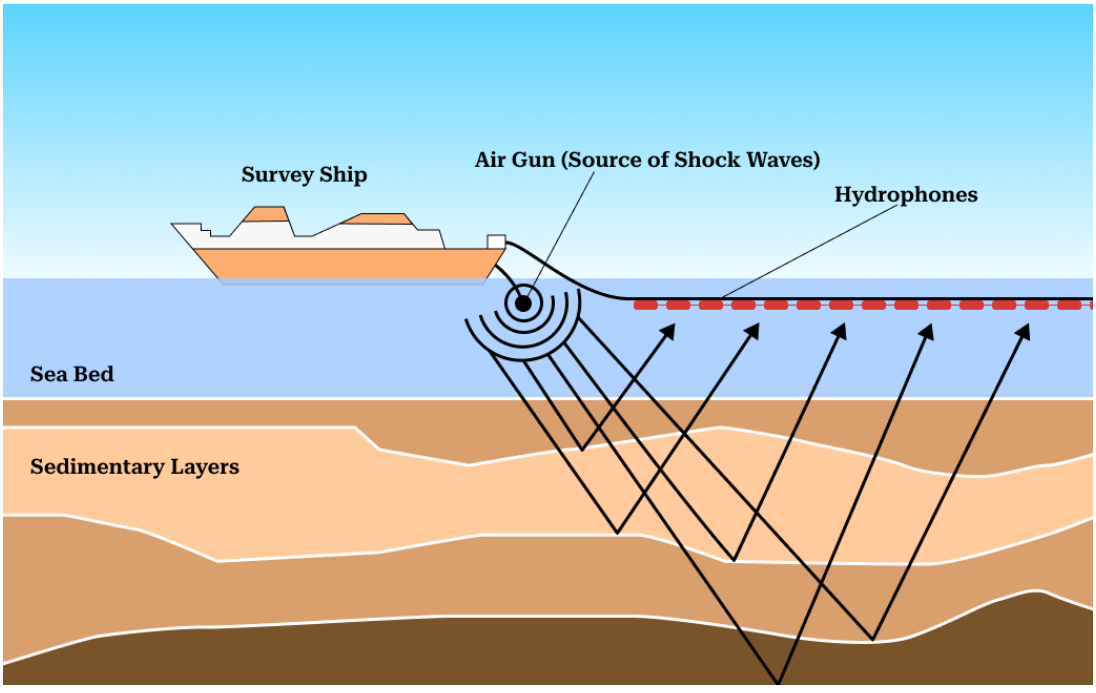


Figure 2.3: Processes used to collect seismic data offshore.

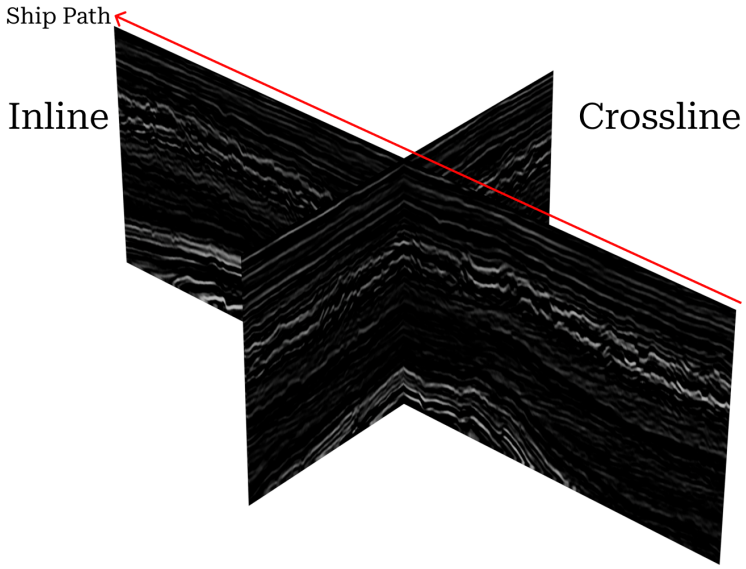


Figure 2.4: Example of an inline and crossline in a cube. The red arrow represents the ship's path.

An important characteristic of seismic data is that adjacent panels, whether inlines or crosslines, are highly similar. Figure 2.5 presents a comparison between three inlines, 198, 199, and 200, from the F3 dataset. As shown, all inlines exhibit very similar patterns, a characteristic that must be carefully considered when defining the dataset's data splits.

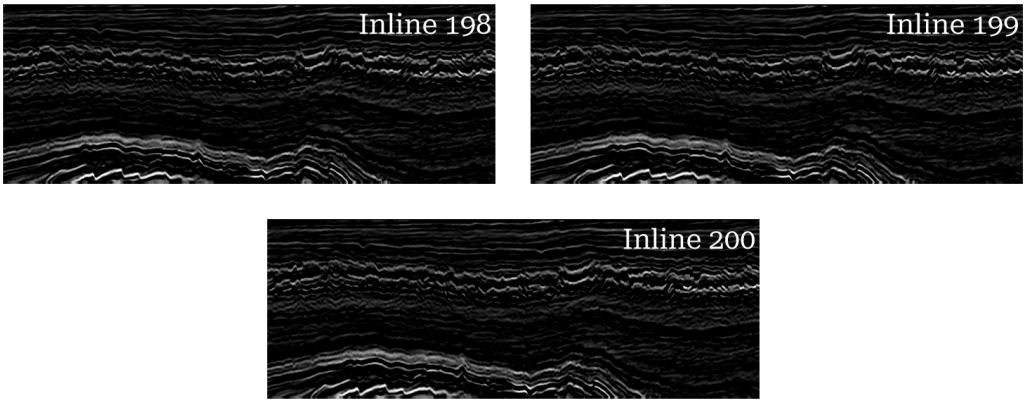


Figure 2.5: Example of an inline and crossline in a cube. The red arrow represents the ship's path.

2.3 Transformers

Transformers are a family of machine learning architectures first introduced by Vaswani *et al.* [29]. Initially designed for natural language processing (NLP) tasks, they serve as the backbone for influential models such as BERT [10] and GPT [22]. Following their transformative success in NLP, the transformer architecture was extended to the computer

vision domain. The Vision Transformer (ViT) [11] marked a significant breakthrough by applying transformer-based models to image classification. This success paved the way for further adaptations of transformers across various computer vision tasks, including semantic segmentation, thereby broadening their impact and utility. In Figure 2.6 we can see a diagram for the ViT architecture.

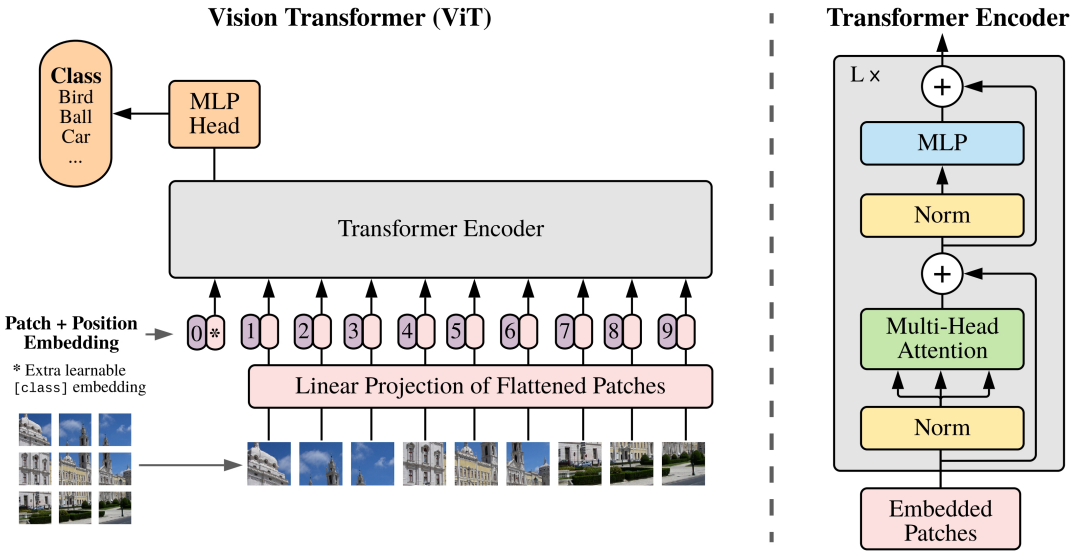


Figure 2.6: ViT Architecture as described by Dosovitskiy *et al.* [11], borrowed from Dosovitskiy *et al.* [11]. Licensed under the Apache License 2.0

The Vision Transformer (ViT) architecture consists of two primary components: the transformer encoder, which serves as the backbone, or encoder, and, in the original ViT, a Multi-Layer Perceptron (MLP) Head as the projection head. For classification tasks, an input image of size $C \times H \times W$, where C is the number of channels, H is the image height, and W is the image width, is divided into patches of size $P \times P$. Each patch is flattened into a vector of size $P^2 \times C$, which is then passed through a linear projection layer (a fully connected layer) to create a projection of size D. The projections are then fed into a positional embedding encoder, which assigns each projection a positional reference. This allows the transformer encoder to establish spatial relationships between the patches.

After receiving their positional encoding, the patches pass through the transformer encoder layer. The transformer encoder consists of a series of transformer layers, L, as shown to the right of Figure 2.6. Within each transformer layer, the patches first go through a normalization layer, followed by a multi-head attention layer. This attention mechanism computes global dependencies between the patches, enabling the model to focus on multiple regions of the image simultaneously. The output of the multi-head attention layer is then combined with the input using a skip connection. The resulting data is passed through another normalization layer, followed by a multi-layer perceptron (MLP), which consists of linear layers with non-linear activations in between.

Once the patches pass through the transformer encoder, they are sent to an MLP head where the classification occurs, utilizing the encoding provided by the transformer layers. When addressing tasks beyond classification, the only required modification is replacing the MLP head with a different task-specific head, which could be an MLP or another

structure better suited to the task at hand.

2.4 Segmentation Models

On this work we employ a series of segmentation models. This section is dedicated to present each model's architecture and their inner workings.

2.4.1 Deeplab V3 and Deeplab V3+

The Deeplab V3 and Deeplab V3+ architectures are quite similar. Thus, we will first describe the Deeplab V3 architecture and then highlight the differences between the two.

The Deeplab V3 architecture consists of a backbone, typically a CNN-based model, followed by an Atrous Spatial Pyramid Pooling (ASPP) module, an upsampling layer, and, finally, a pixel-wise classification layer.

The backbone functions as a feature extractor and offers significant flexibility for customization based on the user's requirements. If needed, the input image can be resized and normalized before being processed by the backbone.

After the backbone, the representation is fed to the ASPP layer, which performs convolutions with varying dilation rates to capture both fine-grained features and broader information. A global average pooling operation is also included to capture global contextual information.

Once out of the ASPP layer, the resulting data passes through an upsampling operation, returning to the original image size. It then undergoes pixel-wise classification using a 1×1 convolution followed by a softmax or sigmoid activation, producing a prediction of size $H \times W \times C$, where C is the number of possible classes.

With the Deeplab V3 architecture outlined, we can discuss the improvements introduced by Deeplab V3+. The two main differences between the two architectures are the addition of a decoder layer and the incorporation of lower-level features from the encoder.

After the ASPP layer, Deeplab V3+ incorporates a lightweight decoder to refine segmentation, particularly along object borders. This decoder integrates high-resolution features from the early stages of the backbone, enhancing detail. Additionally, it introduces a progressive upsampling strategy, as opposed to the single upsampling step used in Deeplab V3. Table 2.1 summarizes the differences between the Deeplab V3 and Deeplab V3+ models.

Aspect	Deeplab V3	Deeplab V3+
Decoder	None	Lightweight decoder
Upsampling	Single step after ASPP	Gradual via decoder
Low-Level Feature Fusion	Not utilized	Integrated from backbone
Boundary Refinement	Less precise	Improved
Computational Cost	Lower	Slightly higher
Application	Faster, coarser segmentation	Higher accuracy, better detail

Table 2.1: Differences between Deeplab V3 and Deeplab V3+

2.4.2 SegFormer

SegFormer is an architecture that combines the ViT architecture with convolutional techniques. It aims to avoid computational bottlenecks while maintaining high-quality pixel-wise predictions. The architecture consists of a hybrid transformer-CNN encoder and a lightweight MLP-based decoder. Figure 2.7 shows a diagram borrowed from Xie *et al.* [31] detailing the SegFormer architecture.

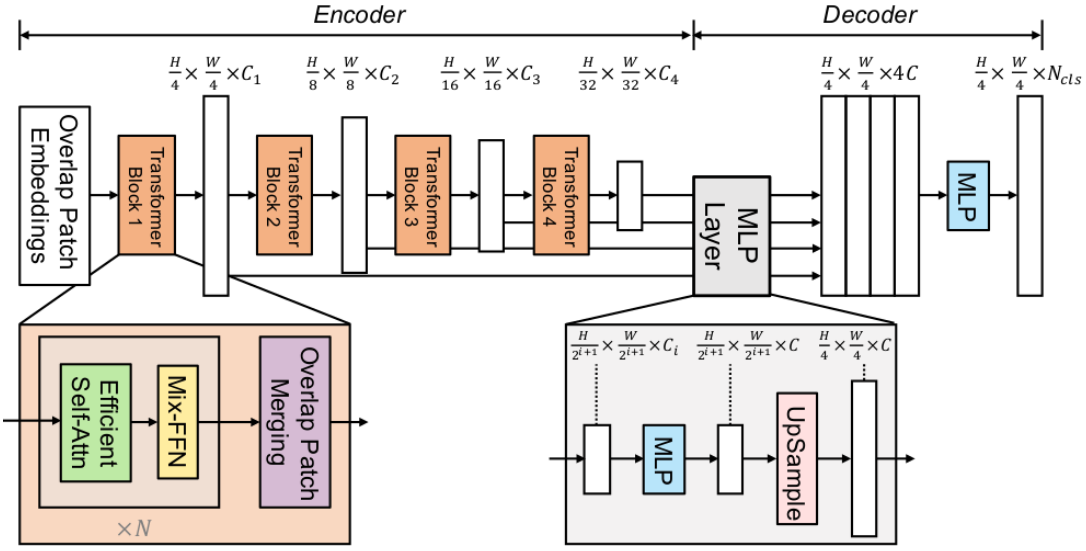


Figure 2.7: SegFormer architecture as described by Xie *et al.* [31], borrowed from Xie *et al.* [31]. Licensed under the Apache License 2.0

SegFormer's encoder combines transformer and CNN layers to extract multi-scale features. It employs overlapping patch embedding layers to convert the image into feature embeddings suitable for the transformer. The patch embedding layer uses a convolutional layer with overlapping patches to capture local features, while the stride reduces the resolution, preserving spatial relationships.

The transformer blocks in SegFormer consist of an efficient self-attention layer and a Mix Feed Forward network. The self-attention layer computes global dependencies between patches, capturing contextual information across the entire image. The Mix Feed Forward network enhances feature representation by combining spatial and channel information, enabling a more comprehensive understanding of the features.

At the end of each block, the feature map's resolution is halved while the channel depth increases. This hierarchical structure of SegFormer ensures the model captures both fine-grained details and global contextual information effectively.

SegFormer's decoder is relatively simple, consisting of a feature fusion stage performed by an MLP followed by upsampling to the final resolution. The feature fusion combines representations from different stages of the encoder into a unified representation. Subsequently, another MLP converts this fused representation into an output image with the desired number of classes. This process produces an output of size $H/4 \times W/4 \times N_{\text{cls}}$, which can then be further upscaled to match the original image dimensions.

2.4.3 Segmenter

The Segmenter architecture [26] is an entirely transformer-based design, utilizing a ViT encoder combined with a transformer-based decoder. Figure 2.8, adapted from Strudel *et al.* [26], presents a diagram outlining the Segmenter architecture.

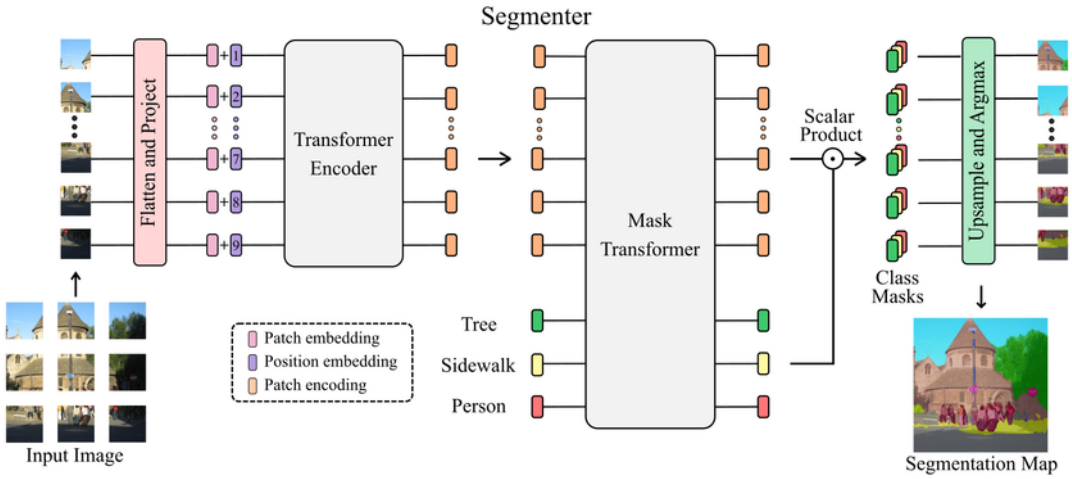


Figure 2.8: Segmenter architecture as described by Strudel *et al.* [26], borrowed from Strudel *et al.* [26]. Licensed under CC BY 4.0

Segmenter's encoder is a standard ViT encoder, used without any modifications. The distinguishing feature of the Segmenter architecture lies in its decoder. Segmenter employs a mask transformer decoder, which takes the refined patch embeddings from the ViT encoder and, using a set of class masks generated by the decoder, segments each patch individually.

The core functionality of the mask transformer layer lies in its ability to learn class embeddings, extract information through attention by combining patch embeddings with these class embeddings, and compute individual class masks. The class embeddings are learnable vectors, each representing a specific class (e.g., tree, sidewalk, person). These embeddings start with random values and are refined during training. In the attention layers, the class embeddings act as "guides", interacting with the patch embeddings via the attention mechanism. Each class embedding captures high-level information about its associated class, directing the mask transformer to focus on the relevant areas of the image.

On the multi-head attention layers the class embeddings are used as queries while the patch embeddings are used as keys and values. The attention mechanism makes the class embeddings attend to the patch embeddings and identify each patch that belongs to their class, allowing class embeddings to capture combined information about its class from relevant patches. At the end of the process the class embeddings are updated with new contextual information about their classes, informed by the patch embeddings.

After the attention mechanism updates the class embeddings, the mask transformer computes class-specific masks for each class. This is achieved by calculating a scalar product between each class embedding and the patch embeddings, producing a mask for each class. Each value in the mask represents the probability of the corresponding class

being present in a specific patch. These scalar product results are then passed through a sigmoid function to normalize the values between 0 and 1, making them interpretable as probabilities.

The masks are then upscaled to match the image's original resolution. After upscaling, they pass through an argmax layer, which assigns each pixel to the class with the highest probability, effectively producing the final segmentation map.

2.4.4 SETR

The Semantic Transformer (SETR) is a vision transformer-based architecture comprising a ViT encoder and two possible decoder configurations: Progressive Upscale (PUP) and Multi-Level Feature Aggregation (MLA). In this work, we focus exclusively on the PUP decoder and will not delve into the details of the MLA architecture. Figure 2.9 illustrates the architecture, showing the ViT encoder (a), the PUP decoder (b), and the MLA decoder (c).

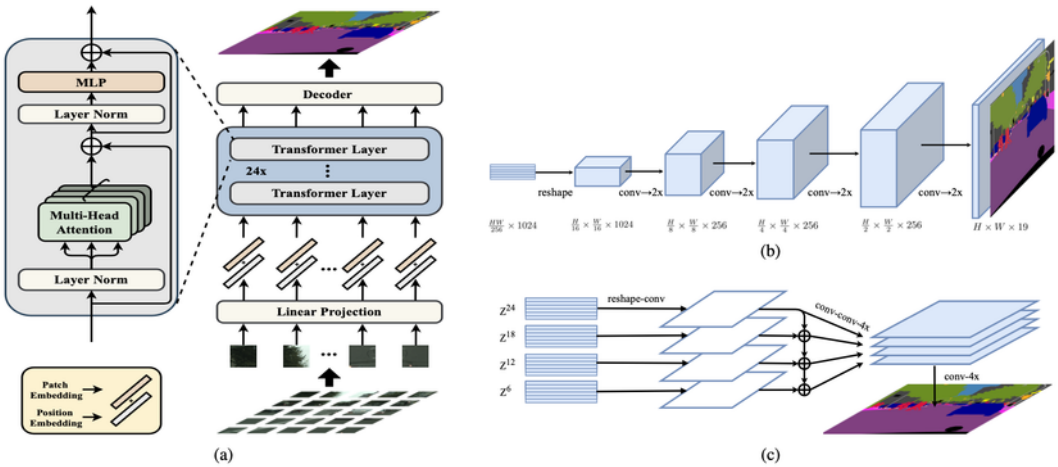


Figure 2.9: SETR architecture as described by Zheng *et al.* [35]. The SETR-PUP, used on this work, is the combination of diagrams a and b. borrowed from Zheng *et al.* [35]. Licensed under CC BY 4.0

SETR's ViT encoder is a standard Vision Transformer enhanced with the ability to pass residual information from intermediate layers, which is utilized for an auxiliary loss mechanism. The Progressive Upscale (PUP) decoder consists of multiple upsampling stages that incorporate convolutional layers and skip connections, allowing for detailed spatial reconstruction. This approach effectively reconstructs pixel-level predictions from class-level information, maintaining spatial detail and ensuring accurate alignment between the predictions and the original image.

The overall SETR architecture consists of a single encoder, one primary decoder head, and three auxiliary heads, each sharing the same architecture as the decoder head. The auxiliary heads leverage residual information extracted from specific intermediate layers of the encoder. These intermediary predictions are combined with the decoder head's prediction during the loss calculation. In this process, the predictions from the auxiliary heads are assigned a lower weight compared to the decoder head's prediction.

2.5 Transfer Learning

Transfer learning is a technique used to enhance a machine learning model's performance on a specific task or dataset by leveraging knowledge gained from a different, possibly related, task or dataset [21]. In essence, transfer learning utilizes insights from one setting, whether a dataset or task, to improve the model's performance in another context [3].

In this work, we used transfer learning to leverage the knowledge gained from training our models' encoders on the ImageNet 21K [23] classification task to enhance segmentation performance on seismic facies. Transfer learning is particularly advantageous in our experiments, as our models are relatively large, with tens of millions of parameters, while our datasets are relatively small, containing only 1,300 to 1,500 examples.

The combination of large models with small datasets often leads to overfitting [34]. To mitigate this risk, we employed transfer learning, which not only helps reduce the likelihood of overfitting [34] but also leverages valuable knowledge derived from training on a large-scale dataset like ImageNet 21K.

2.6 Hyperparameter Search

The definition of hyperparameters for a model has been a longstanding challenge in machine learning. This issue is particularly prominent today, given the extensive range of parameters associated with deep neural networks, including those related to architecture, regularization, and optimization. Developing an automated method to configure these parameters for optimal model performance is a critical step in training machine learning models [16]. Various algorithms can be employed to perform this search. Generally, these algorithms operate by taking a predefined range of hyperparameters, selecting specific combinations for testing in individual trials, executing these trials, and collecting performance metrics. Based on these results, the algorithm identifies the optimal set of hyperparameters within the specified range.

Acquisition functions determine the next point x to evaluate, balancing exploration, searching in unobserved regions, and exploitation, focusing on areas likely to yield a high function value. Common acquisition functions include the Expected Improvement (EI) function, the Knowledge Gradient (KG) function, as well as Entropy Search (ES) and Predictive Entropy Search (PES). Each of these functions aims to optimize the trade-off between gaining new information about the function and refining the search near promising regions [5].

The optimization process begins with an initial set of samples selected through a space-filling design. These samples are used to train the Gaussian Process (GP), which updates the posterior distribution over $f(x)$. Next, the acquisition function is optimized to determine the next point x for evaluation. The function $f(x)$ is then evaluated at this chosen point. This iterative cycle continues until the search budget is exhausted. The search budget can be defined as a limit on the elapsed time, a performance threshold to surpass, or a predefined number of trials without significant improvement [5].

Chapter 3

Related Work

The field of semantic segmentation has witnessed substantial advancements in recent years, both in natural image processing and seismic data analysis. In this chapter, we discuss significant contributions to segmentation of seismic facies with machine learning. The discussion is organized by methodology, distinguishing between transformer-based approaches and other techniques. Additionally, we review and analyze the current state of the art as a whole. This chapter concludes by outlining the hypotheses and research questions that guide this work, establishing a foundation for the subsequent exploration and experimentation.

3.1 Seismic Facies Segmentation with Transformers

Transformers, as a relatively new architecture, have recently gained traction in seismic facies segmentation. Bosco de Miranda and Gattass (2023) [4], inspired by the advancements made by Zheng *et al.* (2020) [35], explored the application of transformer models to the F3 dataset, which is one of the datasets we will explore in this work. Further details about the F3 dataset can be found in Section 4.2.

Bosco de Miranda and Gattass trained two SETR-PUP models: one utilizing masked autoencoders [14] and another leveraging pre-trained weights from the ImageNet dataset. While their results were promising, they emphasized the need for further research to obtain conclusive insights and fully validate the potential of transformer-based approaches in this field.

Still within the context of the F3 dataset, Wang *et al.* (2023) [30] proposed the U-Segformer-Hyper model for seismic facies segmentation. This model combines the Segformer architecture with hypercolumn representations to improve feature extraction. By employing a U-shaped encoder–decoder structure, the model efficiently fuses multi-layer features while integrating a self-attention mechanism to capture complex seismic patterns effectively. Their approach showcases the potential of combining advanced architectural elements to address the unique challenges of seismic data segmentation.

Switching to the Seam AI dataset, Sheng *et al.* (2023) [24] proposed a transformer based architecture that achieved promising results. They propose a foundational model, capable of multiple tasks, not only seismic facies segmentation, pre-trained on a big seismic

dataset and fine tuned on the Seam AI. An important detail to highlight is that Shang *et al.* created a specific data split that differs from what we used on our tests, which was proposed by Alaudah *et al.* (2019) [2].

Guo *et al.* (2024) [12] introduced an innovative approach for adapting foundation models (FMs) from computer vision to geophysical tasks, proposing a workflow that fine-tunes pre-trained models, such as DINOv2, for seismic data downstream applications. Their method employs a parameter-efficient fine-tuning strategy using LoRA [15] layers and evaluates various decoders to enhance feature extraction for seismic facies, salt domes, and faults. This cross-domain adaptation significantly reduces the reliance on large geophysical datasets while showcasing its potential in advancing seismic data analysis.

3.2 Seismic Facies Segmentation with other ML techniques

In the context of the F3 dataset, Alaudah *et al.* (2019) [2] were pioneers in applying ML-based semantic segmentation techniques to seismic facies analysis. Despite utilizing a relatively simple model with modest results, their work laid the foundation for subsequent research into machine learning models for segmenting the F3 dataset. Building on these efforts, Tolstaya and Egorov (2022) [27] proposed a UNet-based model and introduced several techniques to improve model accuracy, including advanced data transformations, pseudo-labeling, depth information, and more. Their work tested these approaches on both the F3 and Seam AI datasets, demonstrating their versatility and effectiveness.

Some studies have explored model ensemble approaches, where multiple models are trained and subsequently vote for the classification of each pixel. Trindade and Roisenberg [28] proposed the AH-Ensemble network for 3D seismic facies classification. This method combines transfer learning from 2D CNNs with geological heuristics to reduce computational costs. By utilizing orthogonal slices of seismic data, they transferred 2D-trained parameters into a 3D convolutional model, enabling efficient seismic facies segmentation.

Similarly, Abid *et al.* (2022) [1] developed a seismic facies segmentation method using an ensemble of CNNs. Their approach employed the DeepLab V3+ and SegNet architectures, utilizing pretrained ResNet-18 and VGG16 encoders to segment seismic data from the F3 block of the North Sea. These ensemble strategies highlight the potential of combining multiple models to improve segmentation accuracy and robustness. Both of this models were applied to the F3 dataset.

On the Seam AI dataset, Kaur *et al.* (2023) [18] proposed two innovative models: one based on the DeepLab V3+ architecture and another leveraging generative adversarial networks (GANs). These models achieved peak performance in the context of the Seam AI dataset, marking a significant advancement in seismic facies segmentation. Kaur's contributions demonstrated the potential of combining traditional segmentation architectures with GANs, pushing the field forward by improving accuracy and robustness in seismic data analysis.

3.3 State-of-the-art Discussion

The field of seismic facies segmentation is still in a developmental stage, with significant variability in datasets, data splits, and even labeling approaches for the same data. These inconsistencies pose challenges when attempting to compare the performance of different models. Additionally, the absence of a standardized large-scale dataset limits the potential for training large models, as there is an increased risk of overfitting due to insufficient data diversity and volume. Establishing uniform datasets and protocols is essential for advancing the field and enabling more robust and comparable evaluations of segmentation models. This work addresses the comparison challenge by utilizing the same datasets, F3 and Seam AI, along with consistent data splits across all models. This ensures that the results can be fairly and reliably compared, providing a solid basis for evaluating model performance.

On Table 3.1, we summarize the works discussed in this chapter, highlighting their methodological similarities and differences. Notably, particularly for the Seam AI dataset, there is no standardized approach for splitting the dataset into train/test groups. This lack of consistency makes direct comparison between results reported by different authors challenging. Additionally, some works did not explicitly state their data splits, referencing only Alaudah *et al.*'s benchmark. In such cases, we assumed the splits were identical to ours, as we followed Alaudah *et al.*'s benchmark. It is also worth noting that validation splits were excluded from the classification as "same" or "other," since Alaudah *et al.* did not define a validation split in their benchmark.

Also, in Table 3.1, we present the best results reported in each work. The first issue made evident by this is the inconsistency in the metrics reported. While the most common metric is the mean intersection over union (mIoU), others prefer to report the frequency-weighted intersection over union (FWIU) or the mean F1 score (MF1). This inconsistency alone makes comparing the works challenging.

Beyond that, even when authors report the same metric, comparisons are not straightforward. For instance, among the works that reported mIoU, one might initially conclude that Abid *et al.*'s [1] model demonstrates the best performance on the F3 dataset. However, Abid *et al.* not only used a different data split from other works, making direct comparisons impossible, but also employed an improper data splitting strategy that likely led to leakage of the test set into the training data. Specifically, they split their dataset randomly, which, due to the physical properties of seismic data, where consecutive examples are physically very similar as shown on Section 2.2, can result in nearly identical examples being present in both the training and test sets, thereby invalidating the training process.

Work	Dataset	Train/Test	Architecture	Best Results
Alaudah *et al.* (2019) [2]	F3	Same	CNN	0.844 (FWIU)
Trindade and Roisenberg (2021) [28]	F3	Same[2]	CNN	0.814 (FWIU)
Abid *et al.* (2022) [1]	F3	Other	CNN	0.9392 (mIoU)
Tolstaya and Egorov (2022) [27]	F3 & Seam AI	Same	CNN	0.85[3], 0.77[4](mIoU)
Wang *et al.* (2023) [30]	F3	Same[2]	Transformer	0.836 (FWIU)
Miranda and Gattass (2023) [4]	F3	Same	Transformer	0.7836 (mIoU)
Kaur *et al.* (2023) [18]	Seam AI [1]	Other	CNN	0.9608 (MF1)
Sheng *et al.* (2023) [24]	Seam AI	Other	Transformer	0.7980 (mIoU)
Guo *et al.* (2024) [12]	Seam AI	Other	Transformer	0.6885 (mIoU)

1 - Kaur *et al.* used different labeling for their work. 2 - Work did not explicitly stated their data split. 3 - Results on the F3 dataset 4 - Results on the Seam AI

Table 3.1: Comparison of works and their characteristics. Train/Test refers to the split of the dataset used by the authors on their works. For cases where the data split was not explicitly stated but referenced Alaudah *et al.*'s benchmark, we assumed they used the same data split as ours. Validation splits were excluded from the comparison, as Alaudah *et al.* did not specify one. The best results column shows the best metric reported by the authors.

Chapter 4

Materials and Methods

In this chapter, we outline the materials and methods used in this study. This includes detailed descriptions of the datasets, models, hyperparameters, and data augmentation and processing techniques. Every step and resource employed in the research process is meticulously documented to ensure reproducibility. Additionally, all configuration files used in this work are openly accessible on GitHub[1].

4.1 Models

In this comparison, we trained five models in total: three transformer-based models, SegFormer [31], Segmenter [26], and SETR [35], selected for their state-of-the-art performance in common image segmentation. The remaining two models are CNN-based, Deeplab V3+ [8] and Deeplab V3 [7]. Deeplab V3+ was chosen due to its reported state-of-the-art performance, while Deeplab V3 was included based on its promising results in internal testing conducted by Petrobras, which worked closely with us. On Table 4.1 we have listed the models, their backbones, their parameter count and expected input shape.

Model	Backbone	Parameter Count	Expected Image Input
Deeplab V3	ResNet 50	25.6M	512x512
Deeplab V3+	ResNet 50	25.6M	512x512
SegFormer	MIT-B5	82M	512x512
Segmenter	ViT-L	307M	512x512
SETR	ViT-L	307M	512x512

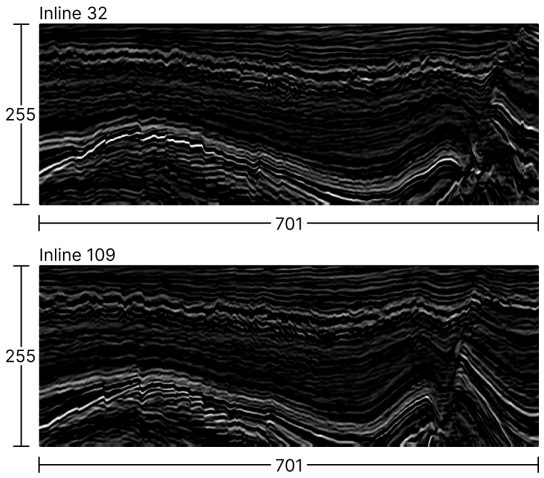
Table 4.1: Models' backbones metrics. Parameter Count in millions (M). The ResNet 50 was chosen over the 101 since it performed better in our exploratory tests.

All models were initialized with pre-trained weights from the ImageNet 21k dataset [23], which were used to initialize their backbones. These weights are essential for the models' performance, as they all have large backbones, and the seismic facies segmentation datasets have a relatively small number of examples. Initializing the models without pre-training resulted in poor performance and showed signs of overfitting on the training set, but this was not investigated further.

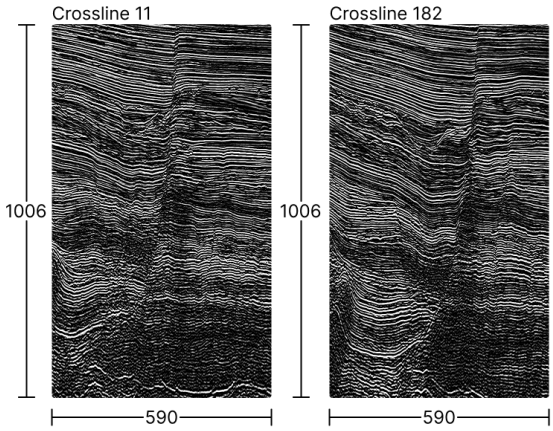
[1]Config files available at `https://gabrielbg.com.br/Evaluation-of-Transformers`

4.2 Datasets

Datasets for semantic segmentation of seismic facies are relatively uncommon due to the high time and resource costs involved in labeling large datasets and most companies keep their proprietary datasets confidential. In this experiment, we used two datasets: the F3 dataset and the Seam AI Parihaka dataset. In Figure 4.1 we can see two inlines and two crosslines from the F3 and Seam AI datasets respectively.



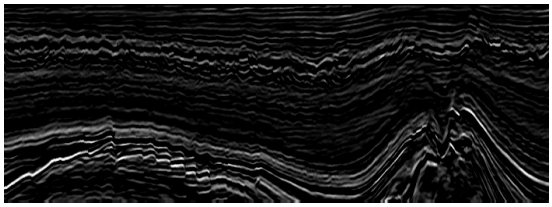
(a) F3 inlines 32, on top, and 109 on the bottom.



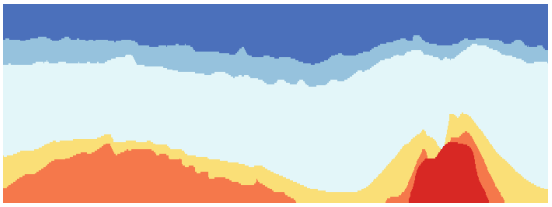
(b) Seam AI crosslines 11, on the left, and 182 on the right.

Figure 4.1: Examples from both datasets.

The Netherlands F3 dataset [25], referred to here as the F3 dataset, contains data from a seismic survey conducted in the North Sea, within Dutch waters. Compared to common image datasets, it is relatively small, consisting of 601 inlines and 901 crosslines, totaling 1,502 examples and 4.7 GB of data. For this experiment, we followed the methodology described by Alaudah *et al.* [2], which involved merging some of the dataset's classes to address very thin and unbalanced categories. The dataset consists of six classes: Upper layer, Middle layer, Lower layer, Rijnland/Chalk layer, Scruff layer, and Zechstein layer. Figure 4.2 shows an example of seismic data and its label from F3's training set. We also adopted Alaudah *et al.* 's [2] train/test dataset division, as shown in Figure 4.4a.



(a) Example of seismic image.

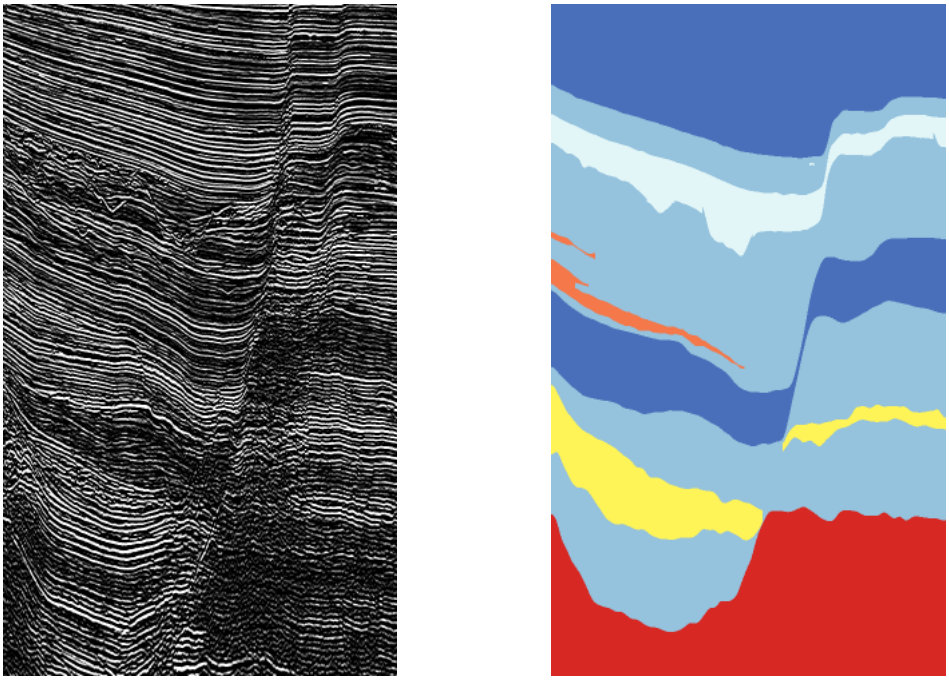


(b) Example of labels mask.

Figure 4.2: Example of seismic image and its corresponding labels mask from the F3 dataset.

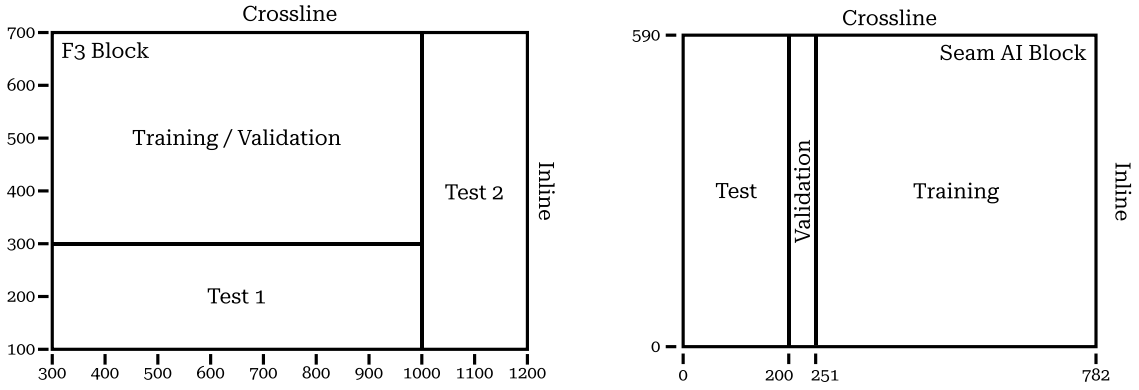
The second dataset used in this experiment is the Seam AI Parihaka dataset [9], referred to here as the Seam AI dataset. This dataset contains data from a seismic survey in

New Zealand waters and was made public through a challenge on the AI Crowd platform, hosted by SEAM [9]. When compared with the F3 dataset, it comprises 590 inlines and 782 crosslines, totaling 1,372 examples, with each example being larger in size, amounting to 8.8 GB of data. The Seam AI dataset includes six classes: Basement/Other, Slope Mudstone A, Mass Transport Deposit, Slope Mudstone B, Slope Valley, and Submarine Canyon System. Figure 4.3 shows an example of seismic data and its label from Seam AI's training set. No layer merging was performed on this dataset. We also followed Tolstaya and Egorov's train, validation, and test split methodology, as illustrated in Figure 4.4b, which details the data split used by our work.



(a) Example of seismic image. (b) Example of labels mask.

Figure 4.3: Example of seismic image and its corresponding labels mask from the Seam AI dataset.



(a) F3 (b) Seam AI

Figure 4.4: Datasets Partitioning used by in this work.

4.3 Augmentations and Data Processing

One way to increase the dataset size without labeling more examples is to apply augmentation to the training images. This also serves as a regularization technique, making the models more resilient to the applied transformations. For this experiment, we applied two transformations: random position cropping and random flipping. After cropping, the images are upscaled to the models' expected input size, which in this case is 512x512. If necessary, padding with zeros is applied to maintain the required dimensions.

We also introduced data processing steps into our pipeline, such as normalizing the data to match ImageNet values, since the pretrained weights were derived from that dataset. This normalization ensures better compatibility between the data and the weights, potentially enhancing the performance of transfer learning.

4.4 Evaluation Metrics and Losses

As previously discussed, the goal of this experiment is to fairly compare the performance of the models. To achieve this, we slightly adapted the MM Segmentation[2] reference parameters, originally designed for the ADE20K benchmark in conventional photography segmentation, as it provided a common configuration applicable to all models.

We made three primary modifications. First, we extended the training regiment to 1000 epochs to account for the longer convergence times typically required by Transformer architectures.

Second, we reduced the default learning rate from 0.01 to 0.001 to better suit the extended training regimen. Lastly, we replaced the standard cross-entropy loss with weighted cross-entropy loss to address class imbalance. The batch size was adjusted from 4 to 2 due to GPU memory constraints, rather than an arbitrary decision. Table 4.2 lists all the hyperparameters chosen for the models.

Hyperparameter	Value
Expected Input Size	512x512
Base Learning Rate	0.001
Loss Function	Weighted Cross Entropy
Weigh Decay	0.0005
Momentum	0.9
Batch Size	2
Optimizer	SGD
Epochs	1000

Table 4.2: List of Models' Hyperparameters.

As shown in Table 4.2, we used the cross-entropy loss function and the SGD optimizer to train our models. Cross-entropy loss is widely used in the semantic segmentation field [35, 17] and outperformed DICE loss, another common loss function in semantic segmentation, in our preliminary tests. Additionally, we applied class-weighted loss based

[2]MM Segmentation Github Page: `https://github.com/open-mmlab/mmsegmentation`

on the class distribution, giving more weight to classes with fewer examples. This was done to prevent the models from overpredicting the more common and unbalanced classes. Equation 4.1 describes the Weighted Cross-Entropy loss.

For evaluation metrics, we selected the mean intersection over union (mIoU), described on Equation 4.2, which is widely used in the context of semantic segmentation. Additionally, we report mean class accuracy (MCA), Equation 4.3, and pixel accuracy (PA), Equation 4.4, although these metrics were not used for ranking, just for comparison with works that did not report the mIoU metric.

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -\sum_{c=1}^{C} w_c \log \left(\frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} \right) y_{n,c} \tag{4.1}$$

$$\text{mIoU} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c + FN_c} \tag{4.2}$$

$$\text{MCA} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{T_c} \tag{4.3}$$

$$\text{PA} = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C} T_c} \tag{4.4}$$

4.5 Software and Tools

For this work, we primarily used the MMsegmentation framework. MMsegmentation is an open-source semantic segmentation framework built on top of PyTorch, and it is part of the OpenMMLab ecosystems. It has a modular design, offering pre-built models, datasets, and evaluation tools for various segmentation tasks. The framework supports state-of-the-art algorithms such as Deeplab V3+, FCN, and SETR, with an emphasis on flexibility and scalability. It facilitates easy model training, fine-tuning, and evaluation, supporting both single and multi-GPU setups, while also enabling seamless integration of custom components like backbones and heads.

Chapter 5

Experimental Results

In this chapter, we present and analyze the results from the training of the five models. The section is divided into two parts, one for each dataset used. For each dataset, we provide and examine the models' performance metrics. Additionally, we conduct a qualitative analysis of the models' predictions, highlighting their strengths and weaknesses.

5.1 Segmenting the F3 Dataset

For the F3 dataset, we report the mean intersection over union (mIoU), pixel accuracy (PA), and mean class accuracy (MCA). Table 5.1 displays the results for all five models. As shown, the SETR model achieved the best performance on this dataset, with an mIoU of 0.7849. Following closely were Segmenter and Deeplab V3, with mIoU scores of 0.7800 and 0.7781, respectively. The last two models were Deeplab V3+, with 0.7748, and SegFormer, which scored 0.7268.

Model	mIoU	PA	MCA
Deeplab V3	0.7781	0.9392	0.8742
Deeplab V3 +	0.7748	0.9365	0.8649
SegFormer	0.7268	0.9182	0.8198
Segmenter	0.7800	0.9353	0.8815
SETR	**0.7849**	0.9338	0.8828

Table 5.1: Models' performance on F3 dataset when trained with 1000 epochs. In bold the best mIoU between the models

We also trained the models for 150 epochs to experiment with a methodology commonly used in works such as Tolstaya and Egorov's [27]. In our experiment, training the models for 150 epochs led to significant underperformance, with some models showing a 12% reduction in their mIoU metric. The comparison between training for 150 and 1,000 epochs is shown in Table 5.2.

Table 5.2 also lists the best validation epoch for each model. Although the models were trained for 1,000 epochs, the model selected as final, for each architecture, was the one that achieved the highest mIoU on the validation set. As shown, most models reached their best performance near the 1,000th epoch, indicating that the additional training

Model	150	1000	Gain	Best Val Epoch
Deeplab V3	0.6918	0.7781	**12,47%**	850th
Deeplab V3 +	**0.7466**	0.7748	3.78%	890th
SegFormer	0.6688	0.7268	8.67%	980th
Segmenter	0.7173	0.7800	8.74%	940th
SETR	0.7369	**0.7849**	6.51%	990th

Table 5.2: mIoU gain on F3 when increasing the training regimen from 150 to 1000 epochs. We also present the best epochs in which each model reached peak validation performance. In bold the best mIoU between the models and the biggest gain in performance.

time was valuable and contributed significantly to their performance. Since the models achieved their best performance close to the 1,000th epoch, it is possible that allowing more training time could further improve their results. On average, the models required approximately 15 hours to complete training, with the two ViT-based models, Segformer, and SETR, requiring over 20 hours.

In terms of prediction quality, we present results from two selected inlines: inline 32 and inline 109. These inlines were chosen to represent challenging and easier cases, with inline 32 being more difficult and inline 109 being easier to predict. Analyzing the models' predictions provides additional insight into their performance, offering nuances that numerical metrics alone may not reveal.

In Figure 5.1, we see the ground truth for inline 32, followed by each model's predictions. At first glance, it is evident that all models struggled to segment the bottom left corner of the inline, which corresponds to the presence of a geological fault. Although SETR achieved the highest mIoU for this inline, with a score of 0.6886, its prediction is far from ideal. It misclassified both the Lower North Sea class, confusing it with the Upper and Middle North Sea classes, and failed to classify the Scruff class. In comparison, Deeplab V3+ performed with a slightly lower mIoU of 0.6857 but handled the misclassification more elegantly, only failing to classify the Scruff class. A comparison between the two predictions is presented in Figure 5.2a.

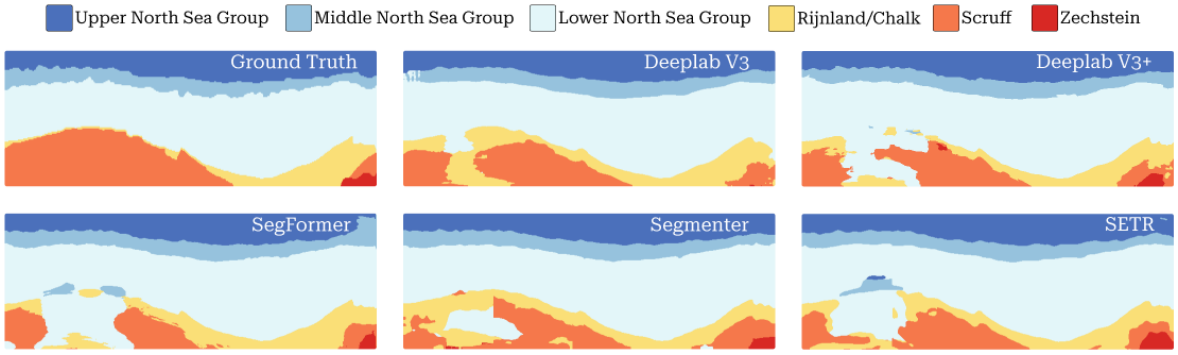


Figure 5.1: Segmentation of inline number 32 from the test split of the F3 dataset, comparing the performance of various models against the ground truth. The DeepLab V3, DeepLab V3+, SegFormer, Segmenter, and SETR models achieved scores of 0.6236, 0.6857, 0.6457, 0.6746, and 0.6886, respectively, for this specific inline.

The Segmenter also produced a reasonable prediction, successfully classifying the first three layers despite the fault region being a challenging area to segment. However, it exhibited a minor misclassification in the bottom right of the prediction, where other models did not encounter errors, as shown in Figure 5.2b. Deeplab V3 and SegFormer performed the poorest in this instance. Although Deeplab V3 has a higher overall mIoU than SegFormer, it was the least effective in this specific scenario, failing to accurately segment the bottom left region, the Zechstein class in the bottom right and the Middle north Sea Group on the top left, as shown in Figure 5.2c.

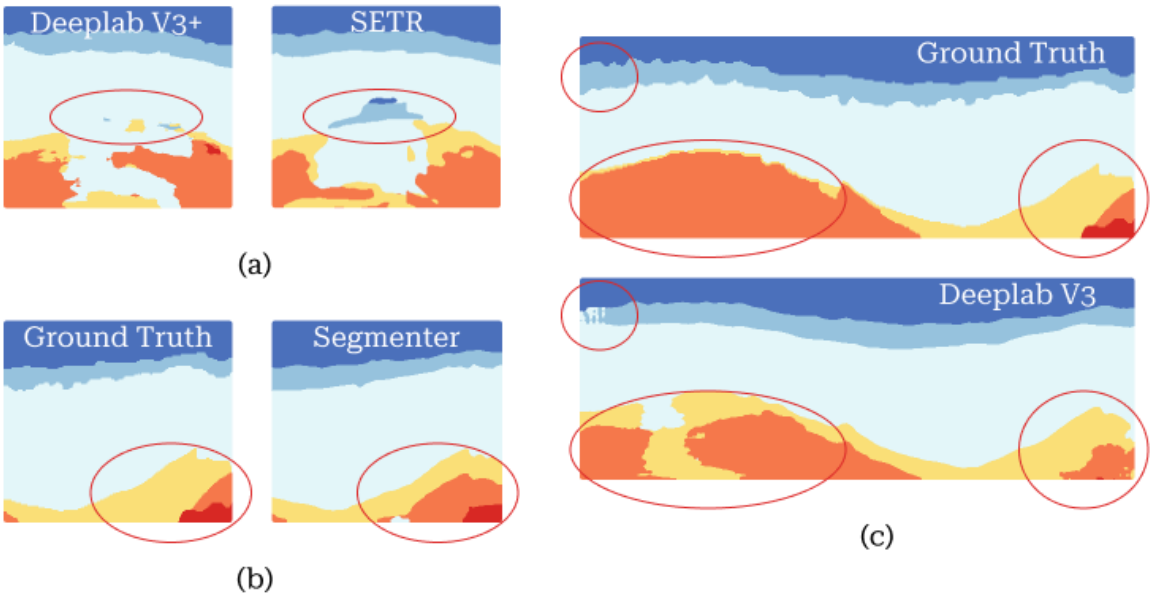


Figure 5.2: Comparison of predictions made on inline 32 from the F3 dataset, shown in full on Figure 5.1. Red circles highlight areas of interest. (a) Comparison between predictions from Deeplab V3+ and SETR. (b) Issues in the Segmenter prediction compared to the ground truth. (c) Defects observed in the Deeplab V3 prediction.

In inline 109, shown in Figure 5.3, where the fault is less prominent, most models achieved successful predictions. The mIoU scores were significantly better compared to the performance on inline 32. Deeplab V3 had the best performance, closely followed by SETR, with scores of 0.8529 and 0.8510, respectively, showing a negligible difference. Both models exhibited minimal errors along the facies borders.

The Segmenter and Deeplab V3+ models also delivered strong predictions, though with slightly more errors along the facies boundaries. The differences between the predictions and the ground truth can be observed in Figure 5.4. Remarkably, the SegFormer model stands out for its underperformance in this scenario, failing to segment the bottom right area and being the only model to show major issues in this region.

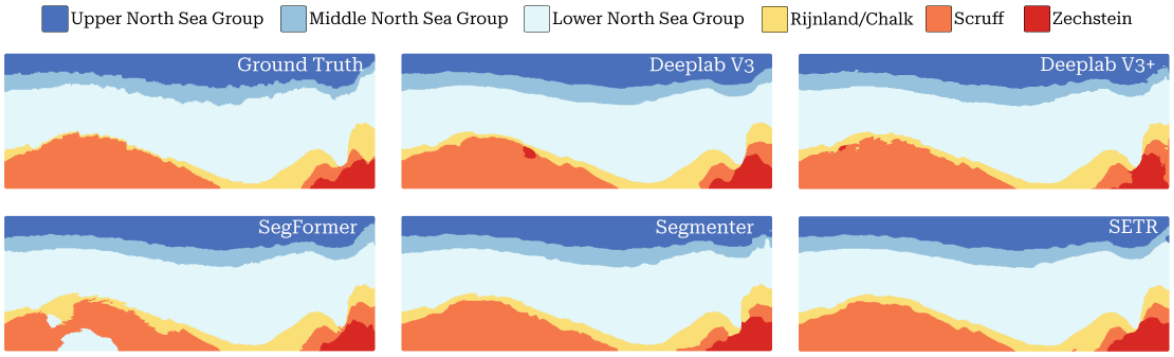


Figure 5.3: Segmentation of inline number 109 from the test split of the F3 dataset, comparing the performance of various models against the ground truth. The models, DeepLab V3, DeepLab V3+, SegFormer, Segmenter, and SETR, achieved scores of 0.8529, 0.8287, 0.8154, 0.8380, and 0.8510, respectively, for this specific inline.

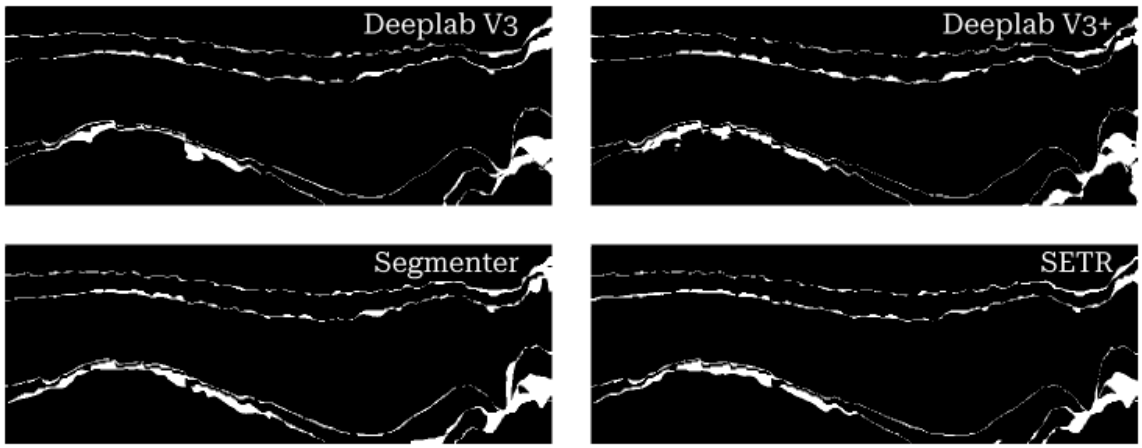


Figure 5.4: Difference between the model prediction and the ground truth for inline 109. Black pixels indicate correctly classified regions, while white pixels highlight misclassified areas. In this view we can see the error in the prediction borders and the major errors occurring in the area where four classes are very close to each other, the bottom right

5.2 Segmenting the Seam AI Dataset

Shifting our focus to the Seam AI dataset, it is evident that the models faced more challenges compared to the F3 dataset. All models experienced a drop in performance, with mIoU scores falling from the 0.77–0.78 range to the 0.70s. As shown in Figure 5.3, the SETR model significantly outperformed the others, achieving an mIoU of 0.76. Following SETR, Deeplab V3 scored 0.7129, highlighting the substantial gap between SETR and the rest. Next were Deeplab V3+, Segmenter, and SegFormer, with scores of 0.7095, 0.7012, and 0.6406, respectively. The SegFormer model, in particular, continued to underperform, even more so on the Seam AI dataset.

For the Seam AI dataset, we also performed a comparative analysis using 150 epochs, as shown in Table 5.4. Unlike the F3 dataset, the results here were less definitive. Apart

Model	mIoU	PA	MCA
Deeplab V3	0.7128	0.9236	0.833
Deeplab V3 +	0.7095	0.9211	0.8244
SegFormer	0.6406	0.8874	0.712
Segmenter	0.7012	0.9167	0.8102
SETR	**0.7605**	0.9377	0.8407

Table 5.3: Models' performance on Seam AI dataset when trained with 1000 epochs. In bold the best mIoU between the models.

from SegFormer, which gained 10% in performance, the other models showed minimal improvement. Additionally, the models reached their peak performance much earlier than in the F3 dataset, with only the SETR model surpassing the 900th epoch mark. Nonetheless, most models still achieved their best performance closer to the 1,000th epoch than the 150th. The recommendation remains the same as with the F3 dataset: training for only 150 epochs may limit performance, especially for larger models, so allowing more training time could be beneficial.

Model	150	1000	Gain	Best Val Epoch
Deeplab V3	0.7047	0.7128	1,15%	620th
Deeplab V3 +	0.7081	0.7095	0.20%	550th
SegFormer	0.5807	0.6406	**10.32%**	800th
Segmenter	0.672	0.7012	4,35%	770th
SETR	**0.7529**	**0.7605**	1,01%	910th

Table 5.4: mIoU gain on Seam AI dataset when increasing the training regimen from 150 to 1000 epochs. In bold the best mIoU between the models and the biggest gain in performance.

Similar to the F3 dataset, we selected two samples to showcase the qualitative performance of the models on the Seam AI dataset. For this analysis, we chose crossline 11 as an example of lower-end model performance and crossline 182 to represent the upper end of performance.

In Figure 5.5, we present the predictions for crossline 11 alongside its ground truth. In this example, the models struggled to segment the Mass Transport Deposit layer, shown in yellow, frequently misclassifying it as the Slope Mudstone B layer, shown in dark blue, as we can see in Figure 5.6. As seen in the F3 dataset, the models tend to underperform in regions with faults, and this issue is clearly evident here due to the presence of a significant fault in the center of the prediction area.

Among the models, SETR delivered the best performance according to the metrics, achieving an mIoU of 0.6713. Upon examining its prediction, SETR produced a generally acceptable segmentation, particularly accurate in predicting the Slope Valley facies located in the center-left of the prediction, shown better in Figure 5.7. However, it exhibited larger errors in segmenting the Mass Transport Deposit facies.

In contrast, the SegFormer model, which had the worst performance, displayed significant classification errors across multiple facies. The Slope Mudstone B, Submarine

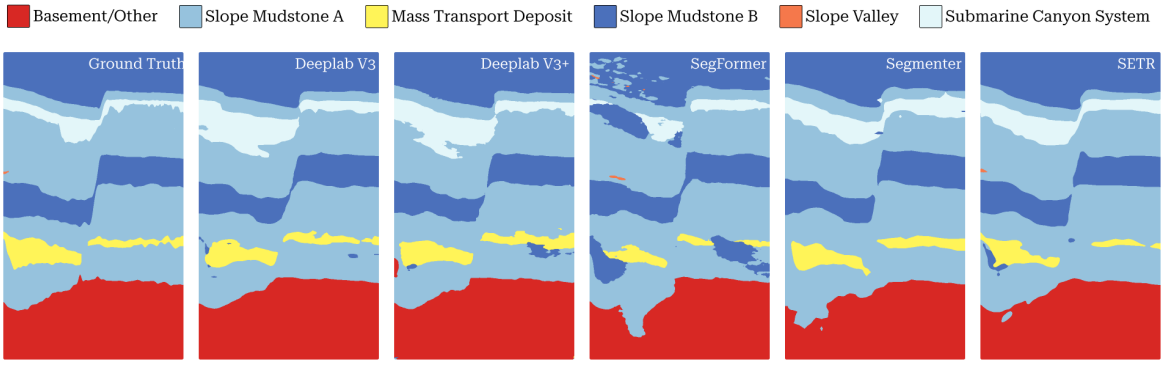


Figure 5.5: Segmentation of crossline number 11 from the test split of the Seam AI dataset, comparing the performance of various models against the ground truth. The models, DeepLab V3, DeepLab V3+, SegFormer, Segmenter, and SETR, achieved scores of 0.6151, 0.6314, 0.5429, 0.6127, and 0.6713, respectively, for this specific inline.

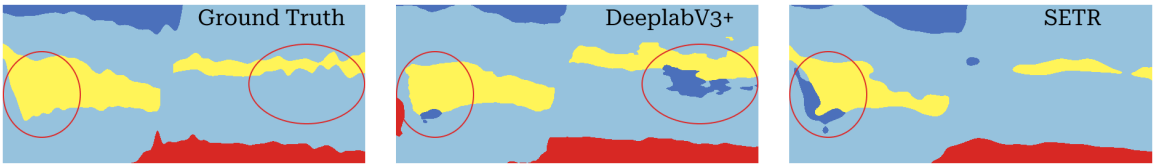


Figure 5.6: Errors on the Mass Transport Deposit layer, exemplified with the SETR and Deeplab V3+ models, the top two performing for crossline 11.

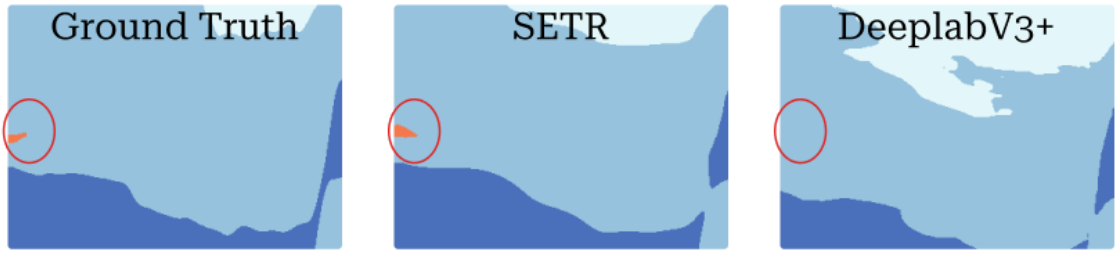


Figure 5.7: Prediction of the Slope Valley class, exemplified with the SETR and Deeplab V3+ models, the top two performing for crossline 11. While the SETR prediction of the Slope Valley class is flawed, it achieves some success in segmenting it, whereas the Deeplab V3+ model was not capable of segmenting it.

Canyon System, Mass Transport Deposit, and Basement/Other layers, representing more than half of the total classes, showed major misclassifications, as indicated in Figure 5.8. Notably, the Mass Transport Deposit facies were almost entirely misclassified, highlighting a substantial flaw in SegFormer's ability to accurately segment this crossline.

Shifting focus to crossline 182, shown in Figure 5.9, we can see that the models performed much better in this scenario, and the metrics for each model reflect this improvement. While SegFormer still exhibited major segmentation problems, the other models

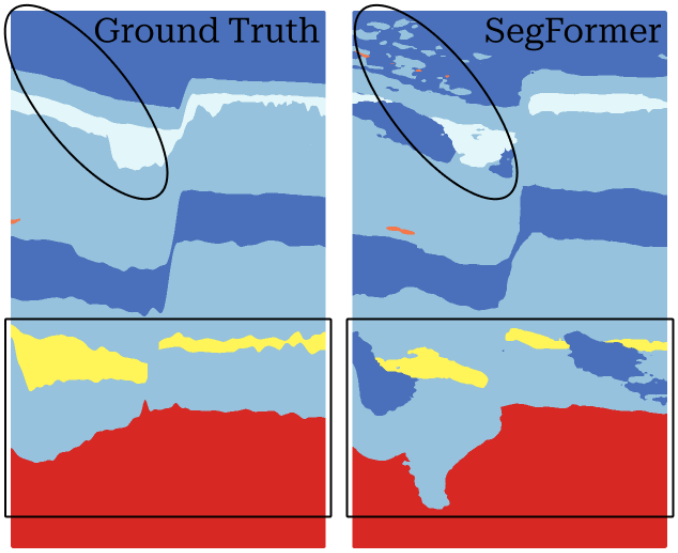


Figure 5.8: Compassion between the SegFormer prediction and ground truth, highlighting major prediction errors.

performed significantly better. Interestingly, no model was able to fully connect both sides of the Slope Mudstone B, shown in Figure 5.10, and Submarine Canyon System layers, indicating a potential limitation in their ability to segment with high detail. Once again, SETR produced the best segmentation, with an IoU of 0.8580 and a highly accurate result, showing no major errors and closely following the ground truth geometry across all facies.

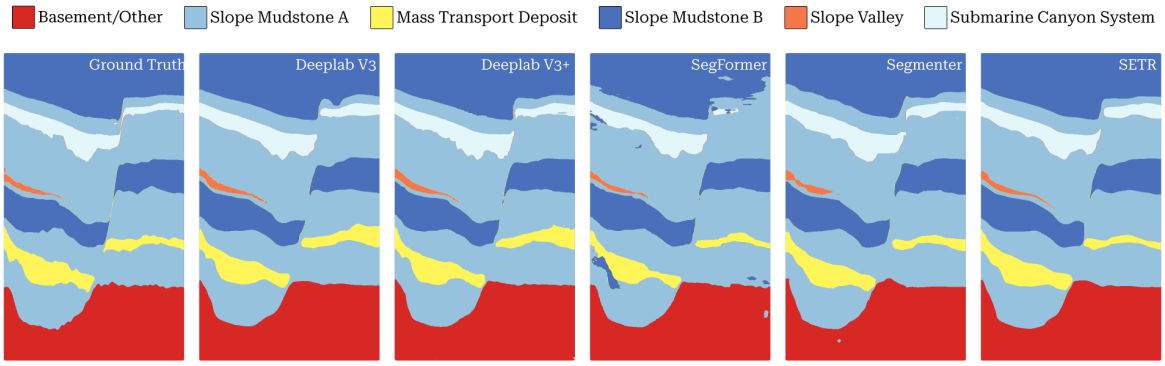


Figure 5.9: Segmentation of crossline number 182 from the test split of the Seam AI dataset, comparing the performance of various models against the ground truth. The models, DeepLab V3, DeepLab V3+, SegFormer, Segmenter, and SETR, achieved scores of 0.7887, 0.8024, 0.7708, 0.8058, and 0.8580, respectively, for this specific inline.

Crossline 182 offers an interesting comparison between the Deeplab V3 and SegFormer models, which achieved IoU scores of 0.7887 and 0.7708, respectively—a difference of only 0.0179. Despite their similar mIoU scores, the segmentation results reveal a clear qualitative difference, with Deeplab V3 outperforming SegFormer in terms of segmentation accuracy. Figure 5.11 shows the difference between the ground truth and each model's

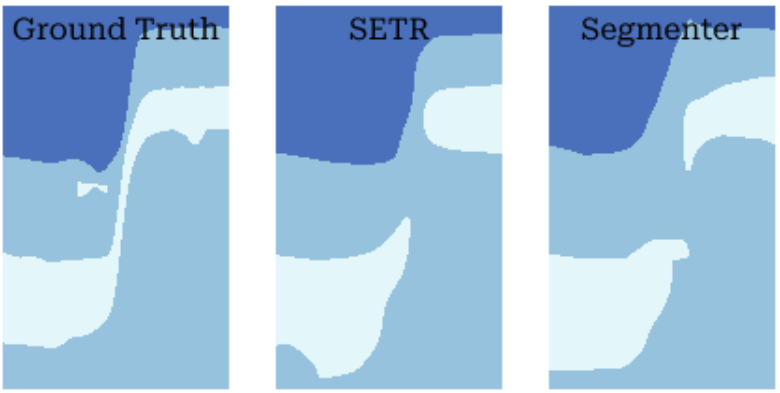


Figure 5.10: Error in the segmentation of the Submarine Canyon System class. This area is just a couple pixels wide, making the prediction harder to make.

prediction. While Deeplab V3 struggles somewhat to match the exact geometry of the ground truth, it generally succeeds in correctly classifying the facies, avoiding significant misclassifications where a region is labeled as one facies but clearly corresponds to another in the ground truth. This discrepancy highlights the importance of evaluating the actual segmentation outputs alongside quantitative metrics, as similar scores can mask meaningful differences in model performance.

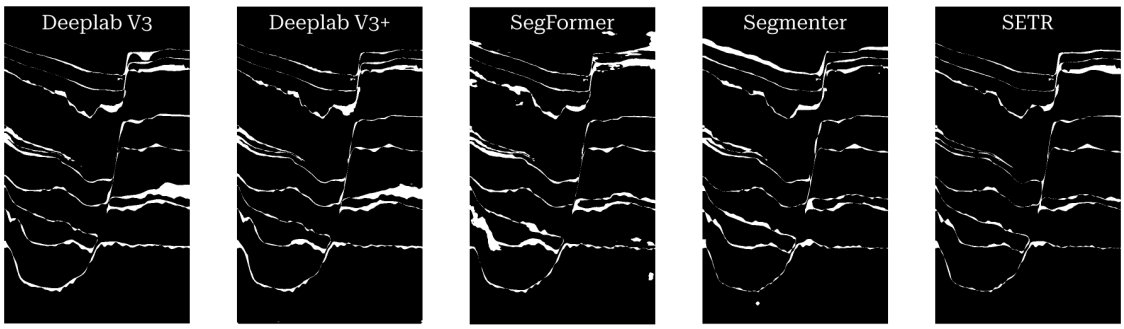


Figure 5.11: Difference between the model prediction and the ground truth for crossline 182. Black pixels indicate correctly classified regions, while white pixels highlight misclassified areas.

5.3 Comparing With the Third-party Results

To contextualize our findings and establish a reference point for our models, we compared our results with state-of-the-art results from the literature. For this comparison, we categorized existing works on the F3 and Seam AI datasets, analyzing their reported results in relation to our own findings. Our analysis focuses primarily on quantitative metrics, as we do not have access to the predictions from other works to conduct a

qualitative evaluation.

5.3.1 F3 Dataset

For the F3 dataset, we compared our results with those reported by Alaudah *et al.* [2], Trindade and Roisenberg [28], Abid *et al.* [1], Tolstaya and Egorov [27], and Wang *et al.* [30]. The models in these works employed more sophisticated training techniques and optimizations compared to our experimental setup, such as incorporating fault generation and depth information. Table 5.5 presents the results for the models proposed in those papers alongside our results.

It is also important to note that Tolstaya's best models were trained using additional labels acquired through a pseudo-labeling technique. Since we did not have access to this extra data, nor did we employ similar techniques, we will not compare our results directly with these models. However, they are still included in the table for reference.

We will also not compare our results with those of Abid *et al.* [1] due to differences in their data-splitting methodology, which we find problematic. In their work, Abid *et al.* [1] randomly split the dataset into 60% training, 20% validation, and 20% test sets. This random splitting raises concerns because the difference between adjacent inlines and crosslines is minimal. If panels in the training set are very close to those in the test set, data contamination can occur, potentially undermining the validity of the results.

Our data shows that the top-performing models, SETR and Segmenter, closely align with Tolstaya's best-performing models, with only a 0.01 difference between the four models. Given that our models lacked hyperparameter tuning, this result suggests that ViT-based models hold significant potential for achieving cutting-edge performance in segmenting the F3 dataset. Further optimization of these transformer models should be considered, including hyperparameter tuning.

Interestingly, despite having the worst performance among our models, the SegFormer model still outperformed the U-SegFormer model from Wang *et al.* [30]. We attribute this to the pre-training with the ImageNet21k dataset, which, due to its vast and varied data, likely gave our models a significant advantage in training.

5.3.2 Seam AI Dataset

For the Seam AI dataset, we compared our results with those reported by Tolstaya and Egorov [27], Sheng *et al.* [24], Guo *et al.* [12] and Kaur *et al.* [18]. Since Kaur reported their results using different metrics than those used in Tolstaya's, Sheng's, Guo's and our work, we present Kaur's results separately. In Table 5.6, we compare Tolstaya's, Sheng's and Gou's results with our own, focusing on metrics such as mIoU, pixel accuracy, mean class accuracy, and the mean F1 score where reported. The inclusion of the F1 score serves as a bridge to Table 5.7, where we compare Kaur's results using their reported metrics: F1 score, mean precision, and mean recall.

As with the F3 dataset, we compare our models only to those trained using the same data. Consequently, models from Tolstaya's work that used additional or modified label data, such as the Aug + Pseudolabels and Aug + Relabeling models, are included in

Model	mIoU	PA	MCA
CNN section + aug [2]	—	0.901	0.804
CNN section + aug + skip [2]	—	0.905	0.817
AH-Ensemble (Seismic and depth) [28]	—	0.893	0.785
Ensemble [1][2]	0.9392	0.9852	0.9688
UNET test set 1 (no aug) [27]	**0.79**	0.94	0.88
UNET (aug) [27]	**0.79**	0.94	0.91
UNET (pseudolabels) [27][1]	0.85	0.94	0.95
U-Segformer-Hyper Section-Based [30]	—	0.907	0.852
U-Segformer-Hyper Patch-Based [30]	—	0.897	0.761
SETR PUP Seismic [4]	0.7753	0.938	0.877
SETR PUP IN21K [4]	0.7836	0.94	0.879
Deeplab V3 (Ours)	0.7781	0.9392	0.8742
Deeplab V3 + (Ours)	0.7748	0.9365	0.8649
SegFormer (Ours)	0.7268	0.9182	0.8198
Segmenter (Ours)	0.7800	0.9353	0.8815
SETR (Ours)	**0.7849**	0.9338	0.8828

1 - Results produced with extra labels
2 - Results produced with an improper data split

Table 5.5: F3 results reported by Alaudah *et al.* [2] (CNN), Trindade and Roisenberg [28] (AH-Ensemble), Abid *et al.* [1] (Ensemble), Tolstaya and Egorov [27] (UNET), Miranda and Gattass [4] (SETR PUP), and Wang *et al.* [30] (U-Segformer) vs our results.

Table 5.6 for reference but are not directly compared against our results.

Similar to the findings with the F3 dataset, our models performed closely to those reported by Tolstaya and Egorov [27], with the SETR model showing just a 0.01 difference compared to Tolstaya's best models, which achieved an mIoU score of 0.77. This pattern mirrors the findings from the F3 dataset. Given that our models were not fully optimized, further refinement of hyperparameters and training techniques could potentially enhance the SETR model's performance.

The work by Sheng *et al.* [24] demonstrates notable improvements in performance, especially with their SFM Fine-tune 512 model, which is also based on the ViT architecture. This model achieved the highest mIoU score, 0.798, among the three works compared. These results suggest that robust training strategies, such as leveraging Semi-Supervised Learning (SSL), can significantly improve model performance.

Our methodology relied on pre-trained weights from models trained on the ImageNet 21K dataset, which contains conventional photographic data. In contrast, Sheng's approach involved pre-training on a large seismic dataset, indicating that domain-specific pre-training may offer advantages. Interestingly, while Miranda and Gattass [4] also employed SSL techniques, models initialized with ImageNet weights outperformed those pre-trained on seismic data on the F3 dataset, as indicated in Table 5.5. This difference may arise from the relatively smaller or less diverse seismic dataset used in the pre-training phase, highlighting the importance of large, high-quality, and diverse datasets for effective pre-training.

Model	mIoU	PA	MCA	MF1
UNET Aug [27]	**0.77**	0.94	0.94	0.86
UNET Aug + ×384 patches [27]	**0.77**	0.94	0.94	0.85
UNET Aug + pseudolabels [27][1]	0.72	0.93	0.96	0.81
UNET Aug + relabeling [27] [1]	0.77	0.94	0.93	0.86
SFM Frozen [24]	0.6417	0.9148	—	—
SFM Fine-tune [24]	0.7294	0.9377	—	—
SFM Fine-tune 512 [24]	**0.7980**	0.9430	—	—
DINOv2-PUP [12]	0.6885	0.9102	—	—
Deeplab V3 (Our)	0.7128	0.9236	0.833	0.8132
Deeplab V3 + (Our)	0.7095	0.9211	0.8244	0.8118
SegFormer (Our)	0.6406	0.8874	0.712	0.757
Segmenter (Our)	0.7012	0.9167	0.8102	0.8017
SETR (Our)	**0.7605**	0.9377	0.8407	0.8531

Results produced with extra labels - 1

Table 5.6: Seam AI results reported by Tolstaya and Egorov [27] (UNET), Sheng *et al.* [24] (SFM), and Guo *et al.* [12] (DINOv2-PUP) vs our results.

For the comparison with Kaur *et al.* [18], who did not report mIoU, we calculated the mean F1 score, mean precision, and mean recall for our models to align with the metrics used in their study. It is important to note that our models were optimized with a focus on IoU rather than F1 score, which could lead to differences in performance evaluation based on these metrics.

Additionally, Kaur's work utilized the original Parihaka dataset with private labels provided by Chevron, which may differ from the dataset we used in this study. As a result, direct comparisons between our work and Kaur's must be interpreted with caution due to potential discrepancies in the datasets.

Table 5.7 presents Kaur's reported results alongside our own, providing insight into how our models perform under alternative evaluation metrics and different data configurations.

Model	MF1	MP	MR
Deeplab V3+ [18]	**0.9608**	0.9606	0.8286
GAN [18]	0.9431	0.9281	0.9621
Deeplab V3 (Our)	0.8132	0.8017	0.833
Deeplab V3 + (Our)	0.8118	0.8056	0.8244
SegFormer (Our)	0.757	0.8326	0.712
Segmenter (Our)	0.8017	0.7977	0.8102
SETR (Our)	**0.8531**	0.8725	0.8407

Table 5.7: The Seam AI results reported by Kaur *et al.* [18] are compared to our results in this study. Since Kaur *et al.* [18] did not provide mean metrics, we calculated the mean values based on the data available in their paper to ensure a consistent comparison. This approach allowed us to align the reported metrics, facilitating a more accurate assessment of model performance across studies.

Our best models fall significantly behind when compared with the results reported by

Kaur *et al.* [18], with a gap of approximately 0.11 in F1 score between our top-performing model, SETR, and Kaur's Deeplab V3+. However, a more telling comparison is between our unoptimized Deeplab V3+ and Kaur's Deeplab V3+. Since both models share the same architecture, we conjecture that the performance gap likely stems from differences in the dataset and model optimization techniques.

Chapter 6

Tuning SETR for Semantic Segmentation

After obtaining the previous results, we selected our best-performing model, SETR, for a hyperparameter search. Conducting the hyperparameter search proved to be a challenging task, primarily because the framework used for our experiments, MMsegmentation, does not natively support this functionality. Additionally, addressing an internal demand for greater flexibility and extensibility, we decided to develop a new framework tailored to our specific requirements and capable of executing the search.

This framework, named Minerva[1], was designed to facilitate efficient training and experimentation with machine learning models while accommodating features such as hyperparameter optimization. In this section, we will briefly discuss the development of Minerva in relation to its relevance to the topic at hand. A detailed account of its architecture, design decisions, and the challenges encountered during its implementation is provided in Appendix .1.

Developing all the features required for this part of the work took over a year, significantly exceeding initial expectations. Despite the extended timeline, the Minerva framework did not achieve the feature parity with the MMsegmentation framework required for conducting the hyperparameter search effectively. Consequently, the hyperparameter search had to be conducted with parameters that differed from those used in MMsegmentation.

This section discusses our findings, highlights the differences between the methods employed with Minerva and MMsegmentation, and examines the shortcomings encountered during the hyperparameter optimization process.

6.1 Methodology

To provide clarity, we outline the method employed for the hyperparameter search and examine the differences between this process and the training conducted using the MMsegmentation framework. Here are laid out the parameters of the search, their shortcomings, and how they differ from what was used to train the models initially.

[1]Minerva's code base is available in `https://github.com/discovery-unicamp/Minerva`

The results presented were obtained using the following pipeline. First, a hyperparameter search was conducted using our custom framework. The parameters used in the search will be described in subsequent sections. After a 12-day search, the best trial was selected based on the validation mIoU. Following that, the models were trained using the same methodology employed for the comparisons reported earlier, utilizing the MMSegmentation framework.

6.1.1 Software and Tools

As briefly mentioned before, for the hyperparameter search step of the work, we switched from MMsegmentation to our in-house framework, Minerva. Minerva is primarily built on top of PyTorch and PyTorch Lightning, a lightweight, high-level framework that simplifies training, scaling, and managing PyTorch models by abstracting boilerplate code.

For the hyperparameter search step specifically, we employed Ray Tune in conjunction with the Hyperopt library. Ray Tune is a scalable framework for hyperparameter tuning and experiment management, supporting distributed and efficient optimization methods, while Hyperopt is a Python library for optimizing hyperparameters using algorithms like random search, TPE, and annealing in a flexible, parallelizable manner. The integration of Hyperopt with Ray Tune uses the Tree-structured Parzen Estimator (TPE) as its acquisition function. We still use MMsegmentation for the final traning of the models.

6.1.2 Search Parameters

For this search, seven parameters were tested. The details of these parameters, including their types, value ranges, and the specific optimizers to which they were applied, are presented in Table 6.1. This table provides a comprehensive overview of the tested parameters, enabling a clear understanding of the experimental setup.

Range types could either be uniform, allowing the optimizer to select any number within a specified range, or choice, where the optimizer picks from predefined values. A third type, independent, means that a separate search was conducted for each listed value. Only the optimizer type parameter utilized the independent range type. This approach was necessary because certain parameters applied exclusively to specific optimizers, and we lacked a mechanism to conditionally include or exclude parameters during the search. The Optimizer column in Table 6.1 indicates the specific optimizers associated with each parameter.

In addition to the hyperparameters under consideration, the models were configured with fixed parameters that remained constant across all trials. These locked parameters include an input size of 256x576 and a number of classes set to 6. These settings were chosen once they were required for the model architecture. The resolution of 256x576 was selected to align with the size used in the pretrained weights obtained from the MMsegmentation framework. This consistency ensured compatibility with the pretrained model configurations.

All trials were conducted with a grace period of 50 epochs, a patience of 5 epochs, and a standard deviation threshold of 0.01 to determine if a trial had plateaued. The total

Hyperparameter	Range Type	Values	Optimizer
Decoder Dropout	uniform	0.01 – 0.2	All
Encoder Dropout	uniform	0.01 – 0.2	All
Aux Heads Weights	choice	0.2, 0.3	All
Freeze Backbone	choice	True, False	All
Learning Rate	uniform	$1e^{-4}$ – $1e^{-5}$	All
Optimizers	independent	Adam, AdamW, RAdam, SGD	–
Weight Decay	uniform	0.01 – 0.001	AdamW, SGD
Momentum	uniform	0.9 – 0.95	SGD

Table 6.1: List of hyperparameters searched, their value range and in what optimizers they were employed.

duration for the search process was slightly over 12 days, totaling 1,469 trials executed. The hyperparameter optimization employed the HyperOpt search algorithm, executed using Ray Tune to manage and distribute the search process effectively.

After identifying the best parameters, we applied them within the MMsegmentation framework to train the models in a manner consistent with our initial evaluations. This approach ensured comparability between the results obtained from the hyperparameter search and the earlier benchmarks.

6.1.3 Data Augmentation and Transformations

For data augmentation, we applied only random flipping along both axes during training. Regarding transformations, since the dataset has a maximum size of 255x701, we cropped the data to 256x576 and padded it using reflection when necessary. This input size was chosen to align with the input size in which our pretrained weights were trained.

6.1.4 Differences between MMsegemntation Pipeline and Hypersearch Pipeline

This section discusses the differences between the hyperparameter search pipeline and the MMsegmentation pipeline. We outline the rationale behind the chosen settings for the search, analyze their implications, and explore how these differences might impact the final results.

The most basic difference between the two pipelines lies in the PyTorch version used. Minerva operates on the newer PyTorch 2.1, while MMsegmentation relies on version 1.1. Although the core implementations of the features we utilized remain consistent across versions, changes in other parts of the PyTorch framework could potentially introduce differences in the results. These variations might stem from updates in numerical precision, optimizations, or subtle behavior shifts in the framework over time.

The second set of differences pertains to the model architecture and can be divided into two main aspects. The first is the expected input size. During our search, we used an input size of 256x576 to ensure compatibility with the pre-trained weights, as detailed in the previous section. In contrast, the MMsegmentation pipeline utilized an input

size of 512x512. MMsegmentation accommodates deviations from the fixed input size by interpolating the new positional encodings to align with those from the pre-trained weights. Unfortunately, this interpolation capability was not implemented in Minerva during the execution of our search.

Due to the input size constraints, conducting a hyperparameter search for the Seam AI dataset was not feasible. The Seam AI dataset features images sized 1006x590 for crosslines and 1006x531 for inlines. Cropping these images to 256x576 would result in a significant loss of information. In contrast, the F3 dataset has a maximum size of 255x701, which means that only 125 pixels in length would be lost for inline examples, while crosslines, sized 255x401, naturally fit the expected input size. Because of that we only performed the search on the F3 dataset.

The second architectural difference relates to the use of distinct learning rates for the backbone and prediction head of the model. In the MMsegmentation pipeline, the prediction head is assigned a larger learning rate to preserve the knowledge encoded in the pre-trained weights, minimizing unnecessary alterations during training. This feature, which enables finer control over the learning rates of different model components, was unavailable in Minerva at the time of our experiments.

There are also differences in the data transformation pipeline, stemming from the disparity in input sizes. In our search, we applied three transformations: cropping, padding, and random flipping. This transformation pipeline was tailored to fit our input size of 256x576. Specifically, during training, the model could receive either a 255x701 or 255x401 image, depending on whether the input was an inline or crossline. To address this variability, we cropped the image to 256x576, padded it with reflections to fill any gaps, and applied a random flip as the final step.

In contrast, the MMsegmentation pipeline requires a different approach due to its larger input size of 512x512. During training, images are scaled so that their smaller dimension equals 512 pixels. Next, the image is cropped to the final size of 512x512, with the cropping process constrained to ensure that the majority class occupies no more than 75% of the cropped area. Finally, a random flip is applied. This pipeline accommodates the larger input size and introduces a constraint aimed at balancing the class distribution within each crop.

6.2 Results

With the differences between the two setups clarified, we can now discuss the results obtained from the hyperparameter search. As previously mentioned, the search was divided into four runs, one for each optimizer tested, to avoid a circular dependency issue in our setup. A total of 1,469 trials were conducted across all four searches, culminating in the results presented in Table 6.2.

After completing the trials, the selected parameters were applied to train the models within the MMsegmentation framework. Table 6.3 presents the results achieved for each optimizer alongside the baseline results from the previous experiments, allowing for a direct comparison of performance improvements.

Optimizer	DD	ED	AHW	FB	LR	WD	M	V-mIoU
Adam	0.0172	0.1190	0.2	False	$2.01178e^{-5}$	–	–	0.9466
AdamW	0.0548	0.0169	0.3	False	$4.96475e^{-5}$	0.0020	–	0.9343
RAdam	0.0223	0.1238	0.3	False	$2.70224e^{-5}$	–	–	0.9458
SDG	0.0354	0.0972	0.2	False	$2.03077e^{-4}$	0.0011	0.9354	0.8652

Table 6.2: Best Results from trials. The table lists the following parameters: Decoder Dropout (DD), Encoder Dropout (ED), Auxiliary Heads Weights (AHW), Freeze Backbone (FB), Learning Rate (LR), Weight Decay (WD), Momentum (M), and Validation mIoU (V-mIoU).

Model	mIoU	PA	MCA
SETR - Adam	0.7835	0.9372	**0.8855**
SETR - AdamW	**0.7875**	**0.9373**	0.8845
SETR - RAdam	0.7704	0.9306	0.8734
SETR - SGD	0.7629	0.9320	0.8689
SETR - Baseline	0.7849	0.9338	0.8828

Table 6.3: Final results after training using the MMsegmentation framework with the hyperparameters obtained from the search. In bold the best result for each metric.

As observed, the hyperparameter search did not yield significant improvements, with the best-performing models being functionally equivalent to the baseline. For instance, comparing the SGD-based model from the search to the baseline, which also utilizes the SGD optimizer, there is an approximate 3% reduction in performance. This discrepancy suggests that the search process may not have been an effective proxy for the actual training conditions employed in the baseline models.

Another possibility is that the models may have suffered from overfitting during the search phase, which would negatively impact their performance on the test set. When analyzing the validation mIoU of the models, we observe that it is very close to 1. It is important to note that our validation set consists of randomly selected inlines and crosslines from the training set. Given that neighboring inlines and crosslines are highly similar, the likelihood of overfitting increases significantly in this setup.

In conclusion, our hyperparameter search did not yield a parameter set capable of significantly surpassing the baseline model, rendering the results inconclusive. Further experiments are necessary to address the limitations encountered in this search. Incorporating novel training methods, such as LoRA, advanced data transformations like fault simulation, and improvements in prediction techniques, such as patch-based predictions, could provide a clearer understanding of the performance potential of SETR and transformer-based models in the semantic segmentation of seismic facies.

Chapter 7

Conclusions

This work investigates the potential of transformer architectures for semantic segmentation of seismic facies, comparing the performance of two CNN-based models with three prominent transformer models. Our findings reveal that transformers demonstrate promising performance compared to CNN models. Furthermore, extending the training duration significantly improved model performance; models trained for 1000 epochs consistently outperformed those trained for only 150 epochs. Data quality and quantity are crucial factors, especially for large models such as those based on the ViT-L architecture. For that reason, creating and annotating larger, high-quality datasets would greatly benefit the research community.

Our qualitative analysis showed that models generally struggled with segmenting facies containing faults. Techniques like fault simulation, as used by Tolstaya and Egorov [27], could potentially enhance model performance in this challenging area. While transformers demonstrated superior performance overall, CNN models remained competitive, highlighting the need for further research to comprehensively evaluate the potential of transformer architectures in the semantic segmentation of seismic facies.

Our hyperparameter search did not yield conclusive results, highlighting the need to enhance the search pipeline to obtain more reliable outcomes. Additionally, incorporating new techniques into the pipeline and evaluating their impact on model performance will be essential for further improvements.

Finally, our study aligns with the findings of Sheng *et al.* [24], Bosco de Miranda and Gattass [4], and Guo *et al.* [12], demonstrating that transformers generally outperform CNNs in semantic segmentation of seismic facies.

From this work the paper "Applying the Transformer Architecture for Semantic Segmentation of Seismic Facies" [13] was published and presented at the 85th EAGE Annual Conference & Exhibition and a full paper is under review.

7.1 Challenges and Solutions

During the execution of this work, a series of hurdles were encountered and overcome. In this section, we outline the solutions employed and discuss unresolved challenges, aiming to provide valuable insights for future studies. This approach not only shares the data

collected but also conveys the experience gained throughout the project. Particularly in the seismic domain, research often lacks detailed technical descriptions of experimental setups and execution, a gap this section seeks to address.

The first challenge we faced was with the MMsegmentation framework. While MMsegmentation is an excellent tool for quickly initiating experiments due to its user-friendly interface, it presents significant difficulties when attempting to modify the code or introduce new functionalities. This is primarily due to its numerous and often convoluted software abstractions, which make it challenging to navigate and adapt. Furthermore, these abstractions complicate the process of verifying how certain operations are executed in the code.

For this reason, we cannot recommend the use of MMsegmentation for research requiring extensive control over the software and its features. Its design prioritizes usability for standard tasks but falls short when flexibility and detailed customization are needed.

Another significant challenge we encountered was also tied to the high degree of coupling in MMsegmentation's codebase. When we required a feature not supported by MMsegmentation, hyperparameter searching in our case, we had to extract and adapt code that was tightly integrated into MMsegmentation. This was particularly true for SETR, which was originally implemented directly within the MMsegmentation framework and lacked a standalone implementation.

The process of porting SETR from MMsegmentation to our framework, Minerva, involved over four months of iterations. This included researching the intricacies of the model's functionality within MMsegmentation, implementing the model in our framework, and rigorously validating its correctness.

Other features presented similar challenges. Implementing functionalities such as assigning different learning rates for the backbone and prediction head, integrating specific data transformations, and supporting positional encoding interpolation all required significant effort. These issues further contributed to the extended development timeline of Minerva, as each feature demanded careful analysis, adaptation, and validation to ensure parity with MMsegmentation's capabilities.

Using MMsegmentation's weights outside its framework also posed significant challenges, as the layer names did not align with standard PyTorch implementations. To address this, we developed a translation script that read the weights dictionary and adjusted the layer names to match the expected format. This additional step introduced complexity and increased the effort required to integrate pre-trained weights into our framework.

Another challenge we faced was checkpointing during the hyperparameter search. Training a relatively large model while running numerous trials quickly led to storage issues, with even basic searches exceeding one terabyte of disk space. To address this, we developed custom hooks to control the frequency of checkpointing, optimizing storage usage.

While resolving this, we encountered another issue related to Ray, the framework used for hyperparameter tuning. Ray's documentation uses the same terminology for both model checkpointing and search checkpointing, which caused confusion when configuring these processes. This lack of clear distinction added unnecessary complexity to the setup.

Lastly, we faced significant challenges related to computational power. The most glaring limitation was the batch size restriction. Even with access to a high-performance supercomputer, we could only use a batch size of 2 due to GPU memory constraints. A larger batch size could potentially improve training stability and performance, but this was not feasible within our setup.

Another issue arose when the supercomputer was unavailable, which severely hindered our progress. The alternative computational resources at our disposal were either too slow or lacked support for multi-GPU setups, rendering our research nearly unviable during these periods.

7.2 Future Work

A significant amount of work remains to be done. The natural next step is addressing the issues encountered with the hyperparameter search setup. Achieving feature parity with MMsegmentation would enable a more accurate comparison with baseline results. Additionally, exploring alternative setups, such as varying input sizes and model architectures, as well as expanding the search space, could lead to more comprehensive and effective hyperparameter searches.

Implementing state-of-the-art techniques, such as fault simulation, alternative loss metrics, and patch-based prediction supported by depth information, would be highly beneficial. Incorporating novel training approaches like LoRA and other self-supervised learning (SSL) methods also holds promise for improving the results achieved in this study. Lastly, standardizing dataset splits would have a significant impact on the field. The current prevalence of studies using the same dataset but with varying data divisions complicates the comparison of results, and establishing standardized splits would greatly enhance consistency and reproducibility.

Some potential sources of bias also need to be addressed. This work's methodology is based on the approach proposed by Bosco de Miranda and Gattass [4]. Consequently, we adopted certain arbitrary techniques, such as the weighted cross-entropy loss and the transformation pipeline. Further ablation studies are necessary to validate the effectiveness of these techniques and to explore alternative options for both the loss function and the transformation methods.

Bibliography

[1] Bilal Abid, Bilal Muhammad Khan, and Rashida Ali Memon. Seismic facies segmentation using ensemble of convolutional neural networks. *Wireless communications and mobile computing*, 2022(1):7762543, 2022.

[2] Yazeed Alaudah, Patrycja Michałowicz, Motaz Alfarraj, and Ghassan AlRegib. A machine-learning benchmark for facies classification. *Interpretation*, 7(3):SE175–SE187, 2019.

[3] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[4] Daniel Cesar Bosco de Miranda and Marcelo Gattass. Vision transformers e masked autoencoders para segmentação de fácies sísmicas. Master's thesis, Pontifical Catholic University of Rio de Janeiro, 2023.

[5] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[6] Paulo Caroli. *Lean Inception: How to Align People and Build the Right Product*. Editora Caroli, 2019.

[7] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.

[9] SEG Advanced Modeling Corporation. Facies identification challenge. https://www.aicrowd.com/challenges/seismic-facies-identification-challenge, accessed 27 October 2023.

[10] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] Zhixiang Guo, Xinming Wu, Luming Liang, Hanlin Sheng, Nuo Chen, and Zhengfa Bi. Cross-domain foundation model adaptation: Pioneering computer vision models for geophysical data analysis. *arXiv preprint arXiv:2408.12396*, 2024.

[13] Gabriel Gutierrez, Carlos Astudillo, Otávio Napoli, Daniel Miranda, Alan Souza, João Paulo Navarro, Leandro Villas, and Edison Borin. Applying the transformer architecture for semantic segmentation of seismic facies. In *85th EAGE Annual Conference & Exhibition (including the Workshop Programme)*, volume 2024, pages 1–5. European Association of Geoscientists & Engineers, 2024.

[14] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[16] Frank Hutter, Matthias Feurer, and Joaquin Vanschoren. *Automatic machine learning: Methods, systems, challenges*. SPRINGER NATURE, 2018.

[17] Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB)*, pages 1–7. IEEE, 2020.

[18] Harpreet Kaur, Nam Pham, Sergey Fomel, Zhicheng Geng, Luke Decker, Ben Gremillion, Michael Jervis, Ray Abma, and Shuang Gao. A deep learning framework for seismic facies classification. *Interpretation*, 11(1):T107–T116, 2023.

[19] Valliappa Lakshmanan, Martin Gorner, and Ryan Gillard. *Practical machine learning for computer vision: End-to-end machine learning for images*. O'Reilly Media Inc, 2021.

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[21] Emilio Soria Olivas, Jos David Mart Guerrero, Marcelino Martinez-Sober, Jose Rafael Magdalena-Benedito, L Serrano, et al. *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI global, 2009.

[22] Alec Radford. Improving language understanding by generative pre-training. *OpenAI*, 2018.

[23] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*, 2021.

[24] Hanlin Sheng, Xinming Wu, Xu Si, Jintao Li, Sibio Zhang, and Xudong Duan. Seismic foundation model (sfm): a new generation deep learning model in geophysics. *arXiv preprint arXiv:2309.02791*, 2023.

[25] Reinaldo Mozart Silva, Lais Baroni, Rodrigo S Ferreira, Daniel Civitarese, Daniela Szwarcman, and Emilio Vital Brazil. Netherlands dataset: A new public dataset for machine learning in seismic interpretation. *arXiv preprint arXiv:1904.00770*, 2019.

[26] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. *arXiv preprint arXiv:2105.05633*, 2021.

[27] Ekaterina Tolstaya and Anton Egorov. Deep learning for automated seismic facies classification. *Interpretation*, 10(2):SC31–SC40, 2022.

[28] Elton Alves Trindade and Mauro Roisenberg. Multi-view 3d seismic facies classifier. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1003–1011, 2021.

[29] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[30] Zhiguo Wang, Qiannan Wang, Yang Yang, Naihao Liu, Yumin Chen, and Jinghuai Gao. Seismic facies segmentation via a segformer-based specific encoder–decoder–hypercolumns scheme. *IEEE Transactions on Geoscience and Remote Sensing*, 61:1–11, 2023.

[31] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in Neural Information Processing Systems*, 34:12077–12090, 2021.

[32] Guoqiang Xu and Bilal U Haq. Seismic facies analysis: Past, present and future. *Earth-Science Reviews*, 224:103876, 2022.

[33] Ozdogan Yilmaz. *Seismic Data Processing: Processing, inversion, and interpretation of Seismic Data*. SEG, 2008.

[34] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[35] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *CVPR*, 2021.

[36] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127:302–321, 2019.

Appendix A: Developing the Minerva Framework

Minerva is a machine learning framework designed to address the challenges commonly faced during research. It is built with a focus on modularity, ease of use, and robust functionality. This chapter explores the architectural philosophies underlying Minerva, the specific problems it aims to resolve, and the challenges encountered during its development.

Minerva originated from the need to perform a hyperparameter search with the SETR model, which was initially only available within the MMsegmentation framework. Although this requirement served as its starting point, Minerva's scope quickly expanded as the potential to create a robust framework addressing specific research needs became apparent.

There was also a demand to unify the tooling used by the discovery members so code could be interchangeable, easily reused and maintainable. All this factors culminated in the development of Minerva. Unifying the tools used by the research team, ensuring that code could be interchangeable, easily reused, and maintainable was also a great motivator.

Minerva is an open-source project that has benefited from contributions by members of both the Discovery Lab and H.IAAC at UNICAMP. The project is led by Gabriel B. Gutierrez, Otávio O. Napoli, and Professor Edson Borin, who oversee its architecture, maintenance, and quality, while shaping its overall vision and direction.

.2 Requirement Discovery

Requirement discovery is a critical phase in software engineering that is sometimes underestimated. It forms the foundation for creating high-quality software and ensures a smoother development process. For Minerva's requirement discovery, an adapted version of the Lean Inception method [6] was employed.

Several meetings were held to gather sufficient input to consolidate an architecture draft and define the requirements for a minimum viable product (MVP). From these discussions, a set of guiding principles emerged, shaping the direction of Minerva. First, a modular approach was identified as essential to facilitate testing of various configurations. Second, the framework needed to offer a straightforward and practical mechanism for executing experiments. Lastly, incorporating features to ensure reproducibility of experiments was deemed a critical requirement.

.3 Architecture

Based on the collected requirements and guiding principles, a macro-architecture was developed for Minerva. The design adopts a Pipeline-Centric Modular Architecture as its backbone. Minerva is built from several modules, each responsible for a specific functionality within the training process of machine learning models. These modules are designed to be highly self-contained, capable of operating independently of external components. Figure 1 shows a Container Diagram with all Minerva's sub modules and their interactions.

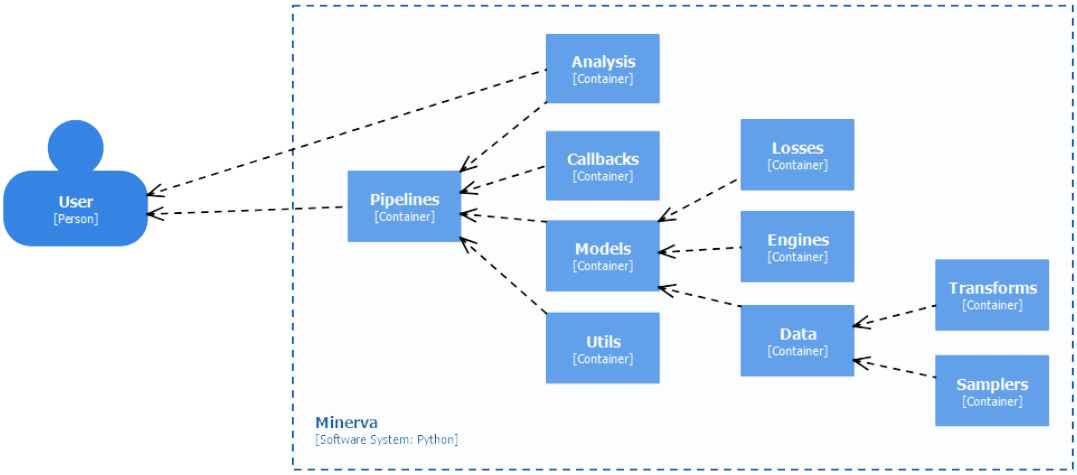


Figure 1: Minerva's Container Diagram from C4 class of diagrams.

The modules are integrated through pipelines, which combine the individual components into a cohesive workflow. Pipelines can also nest other pipelines, enabling code reuse to streamline the creation of new functionalities. This approach ensures that adding features or extending existing capabilities can be done efficiently and economically, without redundant effort.

Each module in Minerva may include sub-modules to further divide and organize its functionalities. This hierarchical structure ensures that complex features are broken down into manageable, reusable components, enhancing clarity and maintainability. By segmenting responsibilities into sub-modules, Minerva achieves a finer granularity of control, making it easier to extend or modify specific parts of a module without impacting the overall system.

After defining the broader architecture, it became necessary to detail the inner workings of each module, select the tools for framework construction, and determine how to leverage existing frameworks to implement the required feature set. For every module, a foundational interface was either built or adopted from existing frameworks. For instance, the losses and metrics modules and datasets sub-module utilize pre-existing interfaces from PyTorch and PyTorch Metrics as the foundation for their specialized classes. Conversely, for modules like transforms and pipelines, where no suitable existing interface was found within the dependencies, custom interfaces were designed and implemented to meet the framework's specific requirements.

Our models are designed with a two-layer structure. The lower layer is constructed using PyTorch's nn.Module and is responsible for defining the model's architecture. Within this layer, models are typically divided into a backbone and a projection head. This division facilitates the pretraining of models on different tasks or datasets that may require alternate projection heads.

These components are then integrated into a PyTorch Lightning LightningModule, which manages training, optimization, and related processes for training, testing, and validation. Users primarily interact with the LightningModule, while the underlying nn.Module components are handled internally, ensuring a streamlined and user-friendly experience.

Leveraging the PyTorch Lightning framework enables support for advanced training setups, such as multi-GPU training and distributed training across multiple nodes in a computing cluster. These capabilities, which are challenging to implement manually, are crucial for efficiently training large machine learning models. By integrating these features seamlessly, the framework ensures scalability and optimized resource utilization, making it ideal for research environments.

.4 Development

Developing Minerva has presented challenges on both the technical and management fronts. Our team's diversity, encompassing a wide range of educational backgrounds and varying levels of development experience, has added complexity to the process. Coordinating efforts and ensuring that all members are aligned with best practices, established processes, and development pipelines is an ongoing task. This responsibility creates an additional overhead for the core team, which must balance framework development with mentorship and quality assurance to maintain high standards of code production.

In this section, we will focus specifically on the development of features required for the research presented in this work. Although numerous additional features were implemented during this period, and the team gained substantial experience in both technical development and management, detailing these processes falls outside the scope of this work.

For this work, two key features were developed: the implementation of the SETR models and the creation of a hyperparameter search pipeline. Additionally, numerous supporting features were introduced, which will be discussed where relevant. All these features were developed entirely within the core team, which minimized the challenges typically associated with the management side of development.

.4.1 SETR

The SETR models were the first key feature implemented, as they served as the target model for the hyperparameter search. Consequently, it was essential to have them fully integrated into Minerva. However, the development process was significantly more complex than usual due to a unique challenge: the only available implementation was tied

to the MMSegmentation framework, and no standalone implementation in pure PyTorch was provided by the model's creators.

Being tied to MMSegmentation meant that we had to decouple the model's implementation from a codebase with an extraordinarily high degree of coupling. This presented a significant challenge, as the framework's tightly interwoven structure required careful isolation of the SETR model components while ensuring compatibility with Minerva's modular design principles. This process required over multiple months of meticulous analysis across three interconnected codebases: MMSegmentation, along with its supporting libraries, MMEngine and MMCV. At each step, we carefully examined the code to understand its functionality, its impact on the model as a whole, and its specific implementation details across the three codebases. We then determined the optimal strategy to decouple these components and integrate them into Minerva, ensuring compatibility with our framework while preserving the original model's integrity.

After successfully decoupling the SETR implementation, the next step was to determine the most effective way to integrate its functionality into Minerva. This involved designing and implementing the necessary modules to align with Minerva's architecture, ensuring modularity and compatibility. Once implemented, extensive testing was carried out to validate both the code and its functionality. This testing phase was critical to confirm that the model performed as expected within the Minerva framework and maintained its original capabilities.

The SETR-PUP model consists of a Vision Transformer (ViT) backbone and a Progressive Upscale Prediction (PUP) head. Since an implementation for the ViT backbone was already available in PyTorch, we used it as a base, making only minor modifications to meet the architectural requirements of SETR. The Progressive Upscale Prediction head, along with any additional supporting functionality required for the model to function correctly, was fully extracted from MMsegmentation and integrated into Minerva.

At the end of this process, we successfully developed a basic implementation of the SETR-PUP model within Minerva. Over time, as new requirements emerged, this implementation underwent significant enhancements, gradually evolving into the most feature-rich model currently available in Minerva. These additional capabilities and their development will be detailed in later sections.

.4.2 Hyperparameter Search

For hyperparameter search, we chose to leverage the capabilities of Ray Tune, a well-established tool known for its robust support for and distributed execution and PyTorch Lightning integration. These features were crucial for developing such a complex functionality. Without Ray, achieving the level of optimization and features present in our current implementation would not have been possible.

Given our initial lack of familiarity with Ray, we adopted a cautious, phased approach to development. We started by conducting small-scale experiments outside of Minerva to learn the fundamentals of Ray's operation. These experiments provided valuable insights into its functionality and how to effectively integrate it with PyTorch Lightning. This iterative process not only improved our understanding of Ray but also equipped us with

the knowledge necessary to design a seamless integration into Minerva.

During these experiments, we discovered that our SETR implementation was missing a key feature required for compatibility with Ray: the ability to instantiate models by passing all parameters via a dictionary, rather than explicitly through the constructor. Additional support was also added for parameterizing the optimizer and weighted loss.

After completing the small-scale experiments, we began implementing the hyperparameter search pipeline for Minerva. This required parameterizing all necessary options for the search, allowing users to configure it according to their needs. Additionally, we developed complementary features, such as custom checkpointing callbacks, as the default ones provided by Ray saved model weights for every epoch of each trial, which could quickly lead to storage issues.

We also introduced support for random transformations applied to both input and label pairs. This feature was critical to develop and required careful design. Since the input and label are paired, any random transformation, such as a flip, must be applied identically to both. Otherwise, the labels would no longer correspond correctly to the input. After internal discussions on how to implement this functionality, we settled on a solution that met our expectations in terms of both code quality and user experience. Support for the Hyperopt framework was also added, implemented as its own separate pipeline. This decision was made because its implementation details and requirements differed slightly from the algorithms provided by default in Ray.

At this stage of development, while all the necessary features for hyperparameter search were already in place, some additional features were still required to match the training setup used in MMsegmentation. Specifically, we needed to implement interpolation of positional embeddings to align weights trained at one resolution with architectures using a different resolution. Additionally, we required a method for performing predictions on patches, certain data transformations like random resize and crop, and a mechanism to resize predictions back to their original size, as the data pipeline resized them to match the model's input size.

Given time constraints and the complexity of certain features, we decided to initiate a backup hyperparameter search while continuing work on the remaining implementations. This backup search provided the results presented in the previous chapters. While we managed to implement several features, including all required transforms and positional embedding interpolation, the patch prediction functionality and resizing of predictions remained incomplete. As these missing features were proving to be more complex and time-consuming, we opted to proceed with the results from the backup search.

The implementation of patch prediction was adapted from a version previously developed by a lab member, though it was not initially designed with hyperparameter search in mind. To accommodate the requirements of the search, modifications were made to the feature API. However, we encountered unresolved issues when attempting to use the feature in distributed environments. Additionally, challenges arose with padding the images to enable patch creation, and with implementing overlapping patches with offsets to improve prediction quality at the patch borders.

Patch prediction was critical for the training pipeline's viability because the model's fixed input size of 512x512 was incompatible with one of the dataset's input dimensions,

255x701. In MMsegmentation, this discrepancy is addressed by resizing the smallest dimension to 512 pixels while maintaining the original aspect ratio for the other dimension. Following this resize, patch inference is applied to generate predictions, and the output is subsequently resized back to the original dimensions. Consequently, patch inference was necessary to enable the use of a model with a 512x512 input size.

Resizing the image back to its original size also posed challenges that we could not solve elegantly. Due to issues with padding for patch inference, all images were padded to a uniform size, regardless of whether their original dimensions were 255x701 or 255x401. A padding size based on the largest dimension was calculated and applied uniformly across all images. This approach introduced a significant issue: it became impossible to determine the original size of each image accurately, preventing us from cropping and resizing it back to its original dimensions.

These were the main difficulties encountered during the final stages of implementing the hyperparameter search, and they ultimately prevented achieving a one-to-one alignment between the search and training pipelines for the models.

.5 Tests

Tests are currently the least developed aspect of Minerva. Unit tests are required only for functionalities that can be evaluated using synthetic data generated at runtime. These tests primarily check whether a functionality behaves as expected but do not necessarily verify its correctness in real-world scenarios. This limitation is particularly noticeable in model tests, where random input data is passed through the model, and the tests only confirm that the forward method produces an output and that the output has the expected shape.

Executing more complex tests is challenging because the GitHub infrastructure we use for continuous testing does not support GPUs. As a result, all tests must run on CPUs, creating a significant disparity between the testing environment and the expected production environment. In production, at least one CUDA-compatible GPU is typically expected, and in many cases, a multi-GPU node or even multiple nodes is the target setup for the framework.

The core team is aware of the existing issues with testing and plans to improve both test coverage and the variety of tests. This includes adding integration tests and other types to ensure more robust validation of the framework.