UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Technology and Safety (ITS)

Segmentation of Polar Mesospheric Summer Echoes using Fully Convolutional Network

Erik Seip Domben

TEK-3901 Master’s Thesis in Technology and Safety … April 2023

To Marte and Helmer.

Abstract

Polar Mesospheric Summer Echoes (PMSE) are strong coherent radar echoes that occur in the 80 to 90 km altitude range of the mesosphere during the Arctic summer months. These echoes are of significant interest to the space physics community as they provide insight into changes that occur in the atmosphere. To better understand these changes, large datasets of PMSE echoes need to be analysed. In this study, we aimed to develop a deep learning model that could segment PMSE signal data for analysis on larger EISCAT VHF datasets. For the task, we employed a UNet and a UNet++ architecture and tested how pretrained weights from other source domains perform. Next, different loss functions were tested and last the novel object-level augmentation method ObjectAug was employed with other image-level augmentation methods to increase model performance and reduce potential overfitting due to a small training dataset. The results indicate that using randomly initialized weights was the better option for the PMSE target domain and that the use of different loss functions only had a small impact on model performance. When using image- and object-level augmentation the best performing model was reached. It was also seen that there exist inconsistencies in the PMSE signal ground-truth labels. Dividing the inconsistencies into two categories: *Granular* and *Coarse*, it was seen that using object-level augmentation had a significantly higher performance on the *Granular* labelled PMSE signal samples. Overall, our study indicates that the best performing model can be used to segment PMSE for larger datasets or as a supportive tool for further labelling of PMSE signal data.

"*Sixty percent of the time, it works every time*"
- Brian Fontana, 2016

Acknowledgements

First and foremost, I would like to thank my supervisor Puneet Sharma for his support, guidance and patience. His feedback and insight has been most valuable to the completion of the thesis. I would also like to thank my girlfriend Marte and our little boy Helmer, whose love, support, and understanding have been a constant source of inspiration and motivation. Their smiles have lit up many of the more difficult days during the writing of the thesis, and I am truly grateful for their unwavering support. I would like to extend my gratitude to the rest of my family for their love and support throughout my academic journey. Lastly, I would like to express my appreciations to the extraordinary people of PW.

Contents

List of Figures

List of Tables

1 Introduction

Polar Mesospheric Summer Echoes (PMSE) are radar echoes that occur in the mesosphere during the Arctic summer months in the polar regions[10]. These echoes are caused by the scattering of radar waves off of small ice particles that form in the 80 to 90 km altitude range of the mesosphere[32]. The formation of these ice particles is closely linked to the complex dynamics of the mesosphere, which are affected by a variety of factors.

By studying PMSE, researchers can gain insights into physical and chemical processes that occur in the mesosphere, including the formation and dynamics of ice particles, the composition of the mesospheric atmosphere, and the effects of solar radiation, solar cycles and other external factors on the mesosphere[23, 13]. This information can be used to improve our understanding of the atmosphere as a whole, as well as to develop better models for predicting and mitigating the effects of atmospheric changes due to climate change.

This thesis builds upon the work of [19, 20] which studies the separability of PMSE regions and background -and ionospheric noise. In [19] Linear Discriminant Analysis is used to pre-select regions that might contain PMSE. In [20] a random forest machine learning model was developed with the intent to investigate the shapes and altitude occurrences of PMSE over time for large datasets of EISCAT observations with the aim of analysing PMSE shapes and structures through different periods of the solar cycle. In Figure 1.1 an example of an observation from the EISCAT dataset is displayed.

In this thesis, we investigate the possibility of using Fully Convolutional Networks (FNCs) for the task of segmenting PMSE signals from the rest of the data.

FCN has become a dominant machine learning approach for the semantic segmentation of images and has shown good results in domains such as medical and satellite imagery. Because Deep Convolutional Networks (DCN) excels at preserving spatial information compared to many other machine learning methods it is well suited for learning contexts and complex patterns in images. FCNs are also filter-based methods similar to that of the random forest model[20] which used a set of hand-crafted static filters. In contrast, the number of filters in FCNs is several orders of magnitudes bigger and actively changed during training to find optimal values of the filters.

In the development of a method to segment PMSE, two FCN architectures are used: UNet[33] and UNet++[42]. To the best of our knowledge, it is the first attempt to employ deep learning models to segment PMSE data. To do this, several experiments are performed with the intent to see if other source domains i.e., natural -and medical image domains, could apply to the PMSE image domain. Further, different loss functions are tested to see if some are more suitable than others for the objective of segmenting PMSE signals. The number of PMSE samples used is relatively small in a deep learning setting which may lead to poor generalization due to over-fitting. As a countermeasure, two different augmentation approaches are tested: image-level augmentation[1] and object-level augmentation[2][41].

The thesis is structured as follows: Chapter 2 outlines the theory behind the methods used for segmenting PMSE, the metrics for evaluating the performance of the FCNs segmentation and the theory behind the Explainable Machine Learning (XML) method used for explaining the predictions on the FCNs. In Chapter 3 the dataset, implementation of the object-level augmentation method, how evaluation is performed and the experimental set-up is explained. In Chapter 4 the experiments are outlined along with the results. Chapter 5 discusses the results from Chapter 4. Finally, in Chapter 6 it is concluded and possible future work is discussed. The source code is publicly available at [3].

1. Image-level augmentation is a technique that applies transformations to an entire image to increase dataset diversity and improve machine learning model performance.
2. Object level augmentation refers to the process of applying transformations to individual objects within an image, such as resizing or changing their position, to increase the variability of a dataset and enhance model accuracy.
3. `https://github.com/Domben93/PMSEsegmentation`

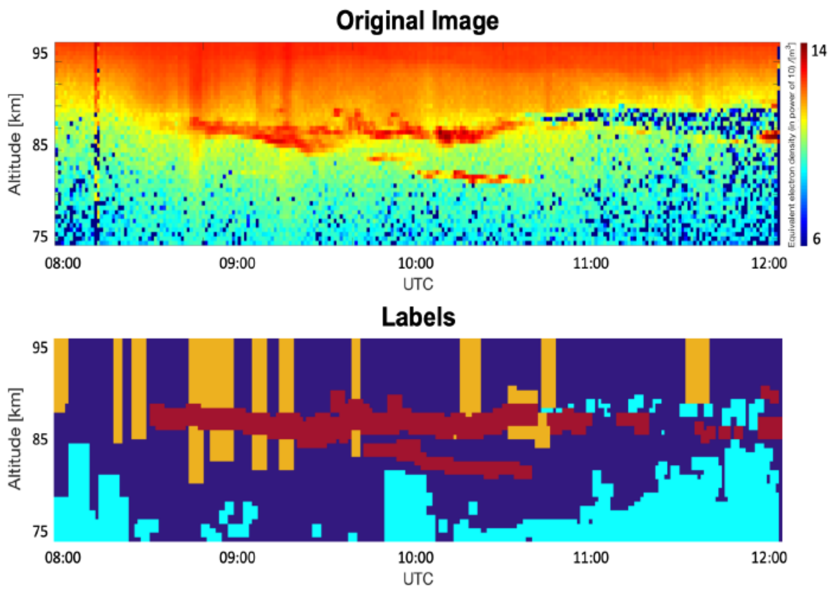

Figure 1.1: The figure is gotten from figure 2 in [20] and shows an example of a PMSE signal sample used in this study. In the top image the PMSE signal image is shown as a heatmap where the pixel values are given in Equivalent Electron Density (in power of 10)/[m^3]. In the bottom image, the ground-truth label is seen where PMSE (red), ionospheric noise (yellow), background noise (cyan) is labelled. The dark blue area are unlabelled pixels.

2 Theory

2.1 UNet architectures

In this section, two UNet architectures used for PMSE segmentation are briefly described. First, the original UNet[33] is described and secondly, an enhanced version of the UNet namely, UNet++[42] is described.

2.1.1 UNet

The UNet architecture[33] is a deep learning model that was originally designed for biomedical image segmentation. But has since been used in many different areas such as satellite and natural imagery. The UNet architecture(see Figure 2.1) consists of two main parts; a contracting and an expansive path which will be referred to as **encoder** and **decoder**, respectively.

The **encoder** reduces the dimensionality of the input data and is a method of extracting important features. In the original paper[33], every contracting layer consists of two 3x3 convolutional filters in sequence followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation that downsamples the feature maps.

The **decoder** upsamples the encoder output and passes it through the same module layers as the encoder. Finally, a 1x1 convolutional layer generates the final output, followed by an activation layer that produces an output map. This

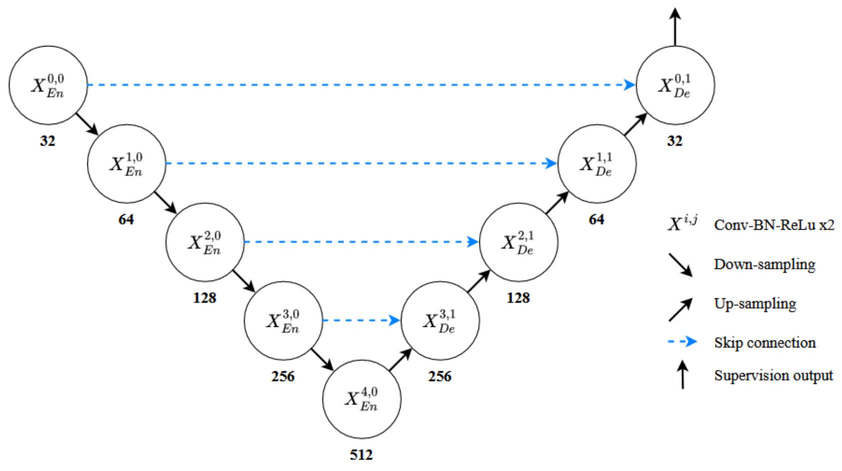

Figure 2.1: UNet architecture. Illustration of the UNet architecture. $\mathbf{X}_{En}$ denotes the encoder layers and $\mathbf{X}_{De}$ denotes the decoder layers. The figure is based on Fig.1 in [16].

map assigns a probability score to each pixel, indicating its likelihood of belonging to different classes.

Between each encoder and decoder layer in the UNet architecture, there is a skip connection[29]. Skip connections directly connect the output of a layer to the input of a subsequent layer that is not necessarily adjacent, allowing direct propagation of information between encoder and decoder layers. This helps preserve spatial information lost in up -and down-sampling operations and enhances feature reuse throughout the network[29].

Given the UNet architecture in Figure 2.1, the output from each node $X^{i,j}$ denoted as $x^{i,j}$ is formulated as follows,

$$x^{i,j} = \begin{cases} \mathcal{H}(x^{i-1,j}), & j = 0 \\ \mathcal{H}([x^{i,j-1}, \mathcal{U}(x^{i+1,j-1})]), & j > 0 \end{cases} \tag{2.1.1}$$

where the $\mathcal{H}(\cdot)$ function denotes the sequence of a convolution operation followed by a batch normalization layer and an activation function. $\mathcal{U}$ denotes the up-sampling operation and [] denotes the concatenation of the feature maps from the encoder layer and the up-sampled feature maps.

2.1.2 UNet++

UNet++[42] is an evolution of the original UNet[33] architecture described in Section 2.1.1. The encoder and decoder are similar in both architectures, but UNet++ has a re-designed skip-pathway that changes the connectivity between the encoder and decoder creating a nested UNet structure. This new skip-pathway structure has dense convolution blocks that bring the semantic level of the encoder feature maps closer to that of decoder feature maps enabling a better flow of spatial information. The assumption is that the optimization problem is simplified for the optimizer as the feature maps between the encoder-decoder pathway are more semantically similar[42]. In Figure 2.2, a representation of the UNet++ architecture is shown where the difference between the UNet architecture in Figure 2.1 is highlighted in blue.

Given the UNet++ architecture in Figure 2.2, the output from each node $X^{i,j}$ denoted as $x^{i,j}$ is formulated as follows,

$$x^{i,j} = \begin{cases} \mathcal{H}(x^{i-1,j}), & j = 0 \\ \mathcal{H}([[x^{i,k}]_{k=0}^{j-1}, \mathcal{U}(x^{i+1,j-1})]), & j > 0 \end{cases} \tag{2.1.2}$$

where $x^{i,j}$ denotes the output of node $X^{i,j}$. The $\mathcal{H}(\cdot)$ function is a convolution operation followed by a batch normalization and an activation function. $\mathcal{U}(\cdot)$ denotes the up-sampling layer and where [] denotes the concatenation of the feature maps from skip connections and the up-sampling layer.

In contrast to the UNet architecture which only has *one* top layer i.e., node $\mathbf{X}^{0,1}$, the UNet++ architecture with its re-designed skip-pathway has $L - 1$ i.e., $\mathbf{X}^{0,j}$, $j \in [1..L]$ where L denotes the layer depth of the network. Each of the top layers can easily be set to produce an output which in turn can be used with different strategies. In the paper on UNet++[42], it is defined two different deep supervision[25] strategies: **Accurate Mode** and **Fast Mode**. With **Accurate Mode**, the output from all top layer nodes is averaged allowing more of the information from the previous layers to contribute to the final prediction. With **Fast Mode**, the UNet++ model can be pruned during training. The pruning is done by checking if the performance of the output from lower top layers has to some extent comparable results. This way the network size can be reduced by removing the upper top layer.

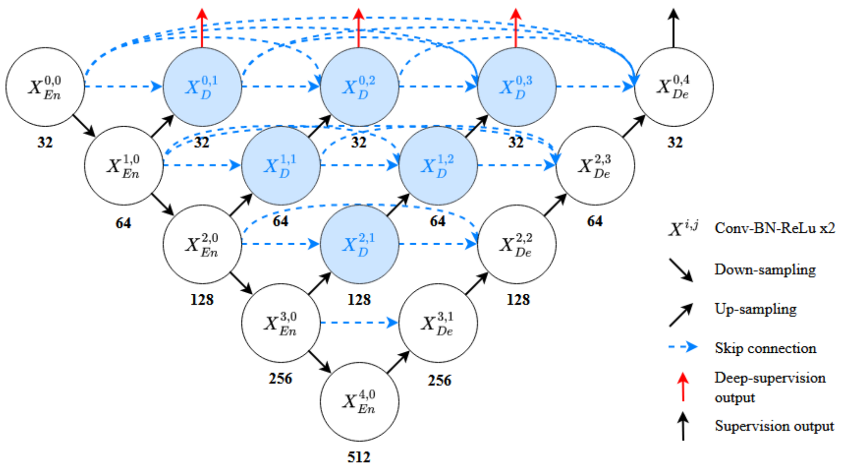

Figure 2.2: UNet++ architecture. Illustration of the UNet++ architecture from input to output. $\mathbf{X}_{En}$ denotes the encoder layers, $\mathbf{X}_{De}$ denotes the decoder layers and $\mathbf{X}_{D}$ denotes the dense layers. The figure is based the Fig.1 in [16].

2.2 Evaluation Metrics

To evaluate the performance of the semantic segmentation models two metrics are used: Jaccard Index and Dice-Sørensen Coefficient. In this section, the two metrics are briefly discussed.

2.2.1 Jaccard Index

The Jaccard index[39], often also referred to as Intersection over Union (IoU) or Jaccard similarity coefficient is a metric that is commonly used as a performance measure in semantic segmentation. The Jaccard index is a similarity measure between two sets denoted as A and B and is formulated as follows,

$$J(A,B) = \frac{|\,A \cap B\,|}{|\,A \cup B\,|} \tag{2.2.1}$$

for binary data, Equation 2.2.1 it can be rewritten as,

$$J = \frac{TP}{TP + FP + FN} \tag{2.2.2}$$

Where TP denotes True Positive, FP denotes False Positive and FN denotes False

Negative. The Jaccard Index does not include the True Negatives (TNs) in the equation, meaning that it does not account for backgrounds that are classified correctly and thus focuses on the foreground. The value of the Jaccard index is given in the range 0 to 1 where a value of 0 indicates that there is no overlap between the two sets and a value of 1 corresponds to perfect overlap between the two sets A and B.

2.2.2 Dice-Sørensen coefficient

Dice-Sørensen coefficient(DSC)[8], commonly refereed to as Dice Coefficient or F1 score, is a measure of similarity between two sets denoted as A and B. It is a widely used and well-established measure of segmentation accuracy. DSC is formulated as follows,

$$DSC(A, B) = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid} \tag{2.2.3}$$

where $\mid A \cap B \mid$ denotes the number of pixels in the intersection between set A and B. $\mid A \mid$ and $\mid B \mid$ denote the number of pixels in sets A and B, respectively. The DSC ranges from 0 to 1, where 1 indicates a perfect match and 0 indicates no overlap between set A and B.

for binary data, Equation 2.2.3 can be rewritten as,

$$\frac{2TP}{2TP + FP + FN} \tag{2.2.4}$$

The DSC is very similar to the Jaccard Index as can be seen by comparing Equation 2.2.2 and 2.2.4. In fact, they are positively correlated but are different in the sense that DSC gives more weight to the intersection which is useful in cases where false negatives should be avoided.

2.3 Data Augmentation

Data augmentation is a common technique used during the training of deep learning models aiming to increase model generalization (avoid over-fitting) and increase performance on unseen data samples[2, 41]. This can be done by generating synthetic data from the data available and subsequently increasing the training set size with artificial samples. The idea is that the artificial samples will try to fill the latent space in the vicinity of the original samples to balance the dataset.

Several techniques can be employed in the creation of synthetic data which varies a lot in terms of complexity. Some of the more classical and straightforward techniques are random rotation, random scaling, random crop and random flipping (vertical or horizontal) and has been proven effective in generalizing and improving the performance of the model[36, 34]. More complex data generation processes, such as Generating Adversarial Networks(GANs)[12, 9], have become increasingly popular in recent years and have seen good results in many different domains. However, GANs often need a large and diverse amount of training samples. For natural images[1], there are often available large datasets such as ImageNet[7] to train GANs. For other types of imagery, such as the EISCAT data images used in this study, there is a lack of PMSE signal data and furthermore, the PMSE signal itself has been subject to scientific research within the space physics community which would make it difficult to verify the precision of any trained GANs.

In this study, two categories of image augmentation are employed: Image-level and object-level augmentation.

2.3.1 Image-level augmentation

Augmentation on the image-level is the most common and easiest implemented form of augmentation[35]. With image-level augmentation the transform is applied to the whole image involving transforms such as flipping, cropping, blurring, contrast adjustment, resizing, cropping and more. Because only image flipping and contrast adjustment are used later in the study, the formulation of the image-level augmentation is limited to the three methods used; Horizontal Flipping, Vertical Flipping and Contrast Adjustment.

Horizontal and Vertical Flipping.
Horizontal and vertical flipping (reflection) are two sides of the same coin. Horizontal flipping is the procedure of flipping the image around the y-axis. For an image denoted as $\mathbf{I}$, the horizontal flipped image denoted $\mathbf{I}_{Hflip}$, is simply described as,

1. Natural images refer to photographs or visual representations of real-world objects or scenes, typically captured using a camera or similar device, and may exhibit variations in lighting, viewpoint, occlusion, and other factors that reflect the complexity and diversity of the natural environment.

$$\mathbf{I}(x,y) \longrightarrow \mathbf{I}_{Hflip}(-x,y) \tag{2.3.1}$$

and similarly for the vertical flip where the image is flipped around the x-axis is described as,

$$\mathbf{I}(x,y) \longrightarrow \mathbf{I}_{Vflip}(x,-y) \tag{2.3.2}$$

Note that in a segmentation case where a label mask is provided, the mask needs to be transformed equally to that of the image **I**

Contrast Adjustment.

Contrast Adjustment[4] is the process of changing the distribution of brightness levels in an image. Mathematically, contrast adjustment involves scaling the intensity values of the image so that they span in a narrower or wider range of values. This is done by mapping the original intensity values (typically in the range [0, 255] for an 8-bit colour or grayscale image) to new values. Given a scaling factor denoted as c, the contrast adjusted image denoted $\mathbf{I}_{Cadj}$ can be formulated as follows for a grayscale image,

$$\mathbf{I}_{Cadj}(x,y) = c \times \mathbf{I}(x,y) + (1-c) \times \mu_{\mathbf{I}} \tag{2.3.3}$$

where $\mu_{\mathbf{I}}$ denotes the mean pixel value of the original image **I** and is given as,

$$\mu_{\mathbf{I}} = \frac{1}{M \times N} \sum_{i=0}^{N} \sum_{j=0}^{M} \mathbf{I}_{i,j} \tag{2.3.4}$$

where N and M denotes the height and width of image **I**.

Because each pixel intensity is scaled, the contrast adjusted image might contain pixel values greater than that of the original image $\mathbf{I} \in [0..255]$ (given that the data type is an 8-bit unsigned integer). Thus, the contrast adjusted image needs to be clamped to this interval. The clamping of intensity values is formulated as,

$$\mathbf{I}_{Cadj}(x,y) = \begin{cases} 255, & \mathbf{I}_{Cadj}(x,y) > 255 \\ 0, & \mathbf{I}_{Cadj}(x,y) < 0 \\ \mathbf{I}_{Cadj}(x,y), & \text{otherwise} \end{cases} \tag{2.3.5}$$

The contrast factor c can be any non-negative value. Setting the contrast factor to 1 yields the same output as the input: $\mathbf{I}_{Cadj} = \mathbf{I}$. For a factor $0 < c < 1$ the contrast of the input image is decreased and for a factor $c > 1$ the contrast is increased. For $c = 0$, $\mathbf{I}_{Cadj}(x,y) = \mathbf{I}_{\mu}$.

2.3.2 Object-level Augmentation

With object-level augmentation, the transforms are applied to the individual object that is present in the image. This is a more complex task than the image-level augmentation and requires that the individual target objects can be separated from the background and the regions of the image where the objects were removed must be filled in to avoid artifacts[2]. In this study, an object-level augmentation method called ObjectAug[41] is employed.

ObjectAug.
ObjectAug[41] is an augmentation method that works on the object-level to generate new samples. The method is defined by the four modules: Image Parsing, Object Augmentation, Background inpainting and Assemble. Image parsing separates the objects from the rest of the image using the ground-truth label leaving the image with left-out areas. Then in parallel, the left-out areas in the image are inpainted and various data augmentation techniques are performed on the individual objects. Last, the objects are placed back in the inpainted image in the assemble module. A representation of the ObjectAug flow is shown in figure 2.3. The details of the four ObjectAug modules are described in the succeeding paragraphs.

I: Image Parsing.
The Image parsing module separates the individual objects from the rest of the background in the image. Given an image $\mathbf{I}^{W,H} \in [0..255]$ and the ground-truth mask $\mathbf{M}^{W,H} \in 0,1$ where a value of 1 denotes the objects foreground and 0 the background. The parsed image with the removed foreground area denoted as $\mathbf{I}_h$ is given as follows,

2. Artifacts in images refer to visual distortions or errors that are introduced during image acquisition, processing, or display, which can affect image quality and accuracy and may cause misinterpretation of image content.

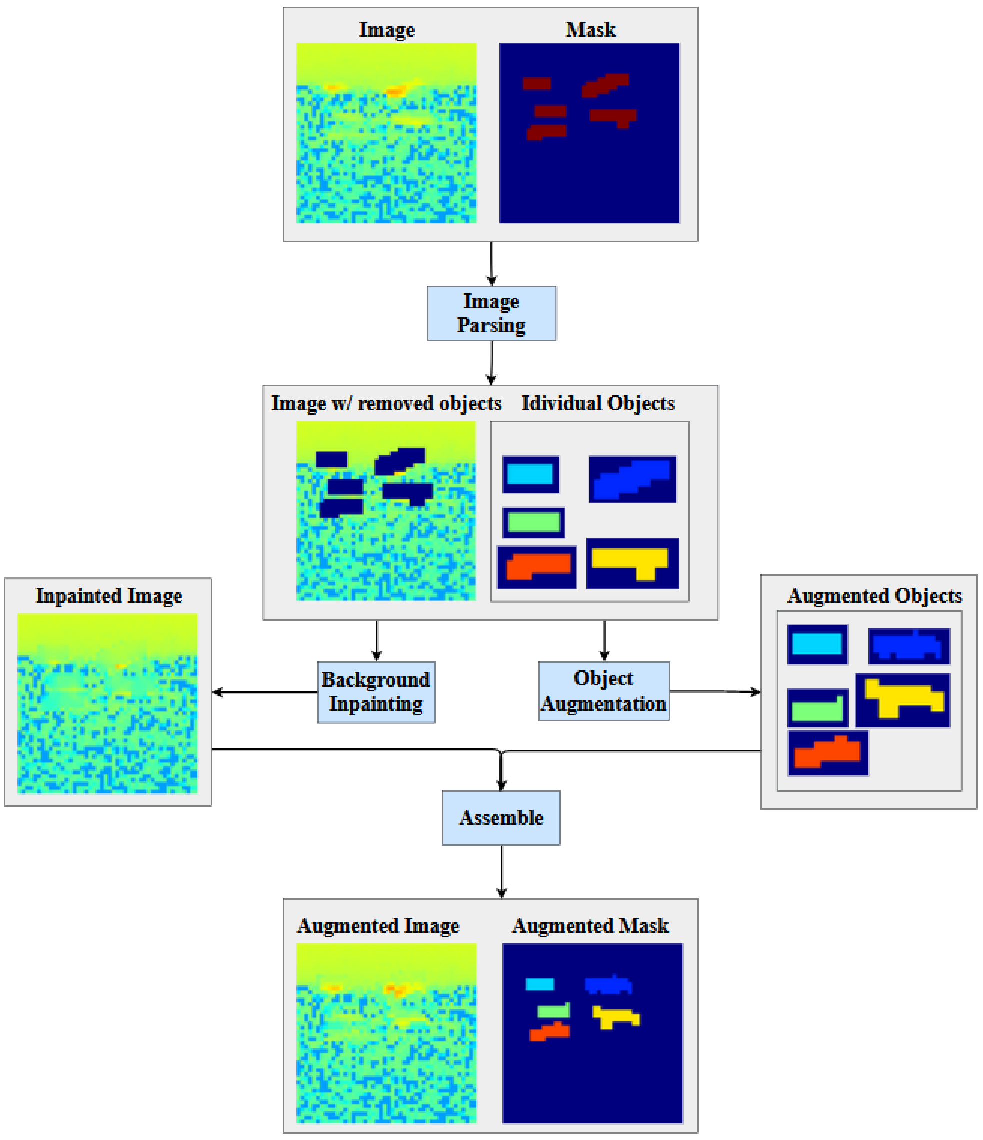

Figure 2.3: Illustration of the flow of ObjectAug[41]. The **Image Parsing** module separates the individual objects from the mask and image. The individual object is subjected to transformations in the **Object Augmentation** module and the image with the removed object areas are inpainted in the **Background Inpainting** module. The object-augmented image and mask are then generated in the **Assemble** module.

$$\mathbf{I}_h = \neg\mathbf{M} \odot \mathbf{I} \tag{2.3.6}$$

where $\odot$ denotes the component-wise multiplication of pixels and $\neg\mathbf{M}$ denotes the binary inverse of the ground-truth mask $\mathbf{M}$.

To reduce the chance of the background inpainting being affected by object pixels that are potentially left out from the ground-truth mask, the boundary of the foreground objects in the ground-truth mask is padded with ones. This is done using an all-ones kernel $\omega^{n \times n}$ convolved with the ground-truth mask $\mathbf{M}$ giving the padded ground-truth mask denoted as $\mathbf{M}_p^k$ as follows,

$$\mathbf{M}_P(x, y) = \begin{cases} 1, & \text{if } \mathbf{M}(x, y) \ * \omega \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{2.3.7}$$

The number of padded ones around the foreground boundary is given by the kernel size i.e., a 3x3 kernel will pad the mask with ones with a range of 1 around the foreground boundary. The padding range can simply be increased by changing the kernel filter size. For clarity, the ground-truth mask $\mathbf{M}$ are zero-padded before the convolution to keep the original size $\mathbf{M}^{W \times H}$.

Using the padded ground-truth mask $\mathbf{M}_p$ given by Equation 2.3.7, the parsed image $\mathbf{I}_h$ is now given as,

$$\mathbf{I}_h = \neg \mathbf{M}_p \odot \mathbf{I} \tag{2.3.8}$$

For further use in the Object Augmentation module, it is needed that the k individual objects are separated from each other into individual images and masks denoted as $\mathbf{I}^k$ and $\mathbf{M}^k$, respectively. All $\mathbf{I}^k$ and $\mathbf{M}^k$ are then cropped into patches around the object area reducing the computational load of the transforms. The k-th cropped object image and mask patch is denoted as $\mathbf{I}_c^k$ and $\mathbf{M}_c^k$, respectively. As the cropped patches are to be assembled back into the inpainted image in the last module, the crop parameters i.e., the centre position and the width and height of the patches are stored and denoted as ϕ^k.

II: Background Inpainting.

In the Background Inpainting module, the image $\mathbf{I}_h$ is inpainted using a DNN with partial convolution from [28]. The inpainting process is given as,

$$\mathbf{I}_{in} = \Psi(\mathbf{I}_h) \tag{2.3.9}$$

where Ψ denotes the inpainting and $\mathbf{I}_{in}$ denotes the resulting inpainted image. The model used for inpainting the images is further described later on in Section 4.2.

III: Object Augmentation.

In the object augmentation module, each k cropped object image $\mathbf{I}_c^k$ and mask $\mathbf{M}_c^k$ is augmented separately with different transformation methods denoted as $\mathbf{T} = [f_1, f_2, ..., f_m]$ with the corresponding probability $\mathbf{P} = [p_1, p_2, ..., p_m]$ of being invoked. The process of applying augmentation on all the masks is described as,

$$f_{aug}(\mathbf{I}, \mathbf{M} \mid \mathbf{P}) = f_1(\mathbf{I}, \mathbf{M} \mid p_1) \circ f_2(\mathbf{I}, \mathbf{M} \mid p_2) \circ ... \circ f_m(\mathbf{I}, \mathbf{M} \mid p_m) \qquad (2.3.10)$$

The defined f_{aug} object augmentation function is then applied to the cropped object images and mask as follows,

$$\mathbf{I}_{aug}^k, \mathbf{M}_{aug}^k = f_{aug}(\mathbf{I}_c^k, \mathbf{M}_c^k \mid \mathbf{P}) \qquad (2.3.11)$$

IV: Assemble.

In this last part of the ObjectAug process, the augmented objects $\mathbf{I}_{aug}^k$ are inserted back into the inpainted image $\mathbf{I}_{in}$. First, the augmented object patches $\mathbf{I}_{aug}^k$ and $\mathbf{M}_{aug}^k$ is restored to its original size using the cropping parameters ϕ^k from the image parsing module keeping the object at its centre position.

$$\mathbf{I}_{uncrop}^k, \mathbf{M}_{uncrop}^k = f_{uncrop}(\mathbf{I}_{aug}^k, \mathbf{M}_{aug}^k | \phi^k) \qquad (2.3.12)$$

Before producing the final assembled image and mask, all k $\mathbf{I}_{aug}^k$ and $\mathbf{M}_{aug}^k$ are merged. The merging of the images is denoted by the function $f_{merge}(\mathbf{I})$ and is described as follows,

$$f_{merge}(\mathbf{I}) = \mathbf{I}^1 \bullet \mathbf{I}^2 \bullet ... \bullet \mathbf{I}^m \qquad (2.3.13)$$

Finally, the assembled image denoted as $\mathbf{I}_{assembled}$ and mask denoted as $\mathbf{M}_{assembled}$ with the augmented objects is generated using equation 2.3.14 and 2.3.15, respectively.

$$\mathbf{I}_{assembled} = \mathbf{I}_{in} \bullet \neg f_{merge}(\mathbf{M}_{uncrop}) + f_{merge}(\mathbf{I}_{uncrop}) \tag{2.3.14}$$

$$\mathbf{M}_{assembled} = f_{merge}(\mathbf{M}_{uncrop}) \tag{2.3.15}$$

2.4 Loss function

For deep learning models to learn, it needs to know the error between the model prediction and a ground-truth. The error is often referred to as loss and is used to update model parameters through back-propagation. Though all loss functions returns a loss value, it can vary depending on the function in use and often has no relevance compared to other loss functions (it is not a metric). These different loss values come from the different ways that the functions penalized bad or award good predictions[18, 17]. Hence, the choice of loss function often depends on the nature of the data and can be a significant factor when it comes to the models ability to learn fast and accurately.

2.4.1 Binary Cross-Entropy

Binary Cross Entropy(BCE) is a binary version of Cross Entropy[40]. Cross Entropy is commonly used in classification and segmentation models and is a measure of the difference between probability distributions Y and Ŷ where Y denotes the network prediction and Ŷ denotes the ground-truth. The binary cross entropy is given as follows,

$$\mathcal{L}_{BCE}(\mathrm{Y}, \hat{\mathrm{Y}}) = -(\mathrm{Y}log(\hat{\mathrm{Y}})) + (1 - \mathrm{Y})log(1 - \hat{\mathrm{Y}}) \tag{2.4.1}$$

2.4.2 Dice Loss

Dice loss[38] is a loss function that is based on the Dice-Sørensen Coefficient (see Section 2.2.2) and is defined as $\mathcal{L}_{Dice} = 1 - DSC$ where DSC is calculated using Equation 2.2.3. Note that for dice loss to be differentiable the normalized logits predictions are used rather than the thresholded predictions that is used with DSC. Taking the normalized logits prediction denoted as Y and the ground-truth denoted as Ŷ the loss is given as follows,

$$\mathcal{L}_{Dice}(\mathrm{Y}, \hat{\mathrm{Y}}) = 1 - \frac{2 \mid Y \cap \hat{Y} \mid}{\mid Y \mid + \mid \hat{Y} \mid} \tag{2.4.2}$$

2.4.3 Focal Loss

Focal Loss[27] is a variant of Binary Cross-Entropy that focuses more on the difficult samples. This is particularly helpful in cases where there is a class or category imbalance. Focal Loss is derived from cross-entropy as,

$$CE(\mathrm{Y}, \hat{\mathrm{Y}}) = \begin{cases} -log(\mathrm{Y}), & \text{if } \hat{\mathrm{Y}} = 1 \\ -log(1 - \mathrm{Y}), & \text{otherwise} \end{cases} \tag{2.4.3}$$

where $y \in \{0, 1\}$ defines the ground truth and $p \in [0, 1]$ is the estimated probabilities from the model output. For more convenient notation, Focal Loss[27] defines the estimated probability of a class as Y_t,

$$\mathrm{Y}_t = \begin{cases} \mathrm{Y}, & if\ \hat{\mathrm{Y}} = 1 \\ 1 - \mathrm{Y}, & \text{otherwise} \end{cases} \tag{2.4.4}$$

as such, cross-entropy can be rewritten as,

$$CE(\mathrm{Y}, \hat{\mathrm{Y}}) = CE(\mathrm{Y}_t) = -log(\mathrm{Y}_t) \tag{2.4.5}$$

In Focal Loss, a modulating factor $(1 - \mathrm{Y}_t)^\gamma$ is added to the cross-entropy. This factor down-weight the easy samples such that the hard samples are given more weight to the final loss. For a $\gamma = 0$ the Focal Loss is equal to cross-entropy. In addition to the modulating factor the authors of the original paper[27] use a weighting factor $\alpha_t \in [0, 1]$. The weighting factor can either be treated as a hyperparameter that is tuned or could be set inversely proportional to the class frequency. The final Focal Loss is expressed as follows,

$$\mathcal{L}_{Focal} = -\alpha_t (1 - \mathrm{Y}_t)^\gamma log(\mathrm{Y}_t) \tag{2.4.6}$$

2.4.4 Boundary Loss

The idea behind boundary losses is to penalize the model for incorrect predictions along the boundaries between the prediction and the ground-truth.

Here, the boundary loss from [21] is formulated. The term *boundary loss* is often used as a description of a type of loss function i.e., similarly to how Binary Cross-Entropy loss and Dice loss are referred to as distribution based and regional based loss, respectively. However, in further parts of this paper the term *boundary loss* will be referred to as the loss from [21] which is formulated as follows.

Given an image $\mathbf{I} : \Omega \subset \mathbb{Z}^{2,3} \longrightarrow [0..255]$ and a ground-truth $\mathbf{g} : \Omega \longrightarrow \{0, 1\}$

where Ω denoted the spatial domain. A pixel p belongs to the foreground region (target) $G \subset \Omega$ if $g(p) = 1$ and $g(p) = 0$ otherwise. $s_\theta : \Omega \longrightarrow [0, 1]$ denotes the sigmoid probability output of being foreground or not. Let $S_\theta = \{p \in \Omega \mid s_\theta(p) = \delta\}$ be the segmentation region given for some threshold δ.

The boundary loss defines a method to build a distance loss $Dist(\partial G, \partial S_\theta)$ that uses space contours in the spatial domain Ω, where ∂G denotes the boundary of the ground-truth region G and ∂S_θ denotes the boundary of the segmented region given by the network output.

Further, boundary loss is inspired by curve evaluation methods[5] which requires a measure for evaluating boundary changes. Here, a non-symmetric L_2 distance on the space shapes is used that gives a measure on the change between the boundaries ∂G and ∂S_θ i.e., the change between the ground truth boundary and thresholded output boundary, which is defined as follows,

$$Dist(\partial G, \partial S_\theta) = \int_{\partial G} \| y_{\partial S}(p) - p \|^2 \, dp \tag{2.4.7}$$

where $\| \cdot \|$ denotes the L_2 norm, $p \in \Omega$ is a point on the boundary ∂G and $y_{\partial S}(p)$ denotes the corresponding point on boundary ∂S perpendicular to ∂G at point p. Equation 2.4.7 directly uses points on the contour ∂S which cannot be used directly as a loss for $\partial S = \partial S_\theta$. Instead, boundary loss approximates Equation 2.4.7 by using an integral approach[6]. In Figure 2.4, an illustration of the differential (Equation 2.4.7) and integral (Equation 2.4.8) approach is shown. By using the integral approach, the need for local differential computation for the contour points is circumvented and instead represents boundary changes as a regional integral as follows,

$$Dist(\partial G, \partial S) \approx 2 \int_{\Delta S} D_G(q) dq \tag{2.4.8}$$

where ΔS denotes the region between the two contours. $D_G : \Omega \longrightarrow \mathbb{R}^+$ denotes the distance map with respect to the ground-truth boundary ∂G. The distance $D_G(q)$ for any point $q \in \Omega$ is calculated by taking the closest contour point $z_{\partial G}(q) = p$ on the ground-truth boundary $\partial G(p)$ such that $D_G(q) = \| q - z_{\partial G}(q) \|$. For further information about how Equation 2.4.8 is approximated see the original paper[21].

The final boundary loss that approximates boundary distance $Dist(\partial G, \partial S_\theta)$ is defined as,

$$\mathcal{L}_B(\theta) = \int_\Omega \phi_G(q) s_\theta(q) dq \tag{2.4.9}$$

where $\phi_G : \Omega \longrightarrow \mathbb{R}$ denotes the level set function of the boundary ∂G where for a point $q \in G$, $\phi_G(q) = -D_G(q)$ and $\phi_G(q) = D_G(q)$ otherwise. $s_\theta(q)$ denotes the sigmoid activated network outputs. The level set function $\phi_G(q)$ encodes the distance from all q points to the boundary ∂G and is pre-computed directly from the ground-truth G as illustrated in Figure 2.5.

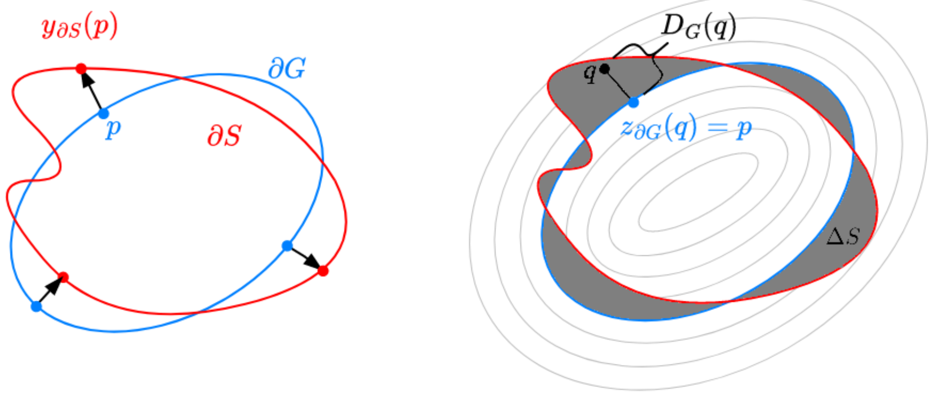

Figure 2.4: Differential (left) and integral (right) approach for measuring boundary change. The illustration is based on Figure 2 in the original boundary loss paper[21].

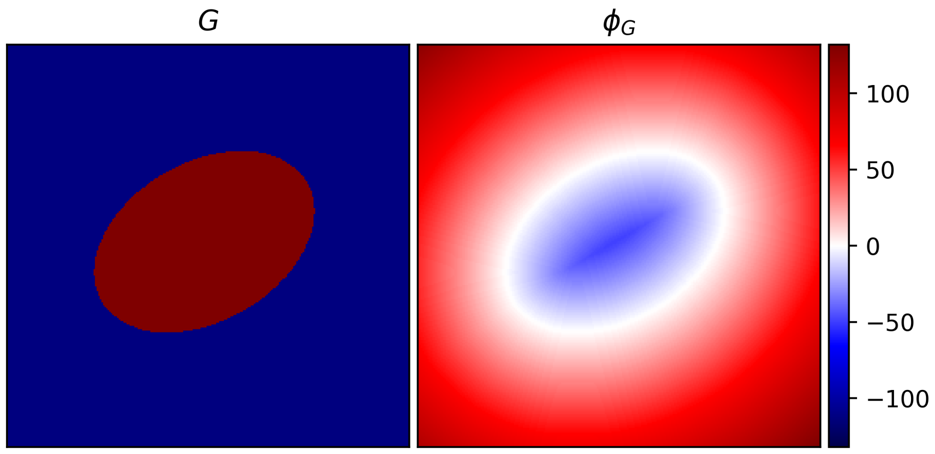

Figure 2.5: Illustration of pre-computed level set function ϕ_G (right image) from ground-truth G (left image).

2.4.5 Combination Loss

Combination loss functions are constructed by combining two or more loss functions. The idea is that different loss functions capture different aspects of the desired behaviour of the model. Put in other words, with combination loss the model is set to learn multiple objectives. This can make the model more robust to different types of noise and variations in the training data as they account for multiple aspects of the data. The main drawback of using multiple loss functions is the increased complexity that is brings in terms of hyper-parameters. In addition to finding hyper parameters for the loss functions, the weighting between how much each loss function contributes to the final loss is also a factor that affects the training process.

In the coming sections, two combination losses used in later parts of this study are briefly discussed.

Dice-BCE Loss

A type of combination loss that was used with the original UNet++[42] is the combination between Dice loss and Binary Cross Entropy. This combination loss is described as,

$$\mathcal{L}_{Dice+BCE}(Y,\hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}\Big(\frac{1}{2}\cdot Y_i \cdot log\hat{Y}_i + \frac{2\cdot Y_i\cdot \hat{Y}_i}{Y_i\cdot \hat{Y}_i}\Big) \tag{2.4.10}$$

where N is the batch size and Y_i and $\hat{Y}_i$ is the predicted probabilities and ground truth, respectively. From Equation 2.4.10 it can be seen that the binary cross-entropy is down-weighted by a factor of 0.5. In further parts of this study, the loss will be referred to as $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$.

Dice-Boundary Loss

Another type of combination loss is the Dice Boundary Loss, which combines Dice Loss [38](see Section 2.4.2) and Boundary Loss[21] (see Section 2.4.4). The boundary loss was suggested as a method for mitigating issues related to regional losses (such as Dice Loss) in cases of highly unbalanced segmentation. Because most regional losses penalize all points within a region equally regardless of the distance from the boundary it can be difficult for regional losses to fit the predictions to the ground-truth regions, particularly for small regions[21]. As such, the boundary loss combined with the regional loss to accommodate for this potential problem. The combination of dice loss and boundary loss is given as the sum of the two losses and will be referred to as $\mathcal{L}_{Dice} + \mathcal{L}_B$ in further parts of this study.

In the boundary loss paper[21] three different weighting strategies of the regional and the boundary loss are proposed defining the weighting parameter denoted as α. The first strategy is named ***constant*** which simply involves setting the parameter to a constant $\alpha = n$ where the total loss is given as follows,

$$\mathcal{L}_{Dice} + \alpha\mathcal{L}_B \tag{2.4.11}$$

the second strategy named ***increasing*** involves setting $alpha > 0$ in Equation 2.4.11 to a low value initially and increase it every i iterations. With this strategy, the dice loss will remain constant while the contribution of the boundary loss increases.

The third strategy is called ***rebalance*** and is formulated as,

$$(1 - \alpha)\mathcal{L}_{Dice} + \alpha\mathcal{L}_B \tag{2.4.12}$$

Similarly to the ***increase*** strategy, the boundary loss is weighted harder every i iteration. But in contrast to the ***increase*** strategy, the ***rebalance*** down weights the dice loss contribution by a factor of $(1-\alpha)$. Note that if $\alpha = 0 \longrightarrow \mathcal{L}_{Dice} = 0$, removing the contribution of the dice loss completely. In the boundary loss paper[21] it is stated that the co-occurrence of a regional loss with the boundary loss is important because cases of empty foreground in the ground-truth and very low predicted values, the gradients can become very low and thus the solution is close to local minimum or saddle point.

2.5 Layer-wise Relevance Propagation

Layer-wise Relevance Propagation(LRP)[30] is a method used to understand and interpret decisions made by deep learning models. LRP uses gradients from backwards propagation to identify the contribution of each input feature to the final decision made by the model. Hence, it is possible to identify which input features that are of high and low relevance for the final predictions made by the network.

LRP is most commonly used for classification tasks with networks such as

ResNet[14] and VGG16[36] where the relevance of the output is back-propagated through the layers of the network until the input layer is reached. LRP assigns a relevance score to each pixel of the input image which indicates how much it contributed to the final output. An illustration of the flow of LRP is shown in Figure 2.6 which shows the UNet[33] architecture from Section 2.1.1 where the forward pass of the input is shown in black and the backwards propagation of the prediction is shown in red.

LRP implements a propagation procedure which is subject to a conservation property that lets the received quantity from one layer propagate to another in equal amount[30]. Considering two consecutive feature maps j and k. The propagation relevance score denoted as R_k at a given point onto the lower layer can be found using,

$$R_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} R_k \tag{2.5.1}$$

where z_{jk} quantifies how much a point j in the higher layer contributes to making the points k in the lower layer relevant.

Depending on the layer type, different rules are usually applied in the LRP. For the case used in this study, only three rules are needed: $z^{\mathcal{B}}$-rule, LRP-ϵ and LRP-γ.

Before explaining the mentioned rules, the Basic Rule (LRP-0)[3] that the other rules are based on is formulated as follows,

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k \tag{2.5.2}$$

The Basic Rule follows Equation 2.5.1 where the contribution of z_{jk} to the higher layer is defined by the activation function denoted and weights of the layer R_k as $z_{jk} = a_j w_{jk}$.

Epsilon Rule (LRP-ϵ).

Applying only the Basic Rule to a network will generate a relevance output that is equal to $Input \times Gradient$. Because gradients can be noisy and thus give a relevance output that is hard to interpret, the Epsilon Rule[3] can be used. By adding a small value $\epsilon \in \mathbb{R}^+$ to the denominator in Equation 2.5.2 it is possible to reduce the contributions of small noisy values from the higher

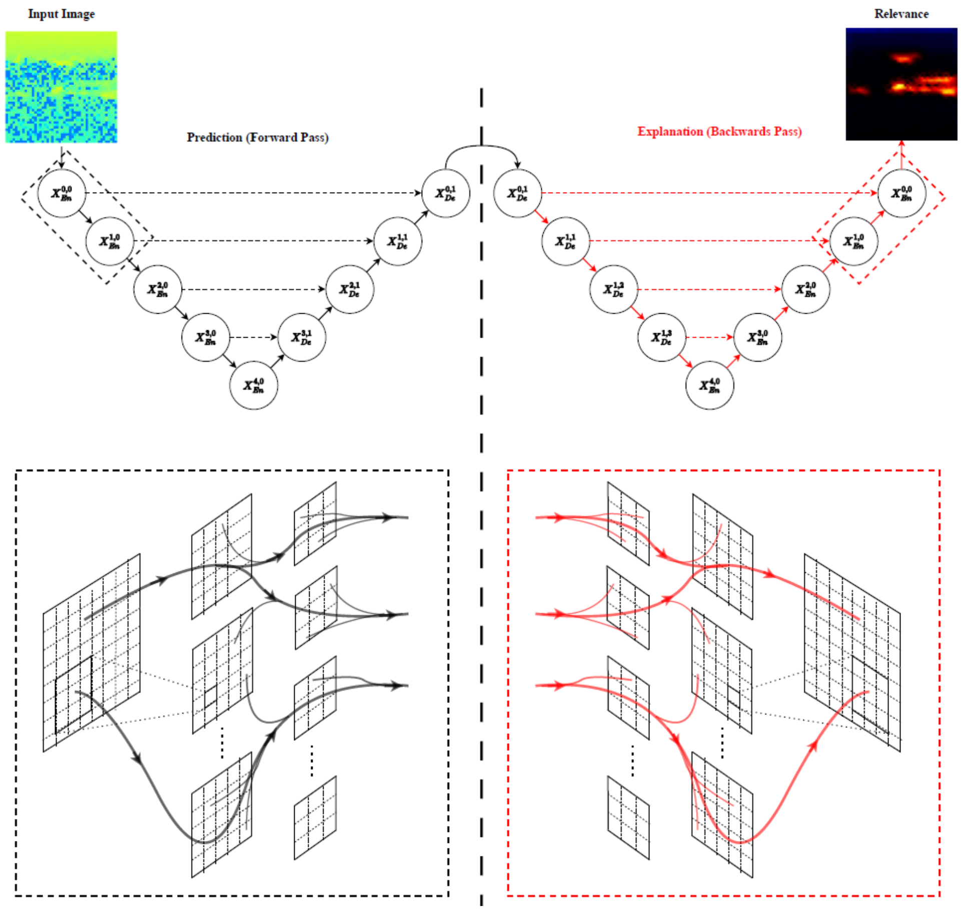

Figure 2.6: Illustration of LRP. The top figure illustrates the LRP flow from the input image to the left and to the explained relevance output to the right. The bottom figure illustration shows a more detailed view of the relevance flow through the networks feature maps in the forward pass (black coloured arrows) and the backwards pass (red coloured arrows) in the first two encoder layers, respectively.

layer k leading to improved explainability. The Epsilon Rule is formulated as follows,

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k \tag{2.5.3}$$

Gamma Rule (LRP-γ).

The Gamma Rule[30] is a way of favouring positive contributions in the backwards propagation of relevance. With this approach, the weights w_{jk} from Equation 2.5.2 is now given as $(w_{jk} + \gamma w_{jk}^{+})$ where w_{jk}^{+} denotes the positive valued weights and $\gamma \in \mathbb{R}^{+}$ denotes the contribution of the positive values weights. The Gamma Rule is formulated as follows,

$$R_j = \sum_k \frac{a_j(w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j(w_{jk} + \gamma w_{jk}^+)} R_k \tag{2.5.4}$$

When positive contributions are prevalent, they can limit the growth of both positive and negative relevance during the backwards propagation phase. Because the positive contributions act as a moderating force, it prevents extreme values from dominating. As a result, more stable explanations that are easier for humans to interpret might be generated.

ZBox Rule ($z^{\mathcal{B}}$).

The ZBox Rule[31] is used for bounded inputs such as images and is formulated as follows,

$$R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j \tag{2.5.5}$$

where $x_i : \forall_i \, l_i \leq x_i \leq h_i$. Here, $l_i \leq 0$ and $h_i \geq 0$ denote the smallest and largest pixel values respectively for each image dimension.

/3 Methods

3.1 Dataset

In this study, we use a dataset that comes from EISCAT VHF(Very High Frequency) radar located in the vicinity of Tromsø. The data are observations of Polar Mesospheric Summer Echoes (PMSE) that are detected during the Arctic summer months in the altitudes of 80 to 90 km[10]. Each sample in the dataset is from *one* observation that typically lasts from a few to several hours within a 75 to 95 km altitude range. The samples contain measured backscatter power as a function of time and altitude where the time (horizontal) resolution is approximately one minute and the altitude (vertical) resolution varies between 0.30 to 0.45 km[20]. The dataset consists of a total of 18 labelled samples and each sample is represented as a grayscale image. The grayscale images will be represented as heatmaps equal to that in [20] when shown throughout the study to make it easier to distinguish between the different parts of the samples. In the top image in Figure 3.1, *one* of the samples in the dataset is shown where a pixel value refers to the Equivalent Electron Density from the standard GUISDAP analysis[26] and where the maximum and minimum value is given in red and blue, respectively. Details about year, month, day and time the different samples were acquired can be found in Table A-1 in the Appendix. For further information about the labelling process, we refer the reader to the original paper[20].

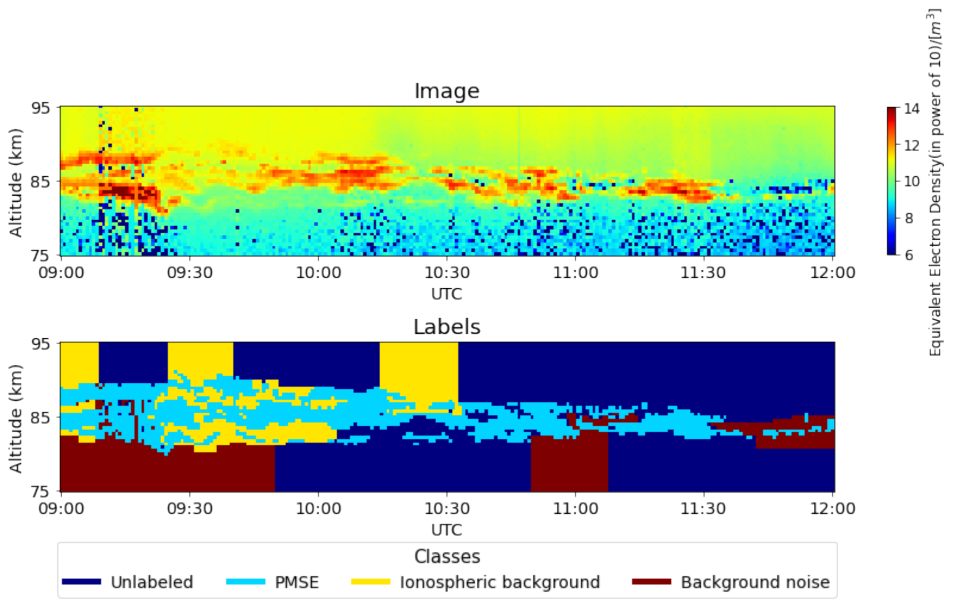

Figure 3.1: Example of a PMSE signal image with the ground-truth label below. The altitude range varies from 75 to 95 km and the time length of the observation goes from 09:00 to 12:00 UTC. The colour scale in the image represents Equivalent Electron Density (in power of 10)/[m^3]. The labels are divided into four classes: Unlabeled(Dark Blue), PMSE(Cyan), Ionospheric Background(Yellow) and Background Noise(Brown).

Class Reduction.

The original dataset has four different classes, namely, PMSE, background noise, unlabelled and ionospheric background[20] (see the bottom image in Figure 3.1). To simplify the process of segmenting out the main class target PMSE. The ionospheric background, background noise and unlabelled classes are merged into *one* class which will be referred to as background, thus making the objective of segmenting PMSE signals binary. This does however create a big class imbalance in the dataset where a global ratio between the foreground and background class is approximately 1:9, but for a single data sample, the class imbalance ratio may be as low as 1:138.

Sample Splitting.

To be used for training and testing in deep learning, each input sample should be of the same size. Hence, each sample is divided into square patches that are zero-padded on each side to make 64x64 pixel samples. The samples in the dataset have four different altitude resolutions: 22, 48, 58 and 60. The samples with a height of 22 pixels are first rescaled to a 44x44 size using nearest neighbour interpolation and then zero-padded to 64x64 sized samples. The intention behind scaling up the samples with the smallest altitude resolution is to make them more similar to the shape of the other samples as regardless of altitude resolution all samples are still given in the 75 to 95 km altitude range.

Dataset Split.
The dataset is split into training, validation and test sets where each sample is split in a stratified manner[1] with the ratio 60%/20%/20%, respectively. The training samples are additionally split such that they overlap by 30%.

Pre-Processing.
The samples, where each pixel is given as a backscatter power, are first standardized into zero mean and unit variance based on the training dataset before it is normalized into float value in the range 0 to 1. While it is strictly not necessary, it is considered a good practice as it helps to improve training stability as features are scaled to a consistent range. In addition, it helps the model converge faster and prevent the vanishing or exploding gradient problem[24].

3.2 Object-level Augmentation

As a method of increasing the number of samples and diversifying the original dataset, the ObjectAug[41] method explained in Section 2.3.2 is employed. Before going into details about the object-level augmentation, the training process of the inpainting DNN used to inpaint the removed areas of the image is explained.

Inpainting DNN.
The objective of the inpainting DNN is to inpaint PMSE regions that have been removed from the image. For this task, the inpainting DNN needs to be trained to learn the background pattern that is present in the PMSE images. To train the inpainting DNN, we use the PMSE dataset as explained in Section 3.1. However, to prevent any partiality in the training of the segmentation models and the inpainting model, the dataset is mirrored. This implies that the segmentation training set serves as the inpainting model validation set, while the segmentation validation and test set serves as the inpainting model training set.

The training data is generated from the original PMSE images by removing areas for the inpainting model to fill. Meaning that the original PMSE image

1. Stratified splitting of samples: dividing the population into subgroups and taking proportional random samples from each stratum to reduce sampling error.

serves as the ground-truth comparing it to the inpainting model prediction. Note that only the areas which are inpainted are compared to the ground-truth when calculating the error. The areas removed from an original PMSE image denoted as $\mathbf{I}_i \in [0..255]^{W,H}$ is given by a pre-computed mask denoted as $M^k_{inp_i} \in \{0,1\}^{W,H}$, where k denotes the number of masks generated for image i and W and H denotes the width and height of the image and mask respectively. A mask $\mathbf{M}^k_{inp_i}$ is generated by creating n rectangular patches denoted as $\mathbf{m}^n_p \in \{1\}^{w,h}$, where $w \in [1..20]$ and $h \in [1..10]$ denotes the width and height respectively and is randomly chosen for each patch. The width and height interval is arbitrarily chosen to simulate the fact the PMSE usually have a pattern that elongates horizontally[20]. All n patches are inserted into the mask $\mathbf{M}^k_{inp_i}$ at randomly selected locations such that it does *not* overlap with any of the PMSE signal regions defined by the ground-truth mask denoted as $\mathbf{M}_{PMSE_i}$. The reason is that we only want the inpainting model to learn the background pattern to avoid the removed PMSE areas being inpainted with a PMSE-like pattern. Hence, the n patches are placed such that $\mathbf{M}^n_p \cap \mathbf{M}_{PMSE_i} = \varnothing$. Note that there are no restrictions on the overlapping between the rectangular patches allowing for more irregular shapes to be generated. An example of a training mask for the inpainting model can be seen in the bottom image of Figure 3.2.

The number n of rectangular patches $\mathbf{M}^n_p$ for each mask $\mathbf{M}^k_{inp}$ given by Equation 3.2.1,

$$k = \frac{W \times H}{200} \tag{3.2.1}$$

where W and H denote the width and height of the PMSE image, respectively. By changing the number of patches as a function of image size the total amount of area removed from each image will be somewhat similar. The constant in the denominator is found through visual evaluation and is chosen such that the rectangular patches are prone to overlapping each other but also avoid large regions to be removed, something the inpainting model might struggle with as noted in [28].

During training, a training sample $\mathbf{I}_i$‘ is generated as follows,

$$\mathbf{I}'_i = \mathbf{M}^k_{inp_i} \odot \mathbf{I}_i \tag{3.2.2}$$

where $\mathbf{M}^k_{inp_i}$ denotes one of the k generated masks for PMSE image i.

For the training of the inpainting DNN, a UNet architecture is used with the same depth as the model described in 2.1.1 using ResNet50[14] trained on ImageNet[7] as the backbone. The same loss from [28] is used with the Adam optimizer algorithm and with a learning rate of 0.0005. The model is trained for a maximum of 10000 iterations with a mini-batch size of 32.

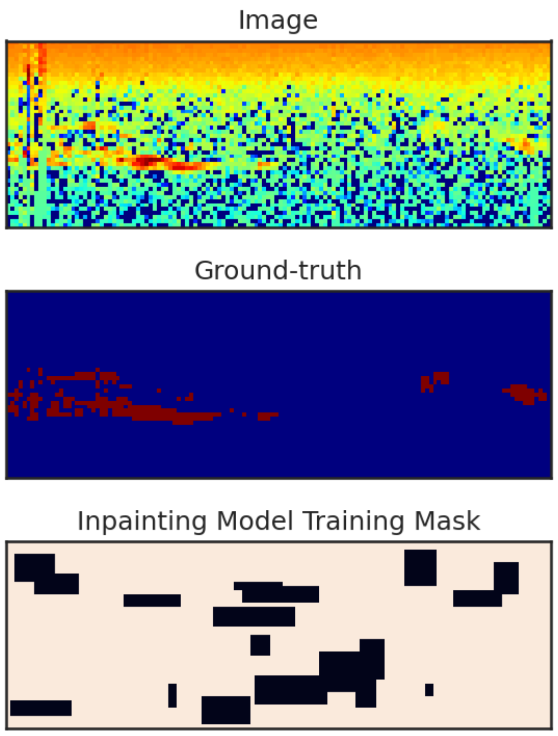

Figure 3.2: The figure shows an example of a mask used in training of the inpainting model. In the top image, the PMSE signal sample is displayed with the ground-truth label in the middle. The bottom image shows the mask used for removing parts (black area) of the PMSE signal image. Note how the black areas do not overlap with any PMSE signal in the ground-truth.

Object Augmentation.

The object augmentation using the ObjectAug[41] method described in Section 2.3.2 requires some additional specifications.

The first module of the ObjectAug[41] method is the Image Parsing(see Section 2.3.2) module where the PMSE regions defined by the ground-truth are extracted from the image. To avoid any leftover pixels that might be unlabelled PMSE, the ground-truth is padded with ones around the foreground PMSE boundary.

There is need for some clarification around what is considered separate PMSE regions. Although some PMSE regions are separated by a suitable number of pixels and can, to a certain degree, be interpreted as "different" PMSE regions, it is not clear to the author of this study if PMSE regions in close vicinity of each other are connected or not. However, given that the separation between PMSE regions is small in many cases, a PMSE region will be considered an individual region if there are 1 or more background pixels separating the boundaries of the regions. This small distance between regions is selected because the number of PMSE regions decreases drastically if the distance is set to a higher value.

For the Object Augmentation module (see Section 2.3.2), ***resizing*** and ***location shifts*** are applied as the augmentation methods where each has a probability of $p = 0.5$ of being invoked.

For the ***resizing*** augmentation, the scaling of the PMSE region is based on a random number denoted as $n \in [-3..3]$ where n denotes the number of pixels the object is scaled up or down. The reason that the objects are only scaled up or down with a small number of pixels is to avoid PMSE regions either becoming too big i.e., the PMSE stretches into regions of other PMSE and/or outside 80 to 90 km altitude range that the PMSE is believed to occur in[10], or too small i.e., the PMSE region disappears or is reduced to only a few pixels.

For the ***location shift*** augmentation, the PMSE region is simply shifted horizontally and/or vertically. The number of pixel points that the objects are shifted is randomly selected in the range $[-3..3]$ (horizontally and/or vertically). The shifting range is limited to only a few pixels to avoid PMSE regions from overlapping by too much or being shifted outside of the 80 to 90 km altitude region. It is however argued that location shifting the PMSE region by only a small amount is sufficient in trying to get the model to learn different boundaries between the background and PMSE regions.

In theory, several different augmentation methods such as rotation, flipping, etc., could be used. However, using the rather conservative ***resizing*** and ***location shift*** augmentation that was mentioned, it is believed that the PMSE is not altered too much and is kept somewhat natural.

The object augmentation process is computationally heavy. Thus, to speed up training, the object-augmented data used during training is pre-computed. For each of the 18 PMSE samples in the dataset, 50 new samples are generated giving 900 new samples. In addition to the 900 new samples, we include 180 samples of the original dataset i.e., the original dataset copied 10 times, such that 20 percent of the total samples are non-augmented. The reason for this is that the inpainted images will create a different background around the PMSE than for the original samples, which might result in a model that is overfitted to the new boundary between foreground (PMSE) and background.

3.3 Evaluation

For the model quantitative performance evaluation, Intersection over Union (IoU) and Dice-Sørensen Coefficient (DSC) are used and are averaged over all samples. The performance is reported for both the validation and test set with mean and standard variation from running the training process 5 times. The best evaluation metrics from each set of experiments is underlined.

In all quantitative results the probability output denoted as P, is thresholded with a value of 0.5 where values above 0.5 are assigned to the foreground (PMSE) class and values below to the background (Noise) class. The thresholded output is denoted as $P_{Threshold}$ and is expressed as follows,

$$P_{Threshold}(x, y) = \begin{cases} 1, & P(x, y) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}, \forall x, y \in P \tag{3.3.1}$$

For all qualitative evaluations, the predictions are generated using the model from the 5 training runs that had the best IoU performance.

The Layer-wise Relevance Propagation[30] from Section 2.5 is used to visualize what regions of the input are considered important for the model output .i.e, relevance. The relevance can be displayed differently and depends on different factors such as the type of attribution method and LRP rules. For the relevance displayed in later parts of the study, it is used a gradient attributions method that computes the gradient and with the LRP rules; LRP-γ and $z^{\mathcal{B}}$ from 2.5. The LRP is implemented using the Zennit[1][2] framework.

3.4 Experimental Set-up

In the experiments, two different UNet architectures are used; The UNet model described in Section 2.1.1 and the UNet++ model described in Section 2.1.2. Given that the problem of segmenting PMSE signal data is reduced to a binary problem (See Section 3.1), sigmoid activation is used to produce the final output. Both models are set to a layer depth of $L = 4$. The number of initial feature maps is set to 32 or 64 and will be denoted as e.g., UNet^{32} or UNet^{64} if 32 or 64 initial feature maps are used, respectively.

Both random initiated weights and pretrained weights are used during the

2. `https://zennit.readthedocs.io/en/latest/`

experiments. In the latter case, the pretrained weights are only used in the encoder layers $X^{i,0}, i \in [0..L]$ (see Figure 2.1 & 2.2). The remaining layers are initiated using Kaiming He[15] initialization. In the random initiated case, all layers are initiated using Kaiming He initialization.

For all experiments, the Adam optimizer algorithm[22] is employed with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ which is suggested as a good starting point in the Adam paper[22]. To avoid overfitting of the data, an early stopping mechanism is used similar to that of algorithm 7.1 in [11] where the best parameters denoted as θ^* are selected based on the validation set error. Depending on if the data is augmented, the patience p is set to a different value. For the models using no augmentation $p = 10$ and for the models using augmentation $p = 20$. For the latter case, the patience is set higher because the validation error is more irregular during training because there are subsequently more different samples. The models are evaluated every 10 iterations using a mini-batch size of 8 randomly selected samples and run until the early stopping criteria are met. The models are trained on an Nvidia RTX3070(Notebook) GPU with 8GB of VRAM.

4 Experiments and Results

In this chapter, we outline the various experiments conducted and the corresponding outcomes. Each section presents a detailed account of the experiment, followed by both quantitative and qualitative results.

4.1 Initial Experiment

Initially, a set of experiments are conducted; One with the UNet[33] (see 2.1.1) model and one with UNet++[42] (see 2.1.2) model with *no* augmentation. Different variations of both models are tested as follows,

1. Random initiated weights with 32 and 64 initial feature maps.

2. Pretrained weight initiation of the encoder layers $\mathbf{X}^{i,0}$, $i \in [0..L]$, where $L = 4$ (see figure 2.1 and 2.2). For the models with 32 initial feature maps, a pretrained UNet model found at [1] is used as the backbone. For the models with 64 initial feature maps, a VGG16[36] model pretrained on ImageNet[7] is used as the backbone.

In this initial experiment, the dice loss function (see Section 2.4.2) is employed

1. `https://pytorch.org/hub/mateuszbuda_brain-segmentation-pytorch_unet`

for all models. To find suitable values for the learning rate and weight decay parameters for the optimizer, a random hyperparameter search was conducted prior to this experiment. The random hyperparameter search was run for the different model variations outlined in 1 and 2 with random learning rate and weight decay in the closed interval $[0.01, 0.0001]$. In Table 4.1, the learning rate and weight decay are listed for each of the models run in the coming experiments for UNet and UNet++. The selection of parameters is chosen based on plots from the hyperparameters searches found in Figure A-1 to A-8 in the Appendix. The values are chosen based on the highest Dice-Sørensen Coefficient (see Section 2.2.2) score in a region where the loss is reasonably stable and where the difference between the train and validation score is not too big i.e., the model is likely to be more generalized and not overfitted to the training set.

Table 4.1: Learning Rate and Weight Decay values used during training of the different models listed in Table 4.2. The values selected are based on the hyperparameter search in Figure A-1 to A-8.

Model - Initiation	Hyperparameters	
	Learning Rate	Weight Decay
UNet^{32} - RandomInit	0.008	0.005
UNet^{32} - Pretrained	0.003	0.007
UNet^{64} - RandomInit	0.006	0.005
UNet^{64} - Pretrained	0.003	0.007
UNet++^{32} - RandomInit	0.005	0.005
UNet++^{32} - Pretrained	0.003	0.006
UNet++^{64} - RandomInit	0.002	0.006
UNet++^{64} - Pretrained	0.001	0.008

The exponential in e.g., UNet^{32} or UNet^{64}, denotes the number of output feature maps from the first encoder layer.

4.1.1 Quantitative Results

Table 4.2 displays the quantitative results from the initial experiment, demonstrating the Intersection over Union (IoU) and Dice Similarity Coefficient (DSC) scores for both the test and validation sets. The best results are underlined. It is apparent from the table that there are only minor variations in performance among the different models. However, the best performing model, UNet++^{64} with randomly initialized weights, displays a slightly higher performance score. Notably, Table 4.2 also indicates that models utilizing pre-trained weights exhibit lower performance than those with randomly initialized weights, despite employing the same model architecture and a different number of initial feature maps.

Table 4.2: Quantitative performance of different UNet architectures with 32 or 64 initial feature maps. IoU and DSC are reported for the test and validation set. The best performing model is underlined.

Model - Weight Initiation	Test		Validation	
	IoU↑	DSC↑	IoU↑	DSC↑
UNet^{32} - RandomInit	**0.654**±*0.006*	**0.791**±*0.005*	**0.710**±*0.007*	**0.830**±*0.005*
UNet^{32} - Pretrained	**0.634**±*0.010*	**0.776**±*0.007*	**0.699**±*0.008*	**0.823**±*0.006*
UNet^{64} - RandomInit	**0.649**±*0.005*	**0.787**±*0.003*	**0.713**±*0.011*	**0.832**±*0.008*
UNet^{64} - Pretrained	**0.645**±*0.005*	**0.784**±*0.004*	**0.702**±*0.005*	**0.825**±*0.003*
UNet++^{32} - RandomInit	**0.654**±*0.012*	**0.790**±*0.008*	**0.713**±*0.005*	**0.833**±*0.003*
UNet++^{32} - Pretrained	**0.632**±*0.027*	**0.774**±*0.021*	**0.692**±*0.030*	**0.817**±*0.021*
UNet++^{64} - RandomInit	<u>**0.666**±*0.010*</u>	<u>**0.799**±*0.007*</u>	<u>**0.727**±*0.008*</u>	<u>**0.842**±*0.005*</u>
UNet++^{64} - Pretrained	**0.649**±*0.006*	**0.787**±*0.004*	**0.719**±*0.008*	**0.837**±*0.005*

The exponential in e.g., UNet^{32} or UNet^{64}, denotes the number of output feature maps from the first encoder layer.

4.1.2 Qualitative Results

To get a better view of where the models perform well and where they struggle, a selection of samples is included in Figure 4.1 and 4.2 showing samples that appear less and more difficult to segment accurately, respectively. In the two figures, the original PMSE signal images are shown in the first column and the ground truth mask in the second. The next four columns show the predictions from the four models UNet^{64} - Pretrained, UNet^{64} - RandomInit, UNet++^{64} - RandomInit and UNet++^{64} - Pretrained, respectively. The predictions from all the models from the initial experiment can be found in Figure A-16 to A-19 in the Appendix.

The accurate predictions displayed in Figure 4.1 demonstrate that all models

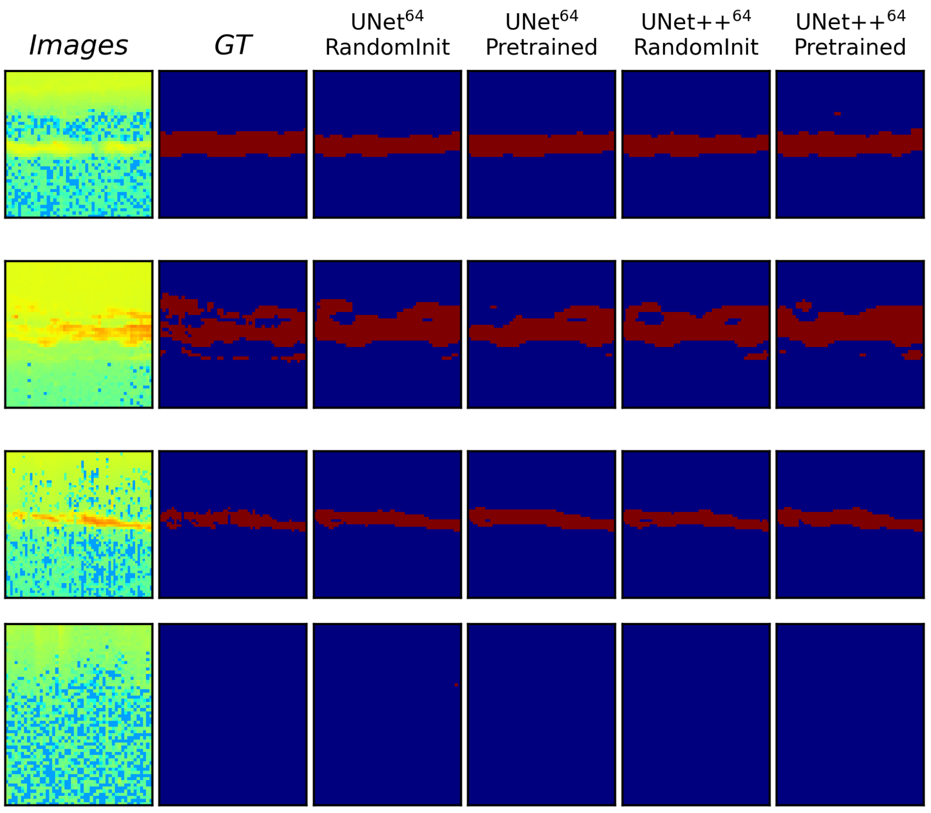

Figure 4.1: Accurate Predictions. Qualitative comparison between UNet^{64} - RandomInit, UNet^{64} - Pretrained, UNet++^{64} - RandomInit and UNet++^{64} - Pretrained showing some of the test samples that the models can segment accurately. The images and the ground-truth labels are shown in the first and second columns, respectively.

can effectively segment PMSE regions, and the predictions are quite consistent across the models used in this study. However, for Image 2 and 3, the models encounter challenges in segmenting smaller PMSE regions and regions with more irregular boundaries. In image 4, where there is only background present, all models can predict this correctly.

The less accurate predictions in Figure 4.2 highlight two images (1 and 2) in which the foreground is empty, yet all models predict PMSE. In Image 1, the predictions vary slightly across models, but all models exhibit a noticeable preference for a few high-valued pixels that are located along a vertical line in the centre of the image. Similarly, in Image 4, all models predict PMSE in a small region located towards the centre-left. Like Image 1, the models appear to focus on a region of slightly higher value compared to the surrounding areas. Interestingly, this region of interest in Image 4 bears some resemblance to other

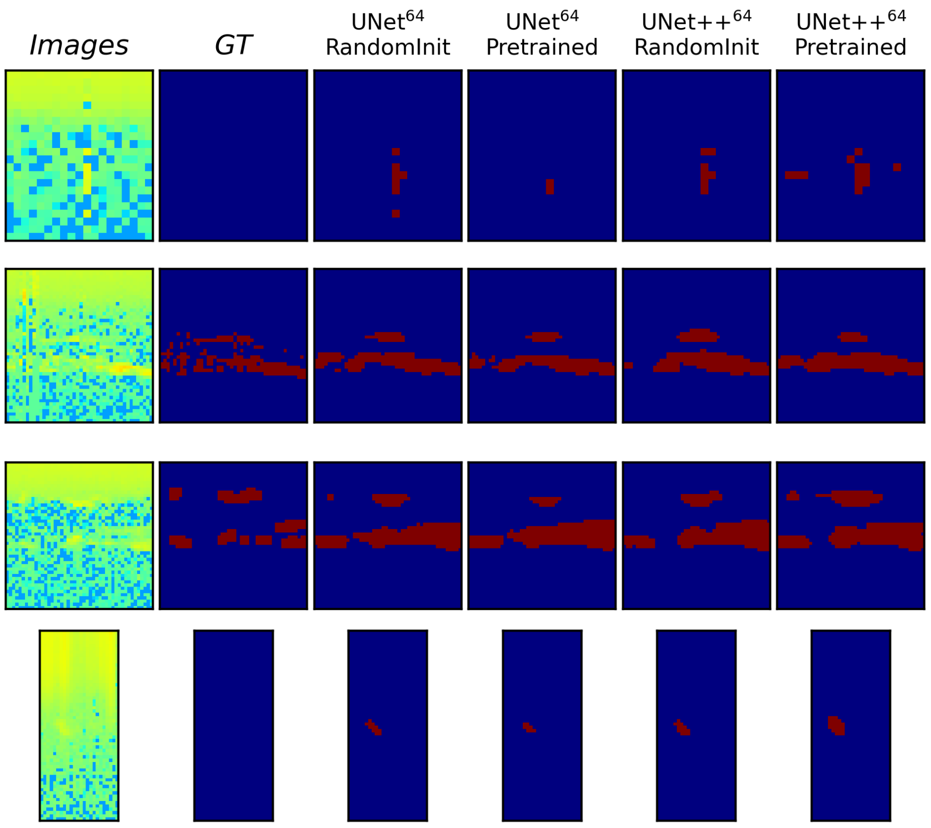

Figure 4.2: Less Accurate Predictions. Qualitative comparison between UNet^{64} - RandomInit, UNet^{64} - Pretrained, UNet++^{64} - RandomInit and UNet++^{64} - Pretrained showing some of the test samples where the models have more difficulties segmenting the PMSE regions accurately. The images and their ground-truth label is shown in the first and second column, respectively.

PMSE regions observed in other images (e.g., Image 3), as they share similar characteristics in terms of pixel values, size, and altitude location.

Upon closer examination of the predictions for Image 2 and 3 in Figure 4.2 and Figure 4.1, it is evident that the models face difficulty in accurately segmenting smaller PMSE regions. Instead, the models tend to group the many smaller regions into larger PMSE regions.

4.1.3 Ablation Analysis of Different Loss Functions

Expanding on the findings of the initial experiment, an ablation analysis is conducted to investigate how alternative loss functions compare to the Dice loss

($\mathcal{L}_{Dice}$). To conduct this experiment, the UNet++[64] model with randomly initialized weights, which exhibited the best performance in the initial experiment (Section 4.1), is used as a baseline. To make a comparison, the same model was trained using four other loss functions: Binary Cross Entropy (BCE) (see Section 2.4.1), Focal Loss (see Section 2.4.3), Dice-BCE Loss (see Section 2.4.5), and Dice-Boundary Loss (see Section 2.4.5), $\mathcal{L}_{BCE}$, $\mathcal{L}_{Focal}$, $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$ and $\mathcal{L}_{Dice} + \mathcal{L}_{B}$, respectively. For $\mathcal{L}_{Focal}$ and $\mathcal{L}_{Dice} + \mathcal{L}_{B}$, some parameters must first be specified.

- $\mathcal{L}_{Focal}$: $\gamma = 2.0$ is set equal to that in the original paper[27] while $\alpha = 0.8$ is chosen such that it is approximately inversely proportional to the foreground frequency.

- $\mathcal{L}_{Dice} + \mathcal{L}_{B}$: The ***increase*** and ***rebalance*** schedule strategies[21] for setting α is used:

  - ***increase*** - $\alpha = 0.01$ initially and is increased by 0.01 every 5 iterations where $\alpha = max(\alpha, 1)$.

  - *Rebalance* - $\alpha = 0.005$ initially and follows a schedule based on the number of iterations as follows,

$$\alpha = \begin{cases} \alpha + 0.005, & \text{if } iter < 100 \\ \alpha + 0.01, & \text{if } 100 \leq iter < 300 \\ \alpha + 0.02, & \text{otherwise} \end{cases} \tag{4.1.1}$$

  This represents a modified schedule for the parameter α, which differs slightly from the *rebalance* strategy used in the original paper[21]. This modification is deemed necessary since the model encounters difficulties when α is increased too rapidly at the beginning. During the initial training phase, α is assigned a smaller value, and its increase rate is set higher to ensure that the loss reaches the final phase, where $\mathcal{L}_{Dice}$ has a very small but non-zero impact while $\mathcal{L}_{B}$ significantly contributes to the final loss.

When α is dynamically changed during training, a problem can arise where the loss might increase, even though the IoU and DSC are improving thus trigging the stopping criteria prematurely. Because the two losses are measures of different objectives and at the same time are weighted dynamically, the total loss might not follow the typical loss learning curve. As such, the IoU metric is used instead of loss as the stopping criteria. From the learning curves in Figure A-12 from training the model with $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ it can be seen for the ***rebalance*** strategy that the loss starts to increases slightly for the validation runs after 150 iterations while the IoU and DSC are still improving. Showing that the loss might not be optimal as a stopping criterion for the $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ loss.

Quantitative Results.

Table 4.3 presents the quantitative findings, indicating that the $\mathcal{L}_{Dice} + \mathcal{L}_{BCe}$ combination exhibits the best performance, but only marginally better than the $\mathcal{L}_{Dice}$ loss. While the distribution-based losses ($\mathcal{L}_{BCe}$ and $\mathcal{L}_{Focal}$) fall short of the regional loss ($\mathcal{L}_{Dice}$) in terms of performance, the combination of $\mathcal{L}_{Dice}$ and $\mathcal{L}_{BCE}$ enhances the performance slightly. Regarding the pairing of $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ and the two distinct α scheduling strategies, the ***increase*** strategy demonstrates better performance. However, regardless of the scheduling strategy, the $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ combination displays a performance decline relative to the top-performing model.

Table 4.3: Quantitative performance of a random initiated UNet++[64] architecture using different loss functions. IoU and DSC are reported for the test and validation set. The best performing model is underlined.

Loss function ($\mathcal{L}$)	Test		Validation	
	IoU↑	DSC↑	IoU↑	DSC↑
$\mathcal{L}_{Dice}$	**0.666**±*0.010*	**0.799**±*0.007*	**0.727**±*0.008*	**0.842**±*0.005*
$\mathcal{L}_{BCE}$	**0.656**±*0.006*	**0.792**±*0.004*	**0.714**±*0.003*	**0.833**±*0.002*
$\mathcal{L}_{Focal}$	**0.647**±*0.003*	**0.786**±*0.002*	**0.695**±*0.002*	**0.820**±*0.001*
$\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$	<u>**0.667**±*0.005*</u>	<u>**0.800**±*0.003*</u>	<u>**0.731**±*0.004*</u>	<u>**0.844**±*0.003*</u>
$\mathcal{L}_{Dice} + \mathcal{L}_{B}$ - *Increase*	**0.662**±*0.013*	**0.797**±*0.010*	**0.722**±*0.011*	**0.838**±*0.007*
$\mathcal{L}_{Dice} + \mathcal{L}_{B}$ - *Rebalance*	**0.650**±*0.011*	**0.788**±*0.008*	**0.703**±*0.012*	**0.825**±*0.008*

- $\mathcal{L}_{dice} + \mathcal{L}_{B}$ is denoted with either *Increase* or *Rebalance* schedule strategy of α.

Qualitative Results.

A visualization of the predictions made by the models trained with the different loss function is shown in Figure 4.3 and 4.4 for the samples that was categorized as accurately and less accurately predicted in the initial experiment, respectively. Here, the $\mathcal{L}_{Focal}$ is not included such that the images in the figures do not get too small. However, all the models with the different loss functions can be seen in Figure A-20 to A-23 in the Appendix.

For the samples in Figure 4.3 and 4.4 it is shown that the predictions are quite similar regardless of the loss function employed. However, there are some minor

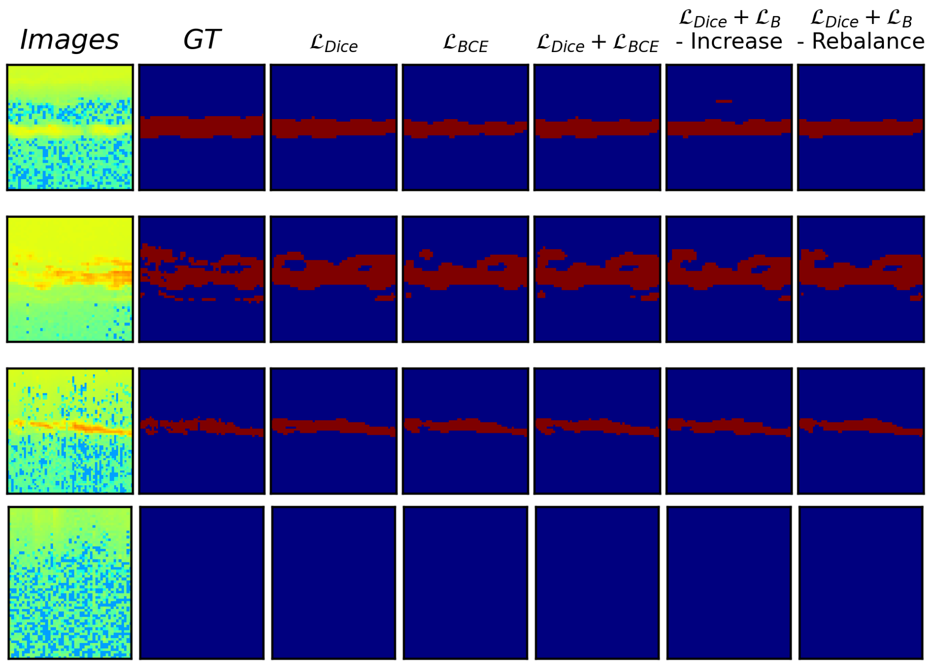

Figure 4.3: Accurate Predictions. Qualitative comparison between the different loss functions $\mathcal{L}_{Dice}$, $\mathcal{L}_{BCE}$, $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$ and $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ (*Increase* or *Rebalance* scheduling) using a random initiated UNet++[64] architecture. The images and their ground truth label is shown in the first and second column, respectively.

variations between the predictions. In Figure 4.3 it is shown in image 1 that the model using $\mathcal{L}_{Dice}$ loss has a prediction that is more in line with the ground-truth than for example the model using $\mathcal{L}_{BCE}$ loss which predicts a narrower PMSE region. The same pattern can be seen from image 2 and 3 in Figure 4.3 and 4.4. However, here the model using $\mathcal{L}_{Dice}$ loss predicts larger PMSE regions compared to the ground-truth and where the models using other losses seem to be more accurate.

Discussion.

The results presented in Table 4.3 and Figures 4.3 and 4.4 indicate that the differences in performance between using different loss functions are minimal, both quantitatively and qualitatively. However, there is a noticeable variation in the predictions made by the different losses, with the regional loss ($\mathcal{L}_{Dice}$) producing larger PMSE regions that encompass the high-valued PMSE signals, compared to the distribution based losses ($\mathcal{L}_{BCE}$ and $\mathcal{L}_{Focal}$) which produce smaller regions. Table 4.3 shows that the distribution-based losses have lower performance than the regional-based loss, which suggests that they may be less suitable for segmenting PMSE signals. It is possible that the limited number

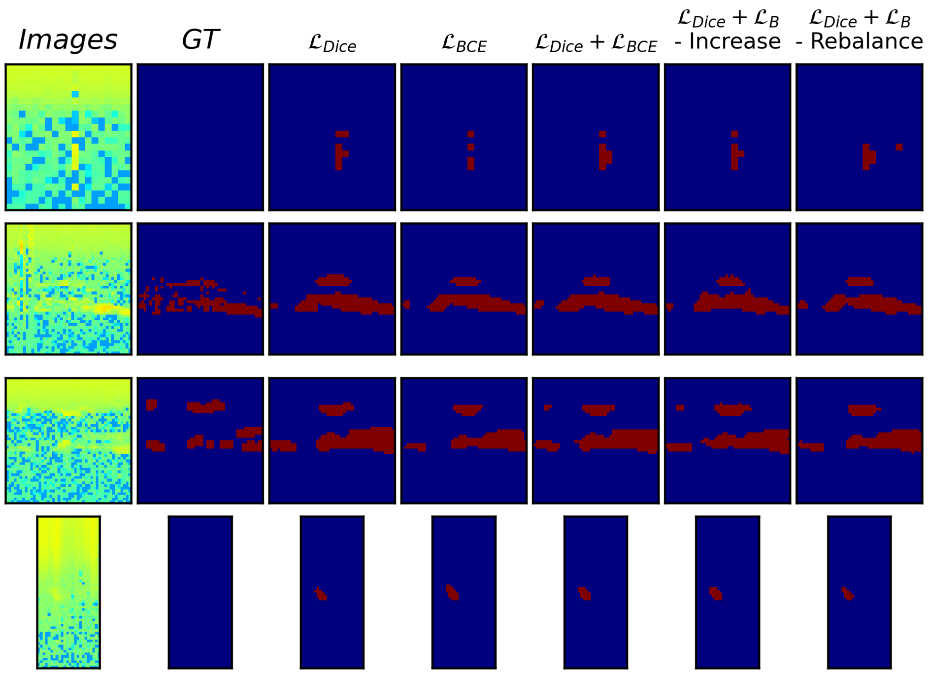

Figure 4.4: Less Accurate Predictions. Qualitative comparison between using the different loss functions: $\mathcal{L}_{Dice}$, $\mathcal{L}_{BCE}$, $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$ and $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ (*Increase* or *Rebalance* scheduling) using a random initiated UNet++^{64} architecture. The images and their ground truth label is shown in the first and second column, respectively.

of training samples may introduce bias in favour of the $\mathcal{L}_{Dice}$ loss, but the evaluation on the validation samples (as shown in Table 4.3) indicates that the performance differences between the models are consistent with those observed for the test samples, suggesting that bias alone may not account for the differences in performance.

Using a combination of $\mathcal{L}_{Dice}$ and $\mathcal{L}_{BCE}$ resulted in the highest IoU and DSC scores, but the improvement compared to using only $\mathcal{L}_{Dice}$ was minimal. The two losses differ in the area around the high-valued PMSE that they include in their predictions. By examining the ground-truth labels in Figures 4.3 and 4.4, it can be seen that there are differences in how detailed the PMSE is classified, with $\mathcal{L}_{Dice}$ being better suited for less detailed ground-truth labels and $\mathcal{L}_{BCE}$ being better suited for more detailed ones. Therefore, combining the two losses may be a good compromise for predicting samples with differing levels of detail in their labels.

Based on the original paper on boundary loss [21], one might assume that using the $\mathcal{L}_{Dice}+\mathcal{L}_{B}$ loss would lead to better identification of a suitable boundary for the foreground PMSE. However, the visual results in Figures 4.3 and 4.4 suggest that this may not be the case, possibly due to the difficulty of the boundary loss in dealing with the many irregularly shaped PMSE regions in the samples. It is however possible that fine-tuning the hyperparameters and adjusting the scheduling of α could address this issue to some extent.

Since the differences in performance among the various loss functions were minimal, this study will use the loss function that demonstrated the highest performance, namely $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$, in further experiments.

4.2 Applying Object- and Image-level Augmentation

In an attempt to enhance the model's performance, we conducted a series of experiments using the ObjectAug method [41] (see Section 4.2). The UNet++^{64} model with randomly initialized weights and $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$ loss is used based on the findings from the previous section. We also examine how simple image-level augmentation methods affect the model performance by conducting a few experiments. First, we used each of the object augmentation techniques listed in Section 2.3.1 individually, namely Horizontal Flip, Vertical Flip, and Contrast Adjustment, with a probability of $p = 0.5$ for invoking the augmentation method. For the Contrast Adjustment method, we randomly selected the adjustment factor c from the range $c \in [0.8, 1.2]$. Additionally, we performed two experiments using combinations of these augmentation methods. The results are shown in Table 4.4, indicating that each individual augmentation method outperforms the baseline. While combining all individual augmentation methods produced a slight improvement over the baseline, it was lower than using the augmentation methods separately. The last row of Table 4.4 shows that the combination of Horizontal Flip and Contrast Adjustment, which had the highest performance increase of the individual augmentation methods, produced a relatively significant performance boost.

Next, two models are trained using the ObjectAug method[41]. Along with training a model using only ObjectAug, another model was trained using the image-level augmentation techniques that demonstrated an increase in performance in Table 4.4. The object-level augmentation methods used in ObjectAug are significantly different from the image-level augmentation methods, indicating that the image-level augmentation techniques could complement the ObjectAug method. The original paper on ObjectAug[41] demonstrated that adding image-level methods to the object-oriented augmentation methods could improve performance.

Table 4.4: Quantitative performance of using different image-level augmentation separately and combined. IoU and DSC are reported for the test and validation set. The best performing model is underlined.

Model - Augmentation	Test		Validation	
	IoU↑	DSC↑	IoU↑	DSC↑
Baseline	**0.667**±*0.005*	**0.800**±*0.003*	**0.731**±*0.004*	**0.844**±*0.003*
w/ Horizontal Flip	**0.682**±*0.010*	**0.811**±*0.007*	**0.742**±*0.008*	**0.851**±*0.005*
w/ Vertical Flip	**0.672**±*0.006*	**0.804**±*0.004*	**0.739**±*0.009*	**0.849**±*0.006*
w/ Contrast Adjust	**0.683**±*0.005*	**0.811**±*0.003*	<u>**0.742**±*0.002*</u>	<u>**0.851**±*0.001*</u>
w/ All Combined	**0.669**±*0.007*	**0.801**±*0.005*	**0.741**±*0.008*	**0.851**±*0.006*
w/ Horizontal Flip & Contrast Adjust	<u>**0.694**±*0.008*</u>	<u>**0.819**±*0.006*</u>	**0.735**±*0.004*	**0.847**±*0.003*

The results are presented in Table 4.5. The baseline model in this table is the same as the one used in Table 4.4, which received no augmentation. The two models trained on the generated dataset in Table 4.5 show significant improvements compared to the baseline model trained without augmentation. Looking at Table 4.5, it is evident that incorporating image-level augmentation into ObjectAug enhances performance. However, the improvement is only marginally better than using image-level augmentation alone.

Table 4.5: Quantitative performance of using no, image-level, object-level and image -and object-level augmentation. IoU and DSC are reported for the test and validation set. The best performing model is underlined.

UNet++^{64} - RandomInit	Test		Validation	
	IoU↑	DSC↑	IoU↑	DSC↑
No Aug	**0.667**±*0.005*	**0.800**±*0.003*	**0.731**±*0.004*	**0.844**±*0.003*
w/ Image-Aug	**0.694**±*0.008*	**0.819**±*0.006*	<u>**0.735**±*0.004*</u>	<u>**0.847**±*0.003*</u>
ObjAug	**0.678**±*0.009*	**0.808**±*0.007*	**0.719**±*0.007*	**0.836**±*0.005*
w/ Image-Aug	<u>**0.701**±*0.010*</u>	<u>**0.824**±*0.007*</u>	**0.730**±*0.003*	**0.843**±*0.002*

Quantitative results.
To visualize the performance of the models performance with the different

augmentation techniques, the same images in that of Figure 4.1 and 4.2 are used. The model predictions can be seen in Figure 4.5 and 4.6 where the column names denote the type of augmentation used. Comparing the model using no augmentation (denoted as No Aug) with the models that employed either image- or object-level augmentation it can be seen that there are some major differences in the predictions for some of the PMSE images. This can be seen in image 4 in Figure 4.5 where it is shown that the models using augmentation incorrectly predict small PMSE regions to the right in the PMSE image and where the model that uses no augmentation predicts this correctly. For image 3 in Figure 4.6 we can see that the models that use augmentation are able to segment the many smaller PMSE regions in an improved manner.

It is noticeable that the models using augmentation are able to predict the PMSE images where the ground-truth label seems more detailed. This is shown in image 2 in Figure 4.6 and image 3 in Figure 4.5 where we can see that compared to the model using no augmentation, the predicted PMSE regions have a more irregular boundary that is more in line with the ground-truth. From the same images, it can also be seen that there are some variations between using image- and object-level augmentation. With object-level augmentation, the regions predicted are a bit narrower and seem to focus more on the high-valued part of the PMSE regions.

The combination of image- and object-level augmentation produces predictions that look like something in between the predictions of the image- and object-level augmentation.

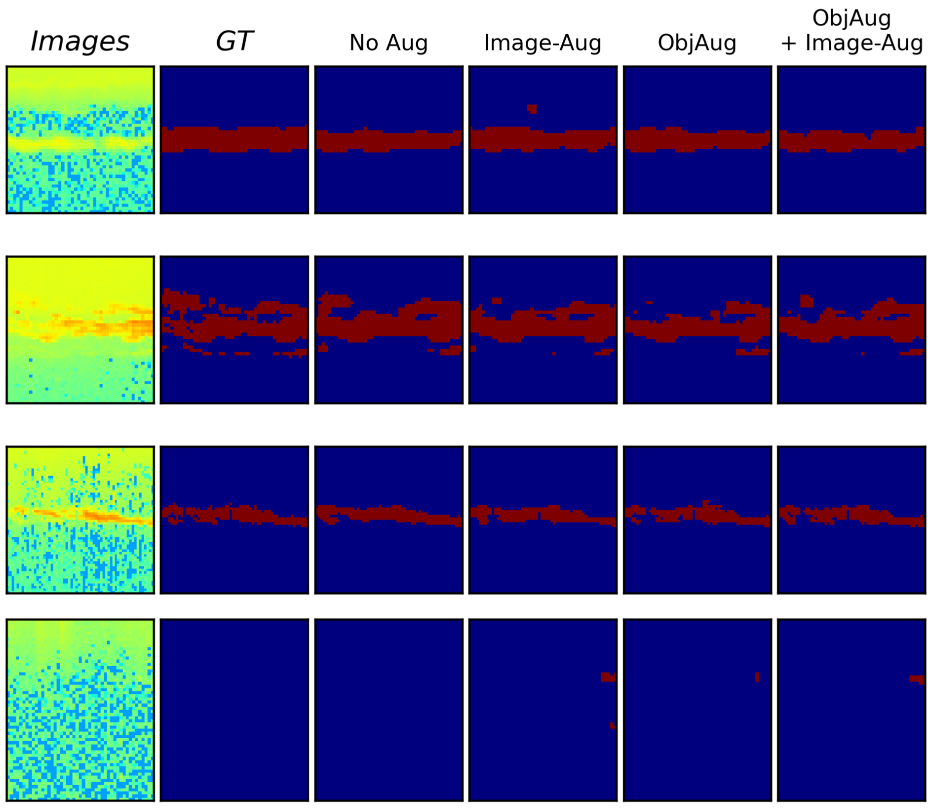

Figure 4.5: Accurate Predictions. Qualitative comparison between using different types of augmentation using a UNet++[64] model with random initiated weights. In the first and second column, the image and ground-truth are shown, respectively. The third column shows the prediction of a model trained with no augmentation. The fourth shows the prediction of a model that used horizontal and contrast adjustment augmentation. The fifth column shows the prediction.

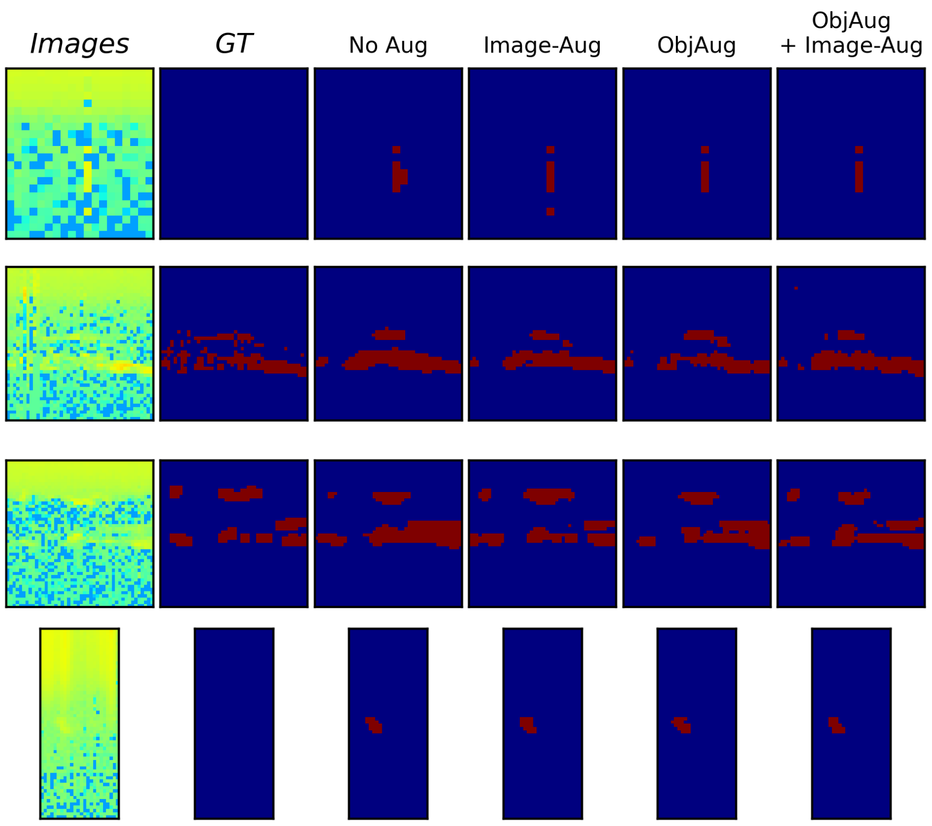

Figure 4.6: Less Accurate Predictions. Qualitative comparison between using different types of augmentation using a UNet++^{64} model with random initiated weights. In the first and second column, the image and ground-truth are shown, respectively. The third column shows the prediction of a model trained with no augmentation. The fourth shows the prediction of a model that used horizontal and contrast adjustment augmentation. The fifth column shows the prediction.

5 Discussion

5.1 Initial Experiment

In the initial experiments, various UNet and UNet++ models were trained, and it was observed that the models exhibited similar performance. However, when comparing models with the same architecture and number of initial parameters, those utilizing pretrained weights in the encoder showed worse performance than their randomly initialized counterparts. For the UNet^{32} and UNet++^{32} models, pretrained weights from the medical imagery domain were used, while weights from the natural image domain were used for the UNet^{64} and UNet++^{64} models. This suggests that other source domains may not be directly applicable to the target domain of the PMSE data through homogeneous domain adaptation, as the feature space of natural and medical imagery differs from that of the PMSE samples.

Interestingly, models utilizing the VGG16[36] backbone trained on the natural image dataset ImageNet[7] showed closer performance to randomly initialized models than those using the backbone from the medical imagery domain. As the backbones are trained on data from different source domains, there may be biases towards specific features in an image. To investigate if there are significant differences between the pretrained models in terms of what is considered important features in the input image, a set of relevance maps were generated using the LRP method from Section 2.5.

Figure 5.1 displays the input image and ground-truth, along with the explained

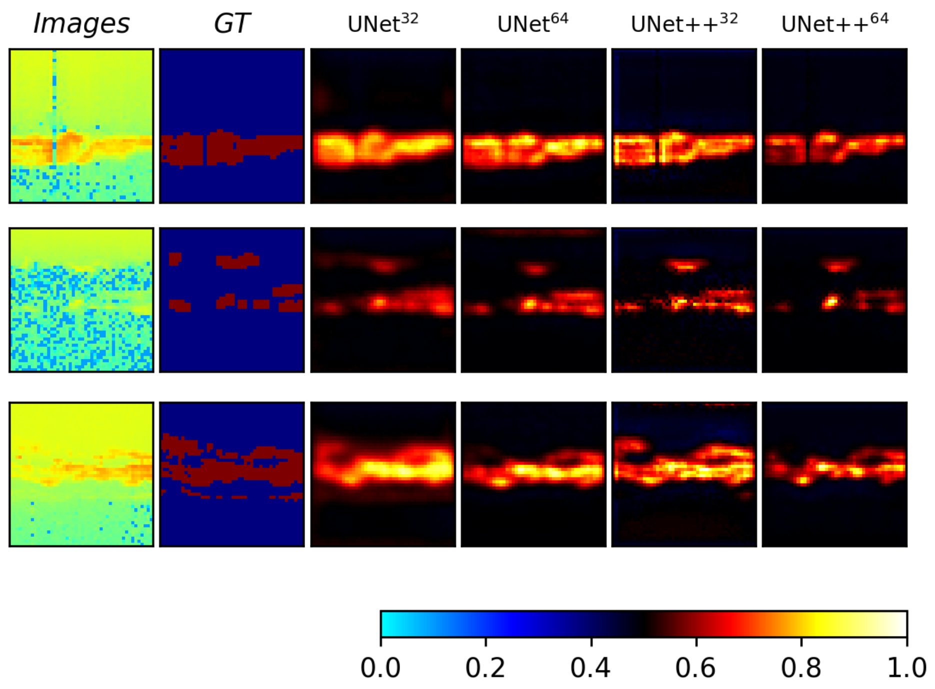

Figure 5.1: The figure depicts four different models utilizing pretrained weights in the encoder and their resulting explained relevance. The UNet^{32}/UNet++^{32} models utilize pretrained weights from the medical image domain, while the UNet^{64}/UNet++^{64} models use pretrained weights from the image domain. The first and second column display the image and ground-truth, respectively. The relevance maps are normalized to be centred around 0.5 (black) to indicate no relevance. Full relevance is represented by a value of 1 (white), while inverse relevance is denoted by a value of 0 (cyan).

relevance from each of the models that used pretrained weights in the encoder. The relevance maps associated with the different pretrained models used in this study were quite similar, as shown in Figure 5.1. Rather, the difference in relevance appeared to be more closely linked to the type of model architecture used. Given that the pretrained models showed worse performance than their randomly initialized counterparts in Figure 4.2, it may be that the difference is due to incorrect model tuning or prediction threshold, rather than biases from other source domains reducing performance.

5.2 Effects of Augmentation

Image-Level Augmentation Effect.
Using image-level augmentation improved model performance, as seen in Table 4.4. This result is not surprising and is consistent with other studies[41, 2, 37] that have shown the positive effect of augmentation. However, the data used in this study differs slightly from more common cases with natural or medical images. Although the image-level augmentation used could be considered conservative, it is argued that this is a good approach, given that PMSE regions occur at specific altitudes and with a pattern that elongates horizontally.

Table 4.4 indicates that vertical flipping of the image shows some improvement in performance compared to using no augmentation. However, it is also apparent from the figure that excluding vertical flip augmentation from the combined augmentation results in improved performance. This suggests that augmentation that changes the PMSE image in a highly unnatural way, such as rotating or flipping the image so that PMSE and the background occur in improbable locations and could lead to worse performance as the model learns patterns and context that are different from the natural occurrence of foreground PMSE and background noise.

Object-level Augmentation Effect.
Table 4.5 shows that using object-level augmentation resulted in worse performance than using only image-level horizontal and contrast adjustment augmentation. However, when the same image-level augmentation was combined with object-level augmentation, there was a slight improvement in performance. Although the increase in performance was marginal, the qualitative results in Figure 4.5 and 4.6 demonstrate that the model utilizing object-level augmentation (*ObjAug*) predicted regions differently. It appeared to be more aware of smaller PMSE regions and able to predict irregular shapes that fit better for some samples (further discussed in Section 5.3).

Using object-level augmentation creates a problem in which the boundary between PMSE and the background is altered. The cause of this effect is not fully understood, but it is believed that the inpainting of images impacts how the models learn the boundary between background and PMSE. During the inpainting process, a slightly larger region around the PMSE is removed to avoid unlabelled PMSE pixels. The inpainting model then fills in this region before replacing the PMSE at its original position. However, as illustrated in Figure 5.3, where no augmentation is applied to the PMSE regions, the inpainted region around the PMSE can be significantly different from the original. The

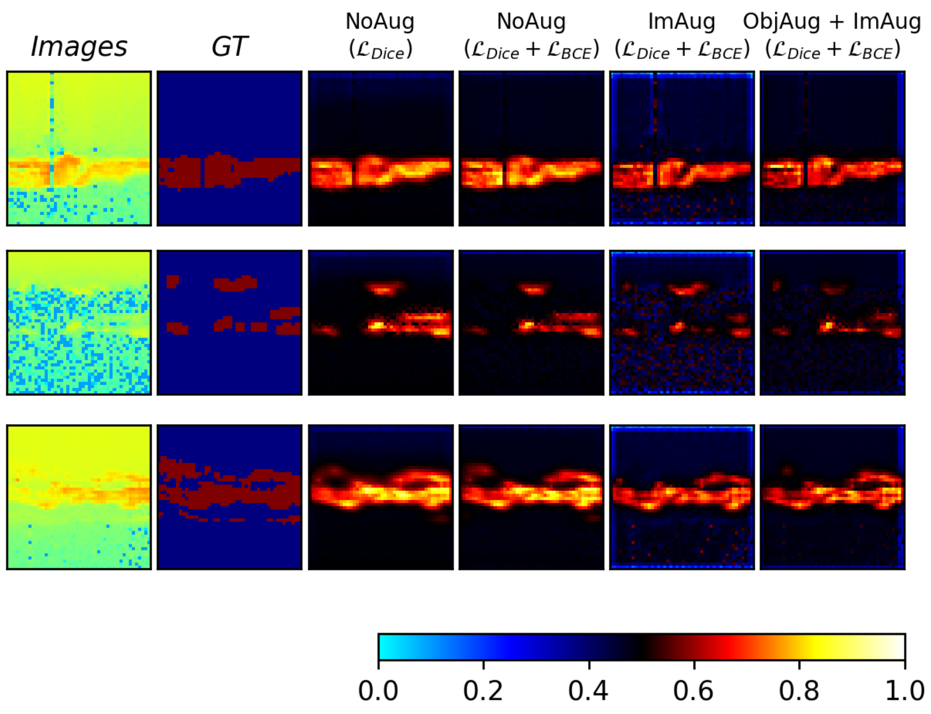

Figure 5.2: The figure displays relevance maps generated from four different models, all utilizing a UNet++[64] architecture initialized randomly. The columns indicate the type of augmentation and loss function utilized: *ObjAug* represents object-level augmentation, *ImAug* denotes image-level augmentation, and *NoAug* indicates no augmentation was employed. The first and second column present the image and ground-truth, respectively. The relevance maps are normalized to be centred around 0.5 (black) to indicate no relevance. Full relevance is represented by a value of 1 (white), while inverse relevance is denoted by a value of 0 (cyan).

result is that the gradient between PMSE and the inpainted background is often increased, possibly influencing the model towards regions with large contrasts or gradients.

Overall Effect of Augmentation.

In order to compare the effects of using and not using data augmentation, we utilized LRP (described in Section 2.5) to determine which features in the input image were deemed relevant by the models. We observed that the models trained with augmentation identified slightly different important features compared to those trained without augmentation. Additionally, the relevance maps of the models trained with augmentation had lower overall relevance compared to those trained without augmentation, indicating a potential bias towards certain areas of the input in the latter case.

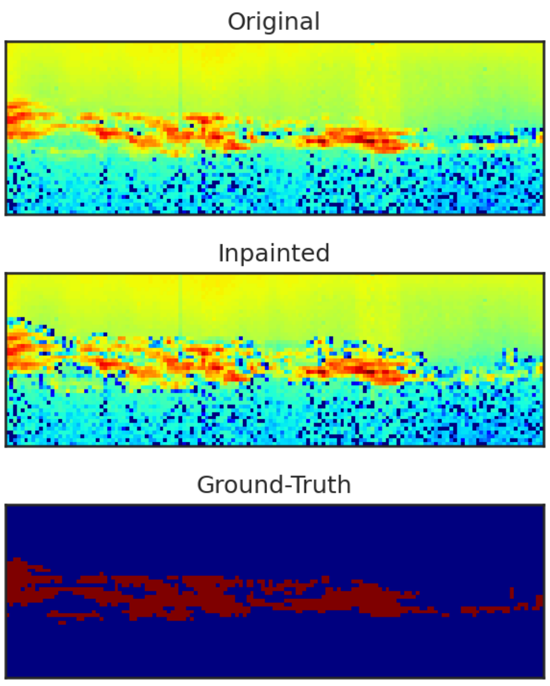

Figure 5.3: Illustration of the different boundary between foreground and PMSE that the inpainted image creates compared to the original image.

Furthermore, while it is expected that the models perform better on the validation set due to the stopping criteria being based on validation loss, our evaluation (as shown in Tables 4.4 and 4.5 in Section 4.2) revealed that the best performing models on the test set did not necessarily have the highest score on the validation set. We also found that the difference between test and validation performance was lower for models trained with data augmentation, suggesting that these models may be better at generalizing to new cases.

Finally, although it is shown that augmentation helps to improve model performance, different augmentation methods, both image- and object-level, should be tested more rigorously in future studies for a more optimal combination of augmentation methods. Also, the inpainting model used with the ObjectAug[41] (see Section (2.3.2)) method is a part that could be improved to generate more natural background noise in the removed PMSE regions.

5.3 Label Inconsistency

Label inconsistency is a problem for supervised deep learning models that may lead to inaccurate and unreliable results. As the author of this thesis is no expert in the field of PMSE, the consistency and precision of the different PMSE labels will not be discussed. However, there is possible to show through several examples of PMSE images and their corresponding ground-truth that there might be inconsistencies in the labelling. First, it is argued that manual labelling of data is highly custom to human factors. Data labelled at different times and by different persons can be a contributing factor to label inconsistency.

In the PMSE dataset, it is seen that some samples have a ground-truth that seems to be more focused on small regions of higher value to that of its neighbouring values while others include more of the lower valued region around the high-valued regions. In Figure 5.4 an example of the mentioned difference is shown. For ease of explanation, the mentioned difference in labels is divided into two categories: *Granular* and *Coarse* labelled samples, respectively. It must be noted that either one of the two categories is concluded as more consistent than the other.

From the histograms to the left of each image and ground-truth pair it can be seen that the *Coarse* labelled PMSE includes lower values than that of the *Granular* labelled PMSE. In the two top samples (sample 1 & 6) the distribution of PMSE labelled values are shown to include some minimum values i.e., an equivalent electron density of 6. Compared to the *Granular* labelled PMSE (sample 10 & 12) the distribution of PMSE labelled values shows that there are only values around an equivalent electron density $\gtrsim 10$.

The difference in the PMSE labelled values shows that there is a possibility that there are some inconsistencies which may confuse the model to make inaccurate or to some degree unreliable results. In many of the results in Chapter 4 it was shown that many models tend to segment larger regions where the ground-truth has more *Graular* labelled PMSE regions.

To further show the issue of inconsistent PMSE labelling and the effect it might have on how models learn, the samples that are seen as *Granular* and *Coarse* labelled are divided into two separate classes. In Figure 5.5 the IoU and DSC metric performance of four different models are shown for the *Granular* and *Coarse* classes. Note that all models are of the UNet++[64] architecture with random initiated weights. From the figure, it can be seen that for the two models that used no augmentation the performance on the *Coarse* class is slightly higher IoU and DSC score than for the *Granular* class. What is also noticeable and previously noted in Section 4.1.3 is that the model using $\mathcal{L}_{Dice}$ had a tendency to segment bigger regions than models using other loss functions such as the $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$. As seen from Figure 5.5, the model with $\mathcal{L}_{Dice}$ has a higher IoU and DSC for the *Coarse* class and is thus in line with what could be seen from the qualitative results in Section 4.1.2.

The two models that utilize augmentation have a significant increase for the

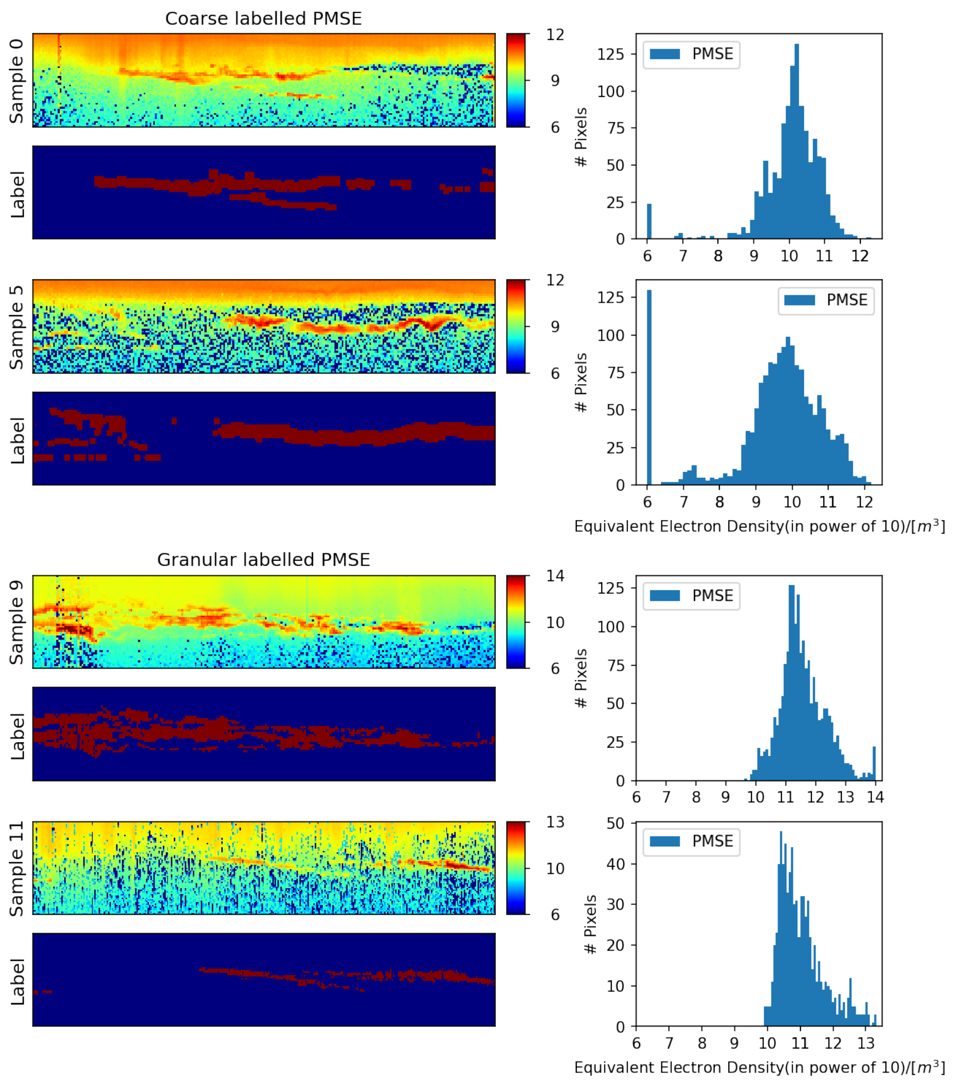

Figure 5.4: Analysis of different labels in the dataset. The figure shows the sample images with the corresponding ground-truth label and a histogram showing the values of the image where the ground-truth indicates PMSE (red). The images and histograms under *Coarse* labelled PMSE shows two samples that are seen as more coarse labelled. The images and histograms under *Granular* labelled PMSE shows two samples that are seen as more granular labelled.

Granular class as shown in Figure 5.5. Both models were seen to have quite similar performance in table 4.5 but as shown in Figure 5.5 they differ when evaluating them based on the *Coarse* and *Granular* classes. The model that only used image-level augmentation had improved performance for both classes, while the model with object-level augmentation could be seen to have significantly better performance on the *Granular* class but worse on the *Coarse* class. As discussed in Section 5.2 on the Object-level Augmentation Effect, the model trained using object-level augmentation tended to predict smaller and more *granular* regions in a better way. This is in line with the results from Figure 5.5 that shows a significant performance increase for the *Granular* labelled samples. As discussed in Section 5.2, there is a difference between the foreground and background boundary in the inpainted images compared to the original images as seen in Figure 5.3 which might make the models trained with object-level augmented data more aware of large gradient changes between foreground and background which matches better from the *Granular* labelled PMSE.

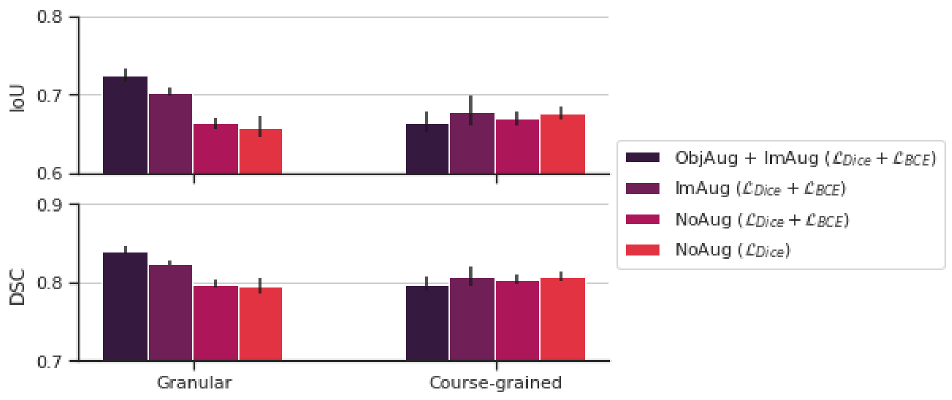

Figure 5.5: Performance comparison between four different models using *Granular* and *Coarse* labelled PMSE as classes. All models use the random initiated UNet++^{64} architecture. The type of augmentation and loss function used during training is listed in the legend. *ObjAug* denotes object-level augmentation, *ImAug* denotes image-level augmentation and *NoAug* denotes no augmentation.

5.4 PMSE Regions Surrounded By Ionospheric Noise

In some of the samples in the dataset, there are PMSE regions that are surrounded by ionospheric noise. The ionospheric noise often has a higher value than the background noise (see Figure 3.1). It is often easier for the model to spot PMSE surrounded by low-valued background noise. The ionospheric noise however usually has a value much closer to PMSE and tends to make it more difficult for the models to segment the PMSE. An illustration of this is shown in the top row of Figure 5.6, where a region that the models struggle with is highlighted by the black square. Another thing to notice in the second row of Figure 5.6 is that when the high-valued PMSE regions are removed and inpainted the difficult region is segmented in a better way by the models. It could be the case that the models are biased toward PMSE with higher values as compared to PMSE with lower values with respect to the ionospheric background. This is something that could be investigated in future studies.

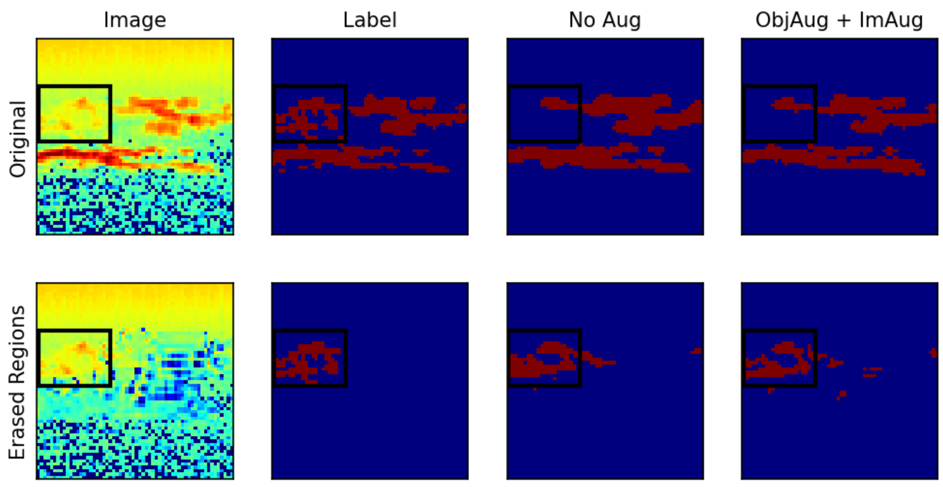

Figure 5.6: The figure shows the difficulties that the models have with predicting the highlighted area (black square). The first and second column shows the image and ground-truth, respectively. And the third and fourth column shows the prediction of the models trained without and with random erasing, respectively. The image in the first row shows the original PMSE sample and the second row shows the same image is shown but where all PMSE regions except one are removed.

To reduce the problem of poorly segmented PMSE regions surrounded by ionospheric noise, it could be possible that using the object-level augmentation and removing some PMSE regions at random could alleviate this problem. As such, we conduct a small experiment where we randomly erase PMSE regions. For the experiment, we use the random initiated UNet++^{64} architecture with image- and object-level augmentation that was seen to have the best performance from the experiments in Section 4.2 and Table 4.5. PMSE regions are erased with a probability of = 0.5. To compensate for the number of lost PMSE regions due to the random erasing, we increase the number of pre-computed augmented samples by 50% compared to the model that does not use erasing.

The results are shown in table 5.1 and as it can be seen the model performance is *lower* on the test set when using random erasing. However, when looking at the second row in Figure 5.7, the new model is slightly better at segmenting out the weak PMSE region when the other PMSE regions are removed. Also, it can be seen that the model using random erasing is less interested in the noise generated by the inpainting model. However, from the original sample in the first row in Figure 5.7, where none of the regions is removed, it can be seen that random erasing does not contribute to any significant improvement.

Table 5.1: Quantitative performance of using object-level augmentation with and without random erasing of PMSE regions. IoU and DSC are reported for the test and validation set. The best performing model is underlined.

Erasing	Test		Validation	
	IoU↑	DSC↑	IoU↑	DSC↑
✗	**0.701**±*0.010*	**0.824**±*0.007*	**0.730**±*0.003*	**0.843**±*0.002*
✓	**0.686**±*0.007*	**0.813**±*0.005*	**0.748**±*0.003*	**0.856**±*0.002*

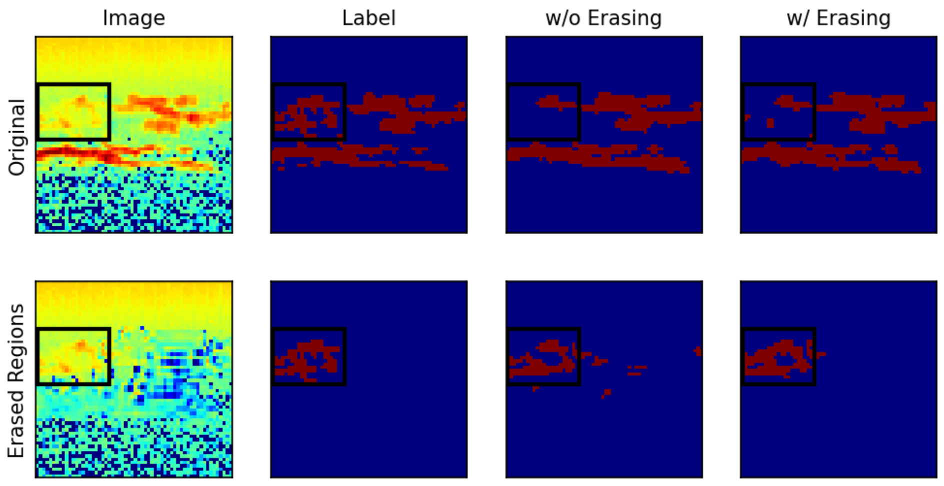

Figure 5.7: The figure shows the effect that erasing PMSE has on model prediction. The first and second column shows the image and ground-truth, respectively. And the third and fourth column shows the prediction of the models trained without and with random erasing, respectively. The image in the first row shows the original PMSE sample and the second row shows the same image is shown but where all PMSE regions except one are removed. The region of interest is highlighted in the black square.

5.5 Different Altitude Resolution

The resolution of the samples in the dataset varies, with consistent time resolution but significant variation in altitude resolution, ranging from 22 to 60 pixels. All samples are limited to a 75 to 95 km altitude region. The reason for the variation is not fully understood, but it is thought to be related to the use of different sample codes (*manda* or *arcd*), except the two samples with an altitude resolution of 22, which both use the *arcd* sample code. Samples with altitude resolutions of 48 typically use the *manda* sample code, as do those with resolutions of 58 and 60, although it is possible that different altitude resolutions using the same sample code may be linked to the year of observation. In Table A-1, samples 1 to 10 have an altitude resolution of 48 (excluding samples 2 and 3, which use the *arcd* sample code), samples 11 to 14 have a resolution of 60, and samples 15 to 18 have a resolution of 58, illustrating the difference between altitude resolution and years of observation.

The impact of the different altitude resolutions on models is not well understood. When the data is divided into classes based on altitude resolution, models appear to perform better on samples with lower altitude resolution, such as 22 and 48. However, because of the small sample size, it is unclear whether this accurately reflects the true nature of the problem, particularly given that the complexity of the test set samples may vary.

The variation in altitude over different years of EISCAT data is an important issue that should be addressed in future research.

6 Conclusion

The goal of this study was to implement a deep learning model that could segment PMSE signal from EISCAT VHF radar data to be used in further analysis and research of changes in the 75 to 95 km altitude range of the mesosphere.

For the objective of segmenting the PMSE signals, a UNet and UNet++ architecture was used. A set of experiments was conducted to find optimal network sizes, initial weight settings and loss functions. Further, different augmentation methods were tested to address the potential problem with few PMSE signal samples and improve model predictions. In the evaluation of the models quantitative performance, the metrics IoU and DSC were reported.

From the results it was found that random initiated weights were best suited for the segmentation task, outperforming the use of pre-trained encoder weights from other source domains. Further, it was shown that the choice of loss functions had only a small impact on the performance metrics although it could be shown from the qualitative results that the use of different loss functions had some impact on how the model segmented different PMSE regions. Through applying both image- and object-level augmentation the best model performance was reached but only slightly better than when only using image-level augmentation. Through the use of LPR it was seen that although some models perform better on the evaluation metrics, the regions of relevance to the models were quite similar. Indicating that the models, regardless of architecture variations, can spot the most important features of the PMSE signal images.

Further, it was noted that there exist inconsistencies in the labelling of the PMSE signals. By dividing the inconsistencies into two categories: *Coarse* and *granular*, it could be seen that the models using object-level augmentation had a significantly better performance on the *granular* labelled PMSE signal images. Also, it was seen that the PMSE signal dataset consisted of samples with different altitude resolutions that can affect the training process.

For future studies, the model can be used over a larger EISCAT VHF radar dataset to investigate changes in the altitude range of 75 to 95 km in the mesosphere. Also, the difference between PMSE signal labelling and PMSE signal altitude resolution could be investigated.

Bibliography

[1] Christopher J. Anders, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Software for dataset-wide xai: From local explanations to global insights with Zennit, CoRelAy, and ViRelAy. *CoRR*, abs/2106.13200, 2021.

[2] Enes Ayan and Halil Murat Ünver. Data augmentation importance for classification of skin lesions via deep learning. In *2018 Electric Electronics, Computer Science, Biomedical Engineerings' Meeting (EBBT)*, pages 1–4, 2018.

[3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140–e0130140, 2015.

[4] Roger Bourne. *Contrast Adjustment*, pages 109–135. Springer London, London, 2010.

[5] Yuri Boykov, Vladimir Kolmogorov, Daniel Cremers, and Andrew Delong. An integral solution to surface evolution pdes via geo-cuts. In *Computer Vision – ECCV 2006*, Lecture Notes in Computer Science, pages 409–422, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] Yuri Boykov, Vladimir Kolmogorov, Daniel Cremers, and Andrew Delong. An integral solution to surface evolution pdes via geo-cuts. In *Computer Vision – ECCV 2006*, Lecture Notes in Computer Science, pages 409–422, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[8] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[9] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. Data augmentation using gans. *CoRR*, abs/1904.09135, 2019.

[10] W. L. Ecklund and B. B. Balsley. Long-term observations of the arctic mesosphere with the mst radar at poker flat, alaska. *Journal of Geophysical Research: Space Physics*, 86(A9):7775–7780, 1981.

[11] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[13] T. L. Gunnarsdottir, A. Poggenpohl, I. Mann, A. Mahmoudian, P. Dalin, I. Haeggstroem, and M. Rietveld. Modulation of polar mesospheric summer echoes (pmses) with high-frequency heating during low solar illumination. *Annales Geophysicae*, 41(1):93–114, 2023.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[16] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. Unet 3+: A full-scale connected unet for medical image segmentation, 2020.

[17] Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, oct 2020.

[18] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659, 2017.

[19] Dorota Jozwicki, Puneet Sharma, and Ingrid Mann. Investigation of polar mesospheric summer echoes using linear discriminant analysis. *Remote Sensing*, 13(3), 2021.

[20] Dorota Jozwicki, Puneet Sharma, Ingrid Mann, and Ulf-Peter Hoppe. Segmentation of pmse data using random forests. *Remote Sensing*, 14(13), 2022.

[21] Hoel Kervadec, Jihene Bouchtiba, Christian Desrosiers, Eric Granger, Jose Dolz, and Ismail Ben Ayed. Boundary loss for highly unbalanced segmentation. *Medical Image Analysis*, 67:101851, jan 2021.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[23] Ralph Latteck, Toralf Renkwitz, and Jorge L. Chau. Two decades of long-term observations of polar mesospheric echoes at 69°n. *Journal of Atmospheric and Solar-Terrestrial Physics*, 216:105576, 2021.

[24] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[25] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets, 2014.

[26] Markku S. Lehtinen and Asko Huuskonen. General incoherent scatter analysis and guisdap. *Journal of Atmospheric and Terrestrial Physics*, 58(1):435–452, 1996. Selected papers from the sixth international Eiscat Workshop.

[27] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[28] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. *CoRR*, abs/1804.07723, 2018.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[30] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, pages 193–209. 09 2019.

[31] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, 65:211–222, 2017.

[32] M. Rapp and F.-J. Lübken. Polar mesosphere summer echoes (pmse): Review of observations and current understanding. *Atmospheric Chemistry*

and Physics, 4(11/12):2601–2633, 2004.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[35] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556,* 09 2014.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[38] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sébastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *CoRR*, abs/1707.03237, 2017.

[39] Wikipedia contributors. Jaccard index — Wikipedia, the free encyclopedia, 2023. [Online; accessed 11-March-2023].

[40] Ma Yi-de, Liu Qing, and Qian Zhi-bai. Automated image segmentation using improved pcnn model based on cross-entropy. In *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, pages 743–746, 2004.

[41] Jiawei Zhang, Yanchun Zhang, and Xiaowei Xu. Objectaug: Object-level data augmentation for semantic image segmentation. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[42] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. *CoRR*, abs/1807.10165, 2018.

Appendix

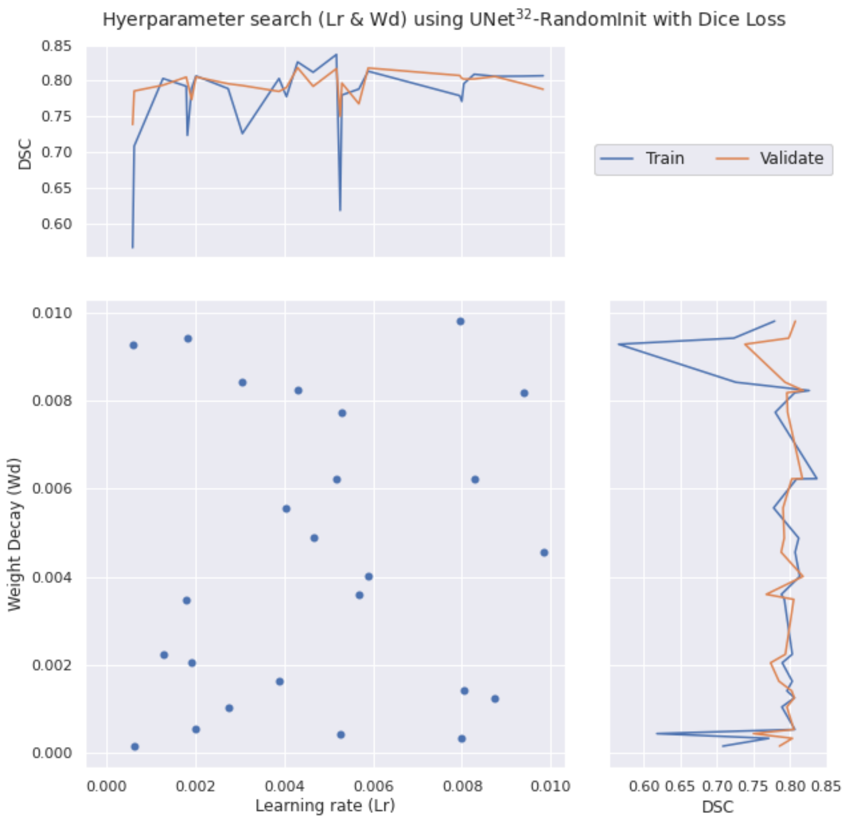

Figure A-1: Hyperparameter search for learning rate and weight decay using dice loss with UNet^{32} architecture with randomly initiated weights.

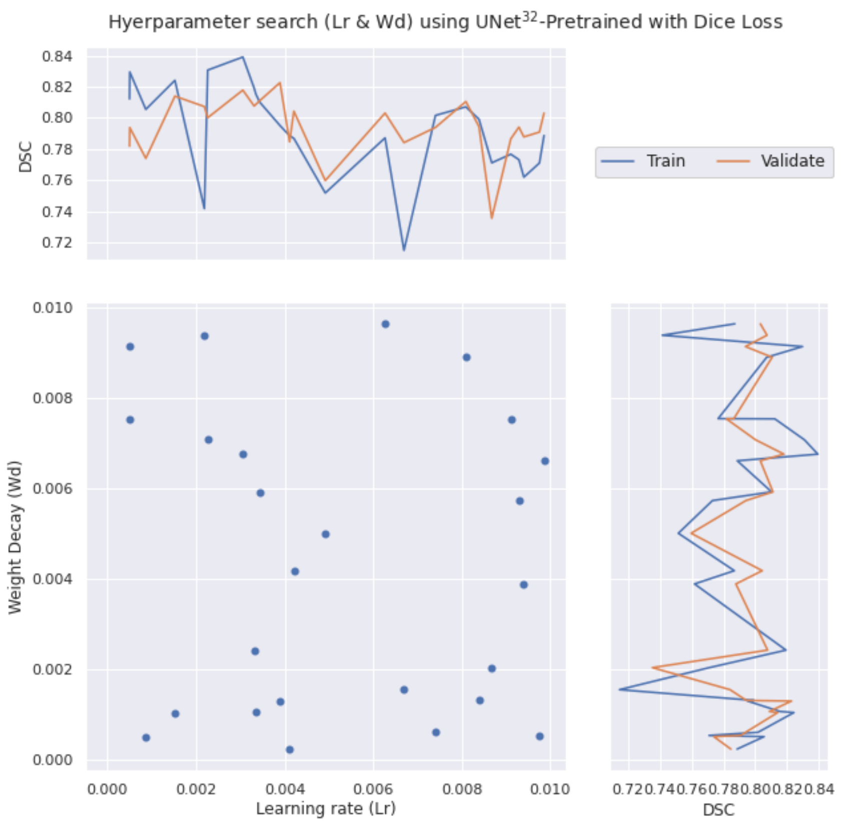

Figure A-2: Hyperparameter search for learning rate and weight decay using dice loss with UNet^{32} architecture with pretrained encoder.

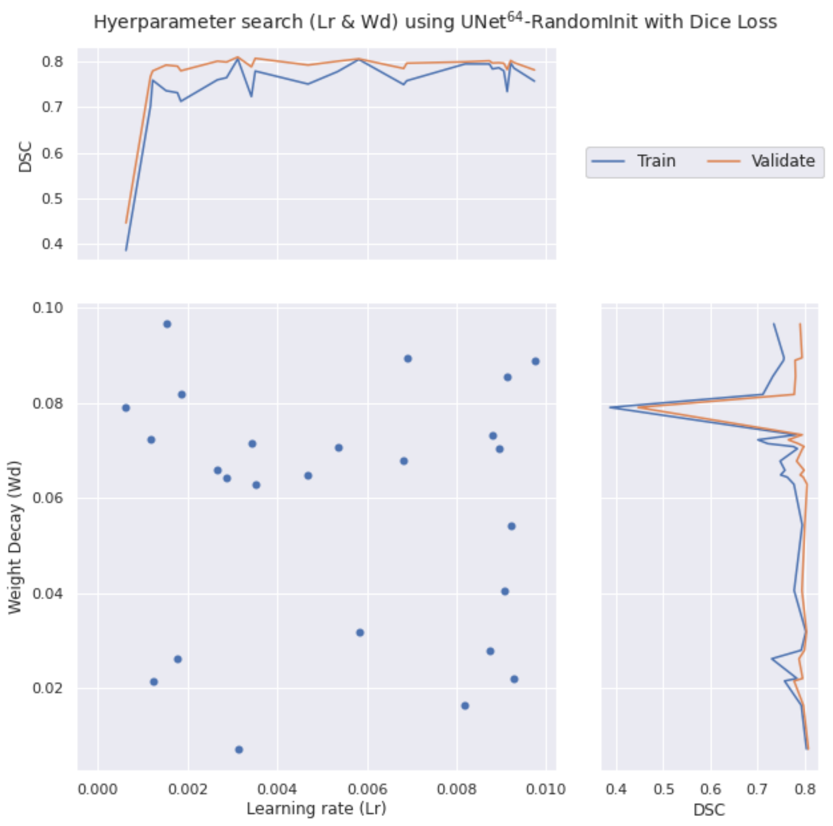

Figure A-3: Hyperparameter search for learning rate and weight decay using dice loss with UNet^{64} architecture with pretrained encoder.

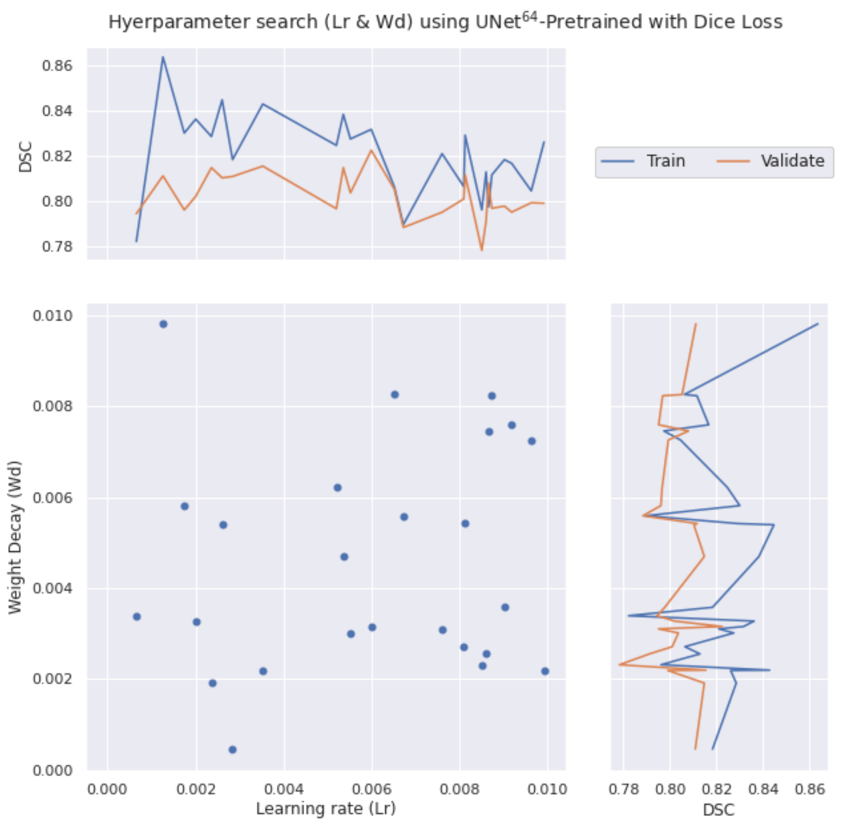

Figure A-4: Hyperparameter search for learning rate and weight decay using dice loss with UNet^{64} architecture with pretrained encoder.

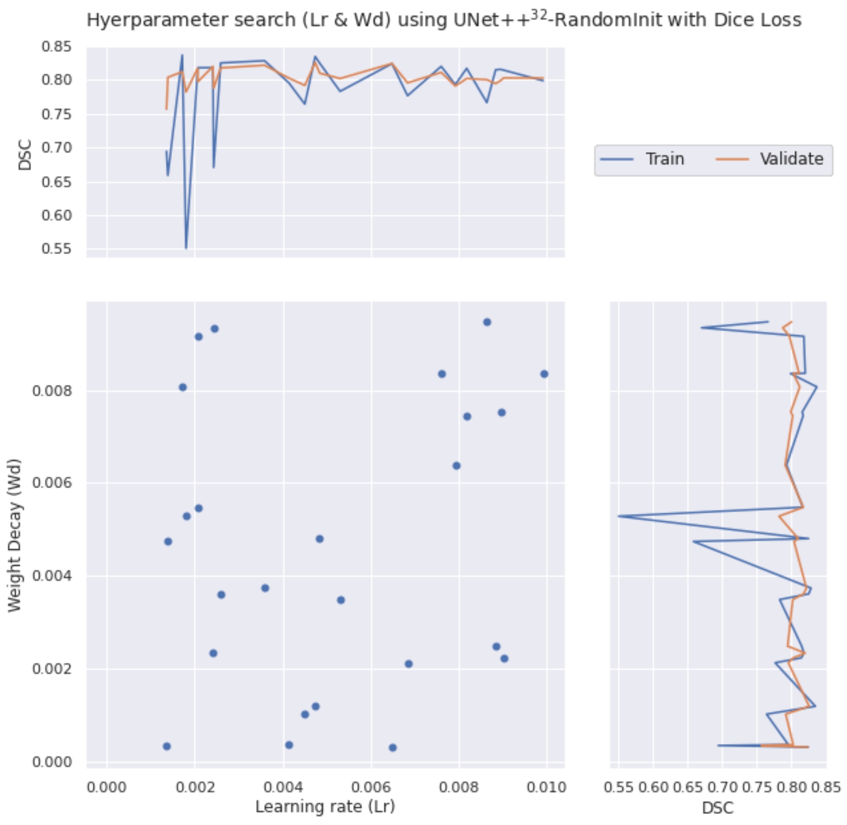

Figure A-5: Hyperparameter search for learning rate and weight decay using dice loss with UNet++^{32} architecture with randomly initiated weights.

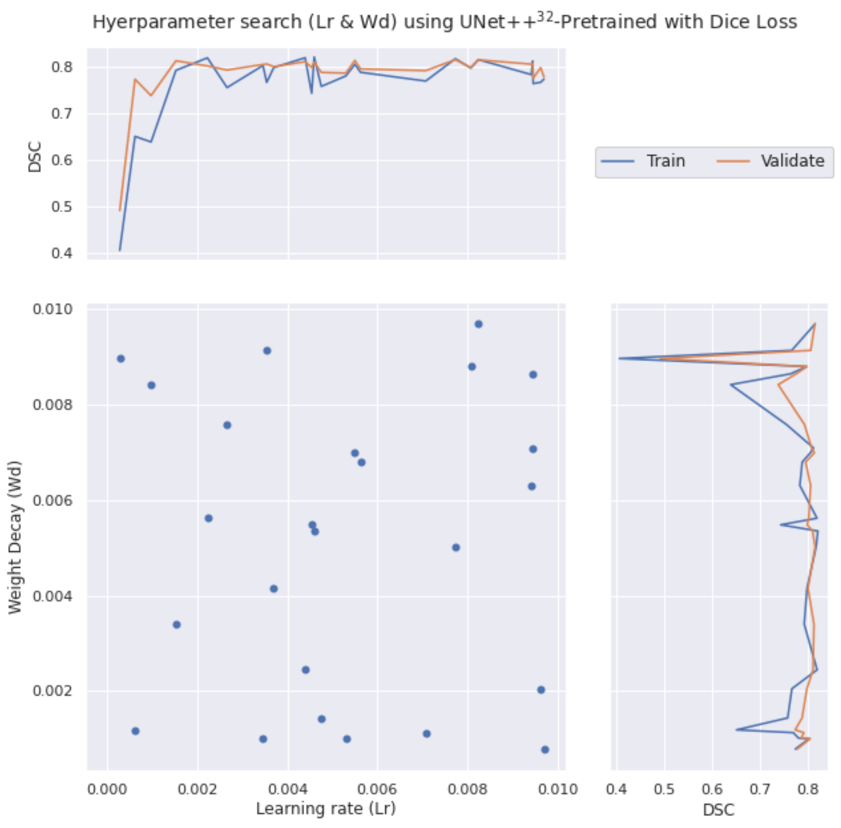

Figure A-6: Hyperparameter search for learning rate and weight decay using dice loss with UNet++^{32} architecture with pretrained encoder.

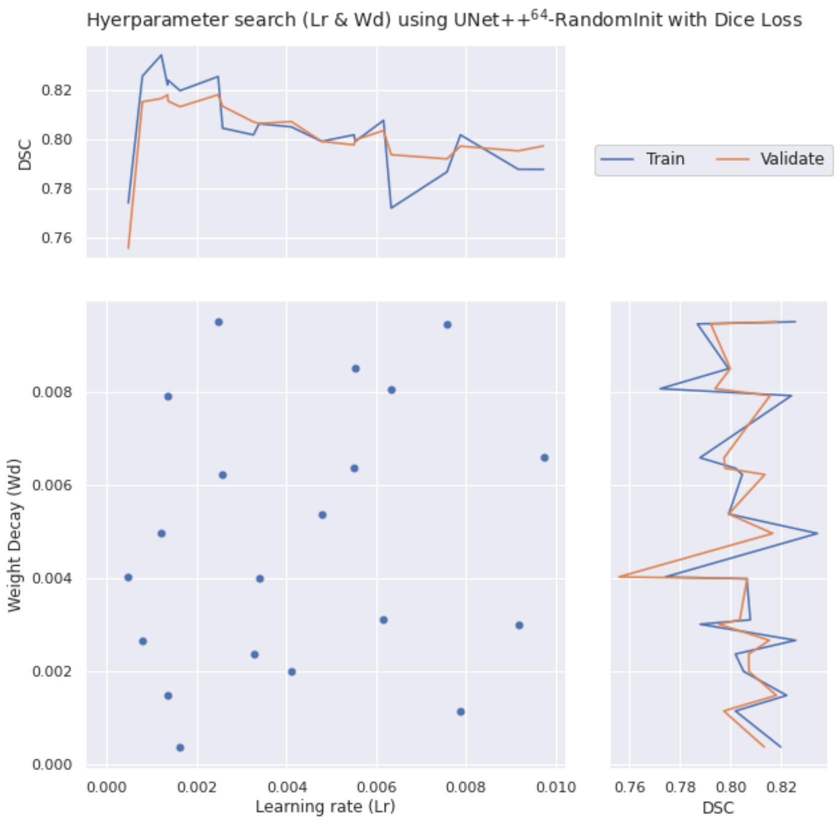

Figure A-7: Hyperparameter search for learning rate and weight decay using dice loss with UNet++^{64} architecture with randomly initiated weights.

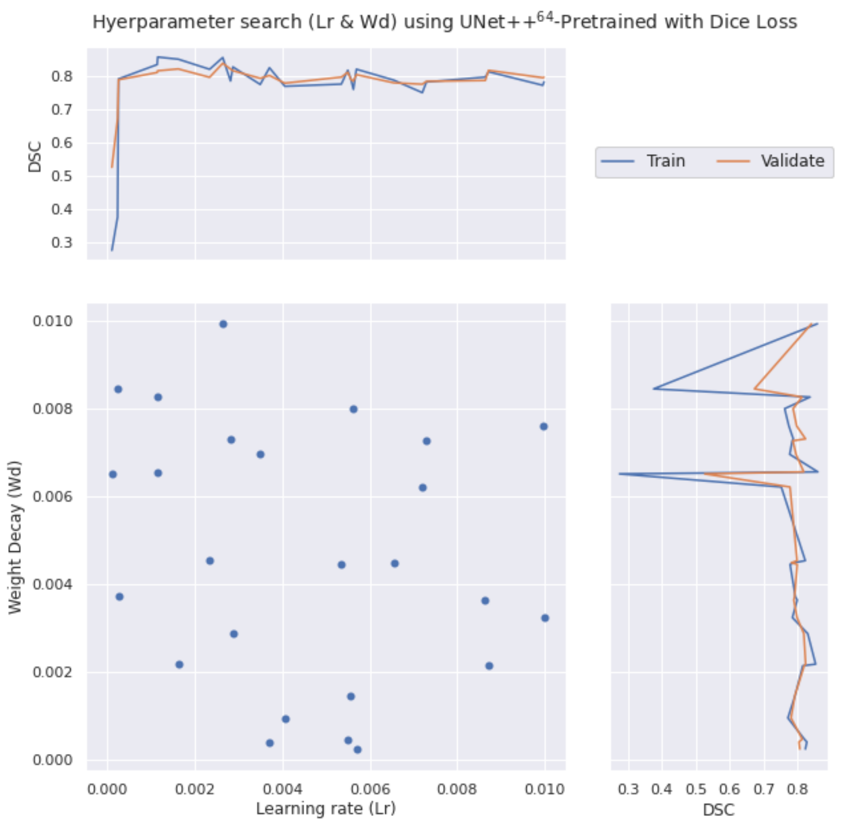

Figure A-8: Hyperparameter search for learning rate and weight decay using dice loss with UNet++^{64} architecture with pretrained encoder.

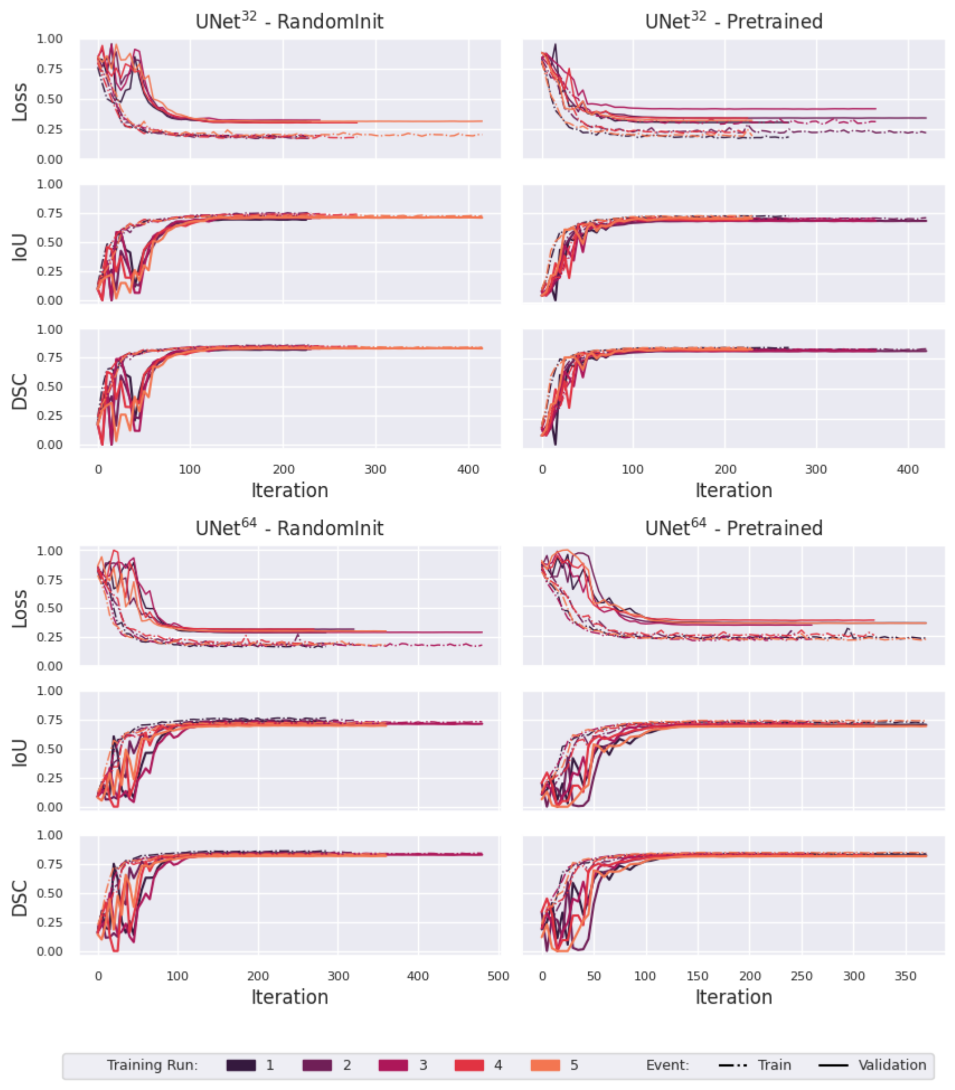

Figure A-9: Learning curves for UNet models trained in the initial experiment in Section 4.1 and Table 4.2.

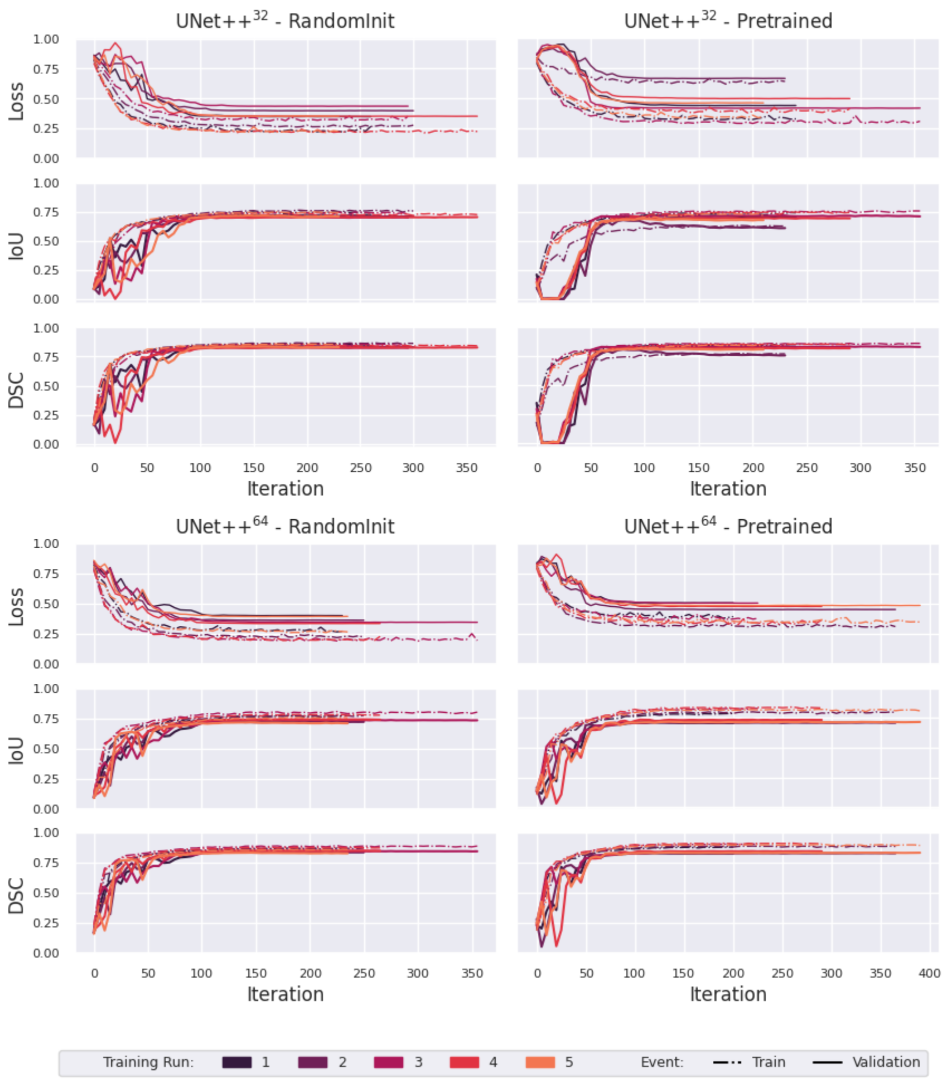

Figure A-10: Learning curves for UNet++ models trained in the initial experiment in Section 4.1 and Table 4.2.

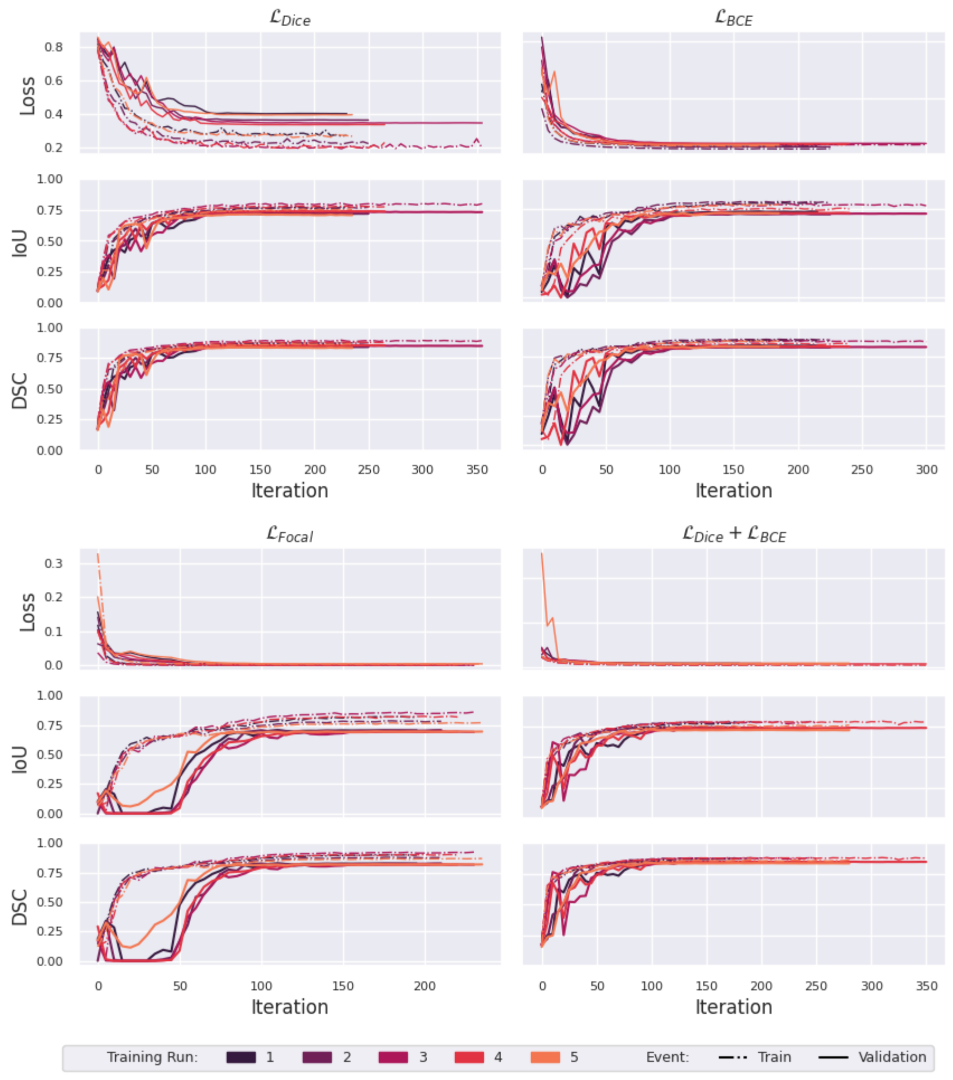

Figure A-11: Learning curves from the experiment on using different loss functions in Section 4.1.3 and Table 4.3.

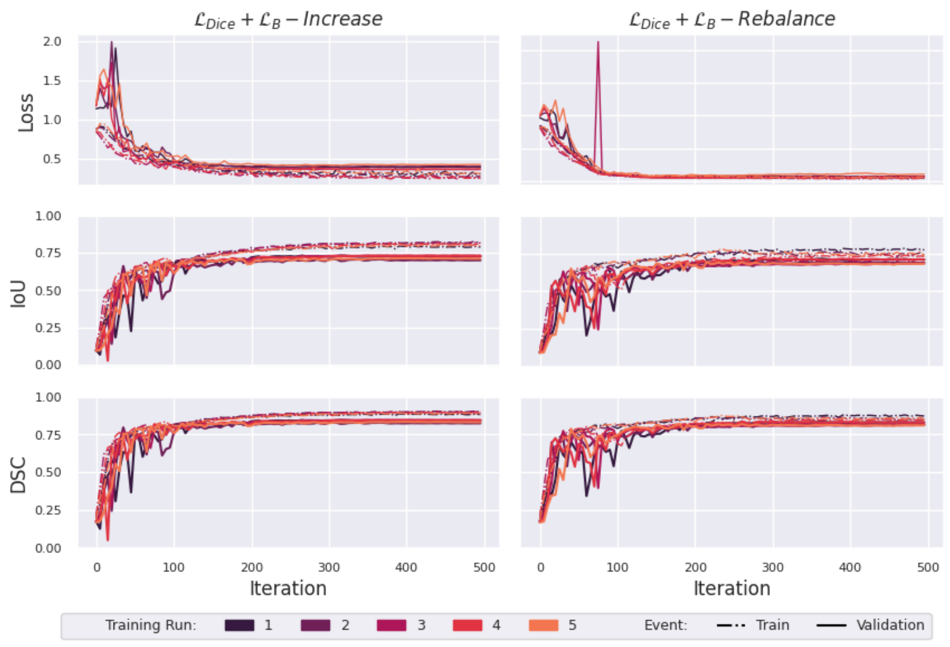

Figure A-12: Learning curves from the experiment on using different loss functions in Section 4.1.3 and Table 4.3.

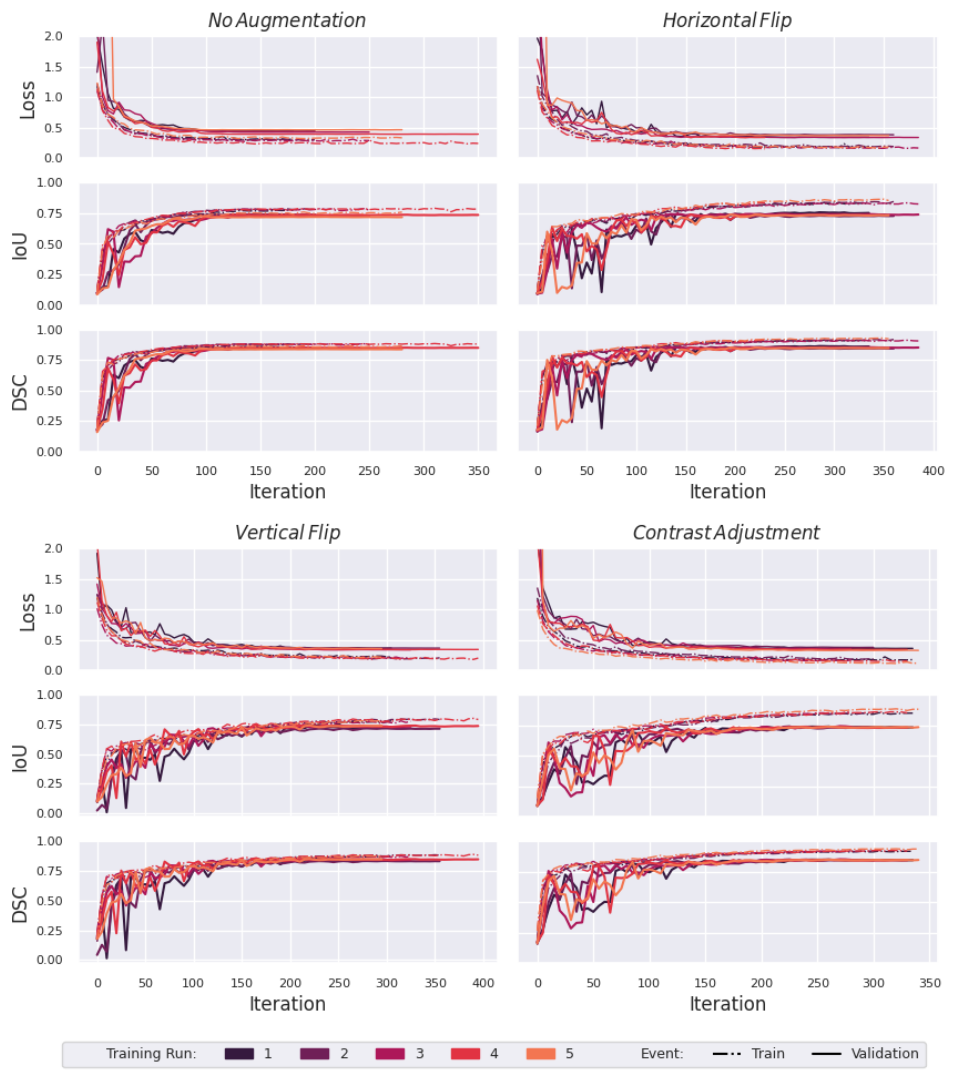

Figure A-13: Learning curves from the experiment on using different image-level augmentation methods in Section 4.2 and Table 4.4.

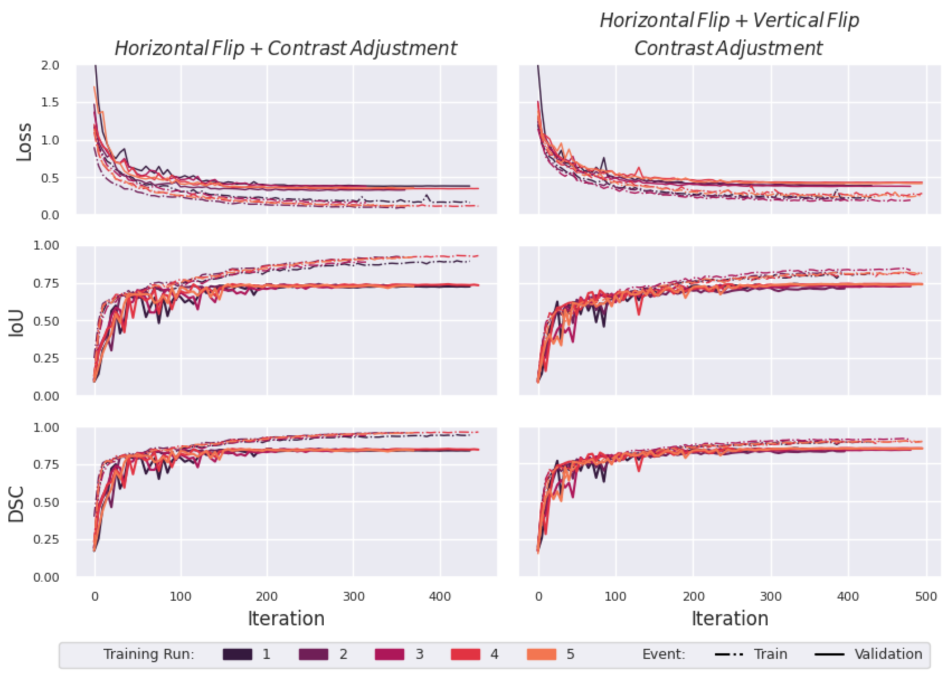

Figure A-14: Learning curves from the experiment on using different image-level augmentation methods in Section 4.2 and Table 4.4.

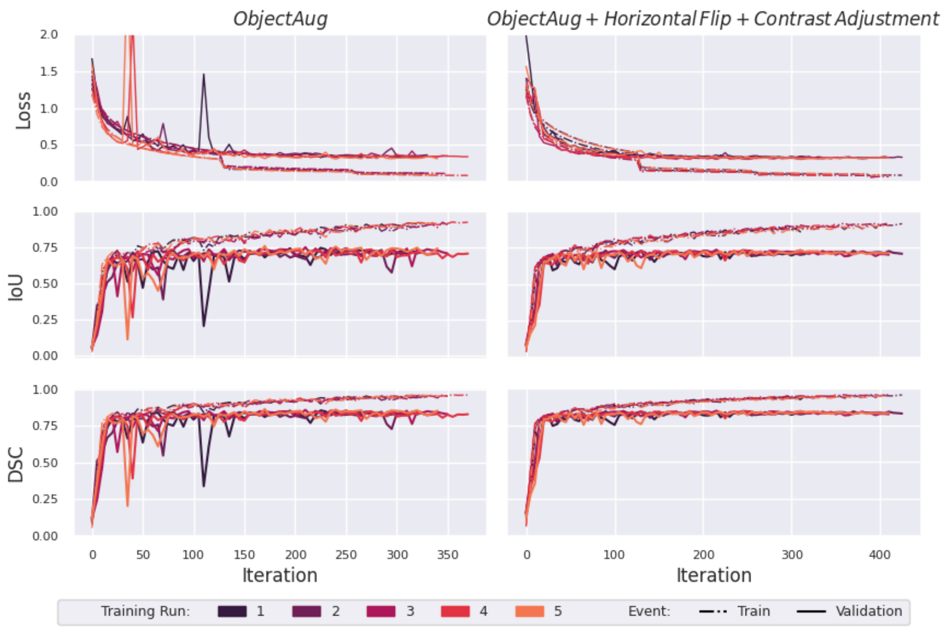

Figure A-15: Learning curves from the experiment on using the ObectAug method in Section 4.2 and Table 4.5.

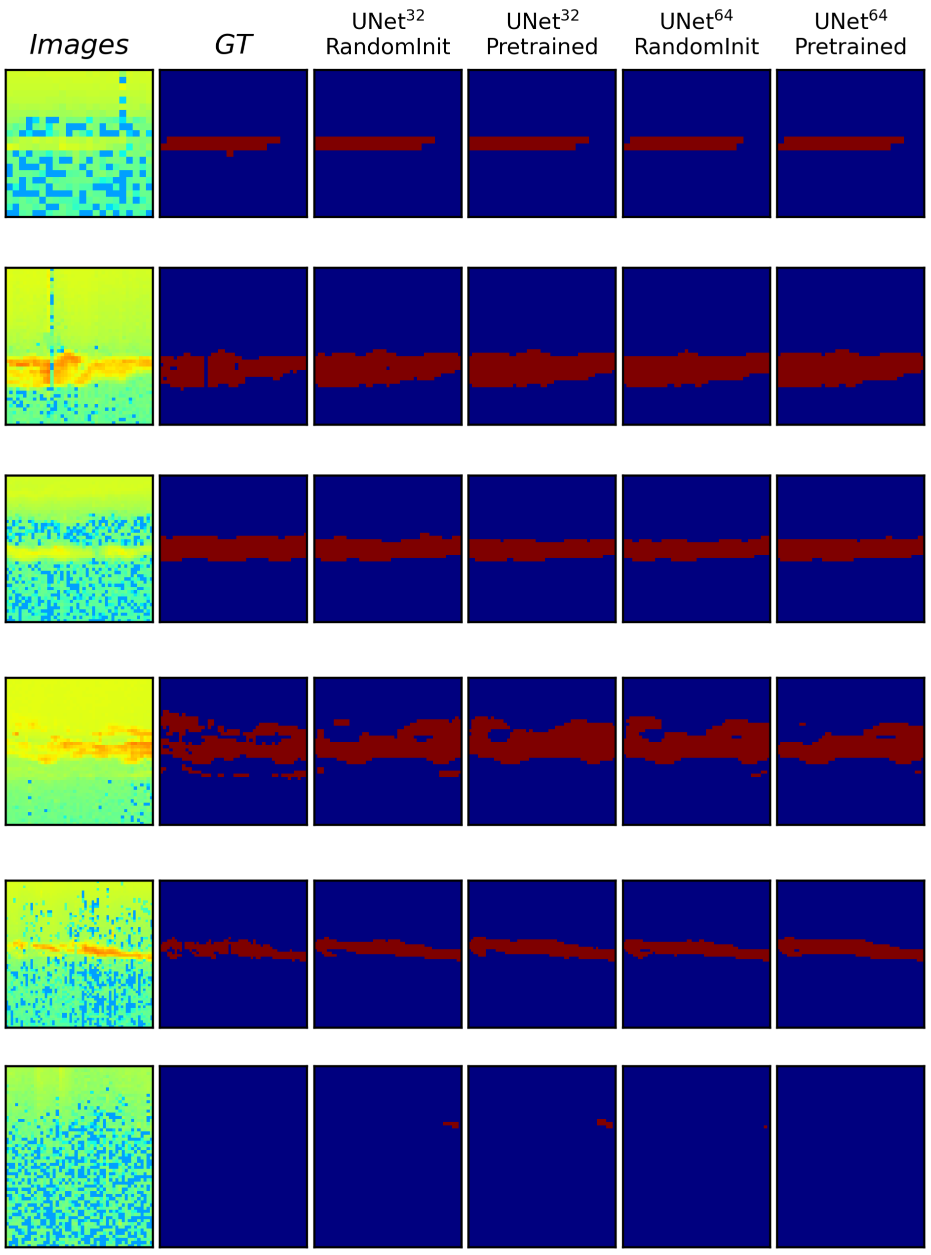

Figure A-16: Initial experiment Unet. Accurate Predictions.

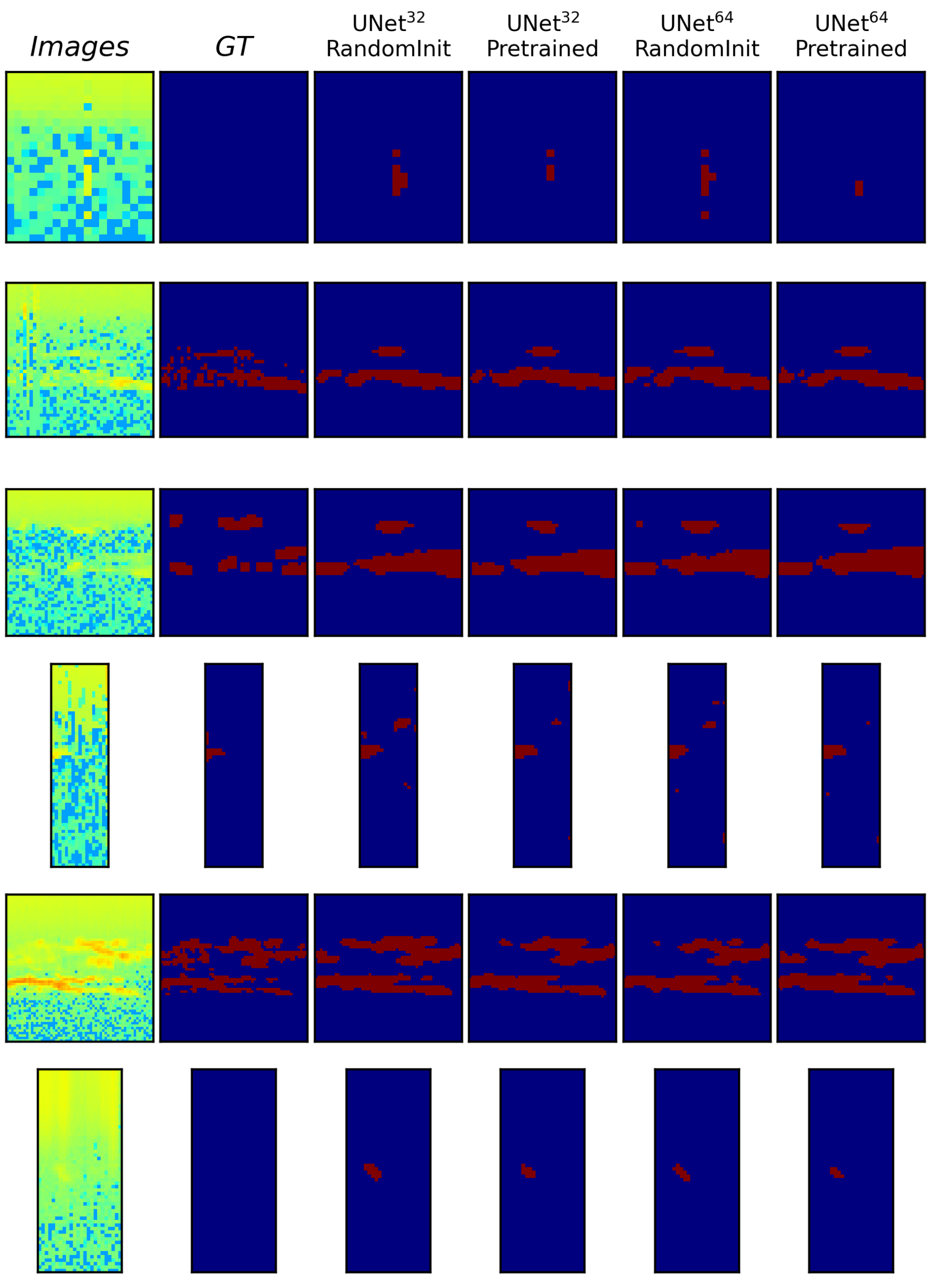

Figure A-17: Initial experiment Unet. Less accurate predictions.

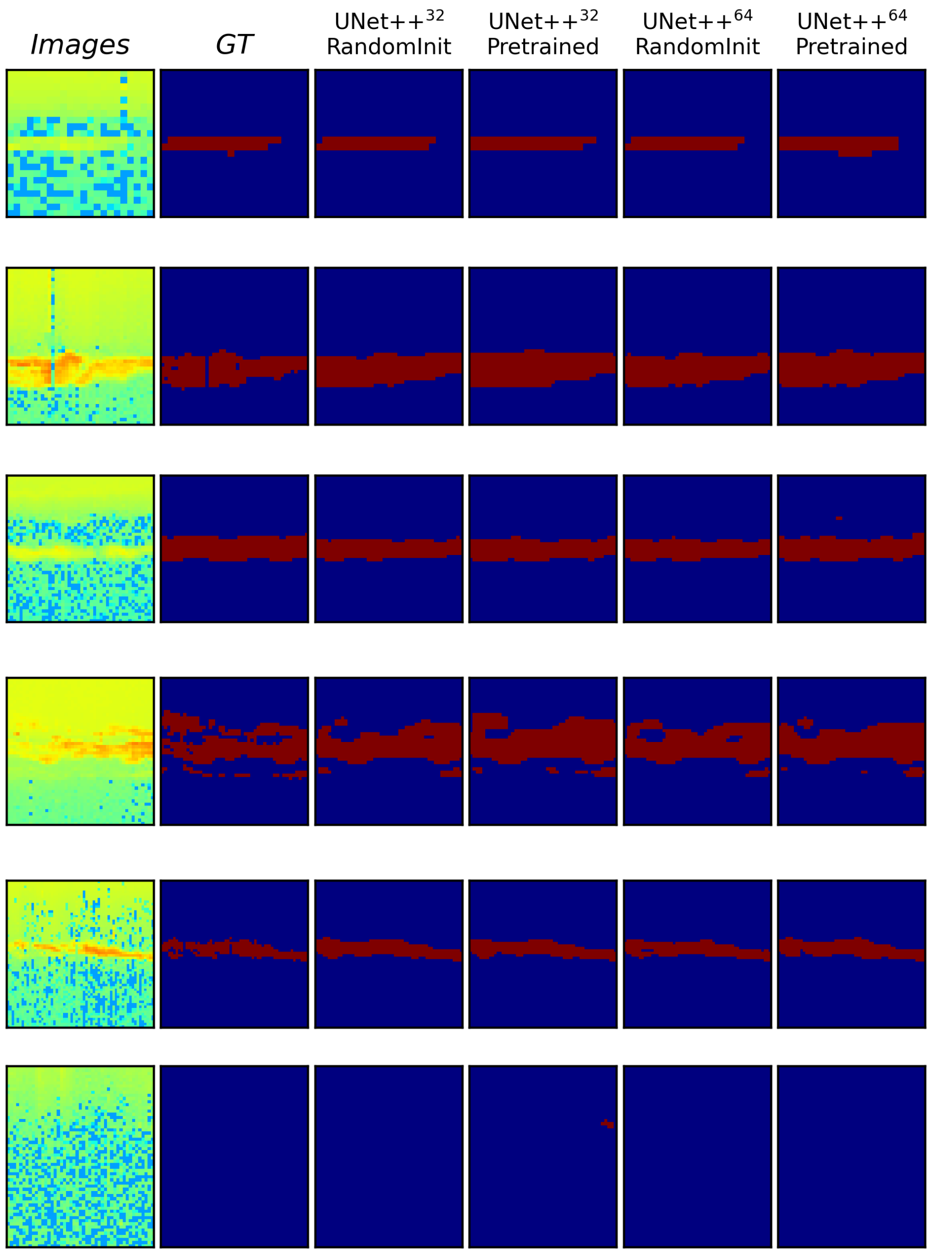

Figure A-18: Initial experiment Unet++. Accurate Predictions.

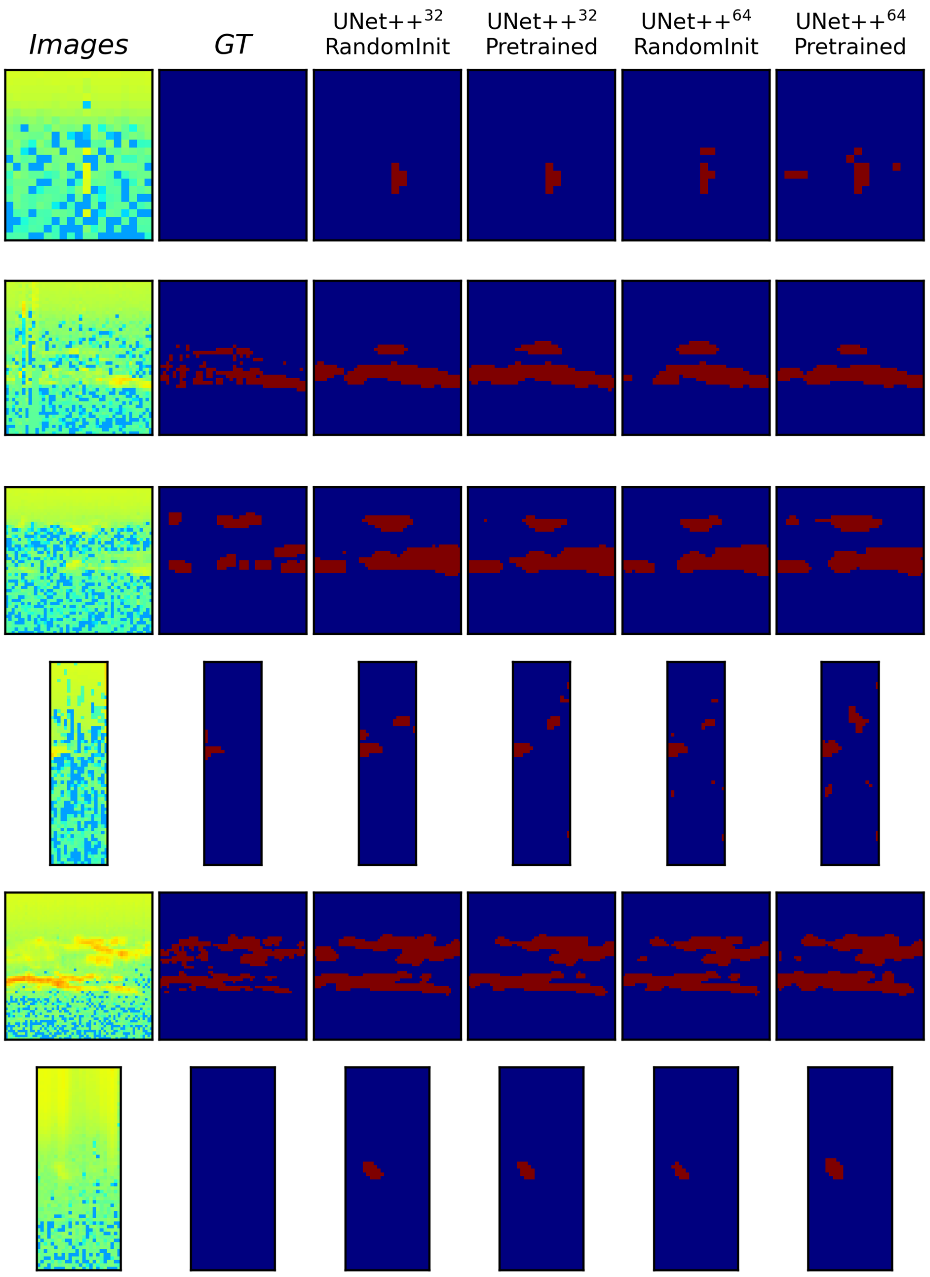

Figure A-19: Initial experiment Unet++. Less accurate predictions.

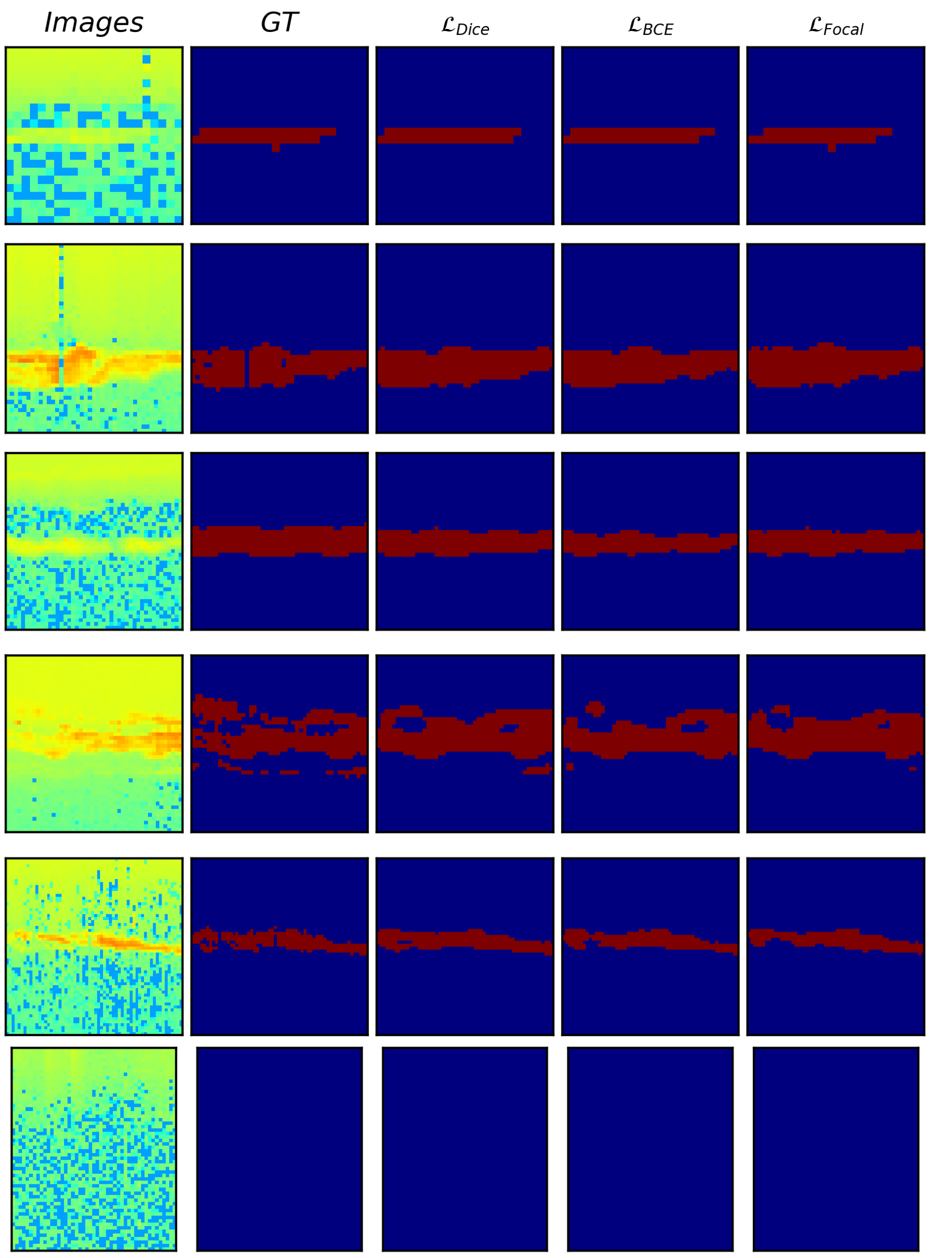

Figure A-20: Easy Samples. Qualitative comparison between the different loss functions $\mathcal{L}_{Dice}$, $\mathcal{L}_{BCE}$ and $\mathcal{L}_{Focal}$ using a random initiated UNet++[64] architecture. The images and their ground truth label is shown in the first and second column, respectively.

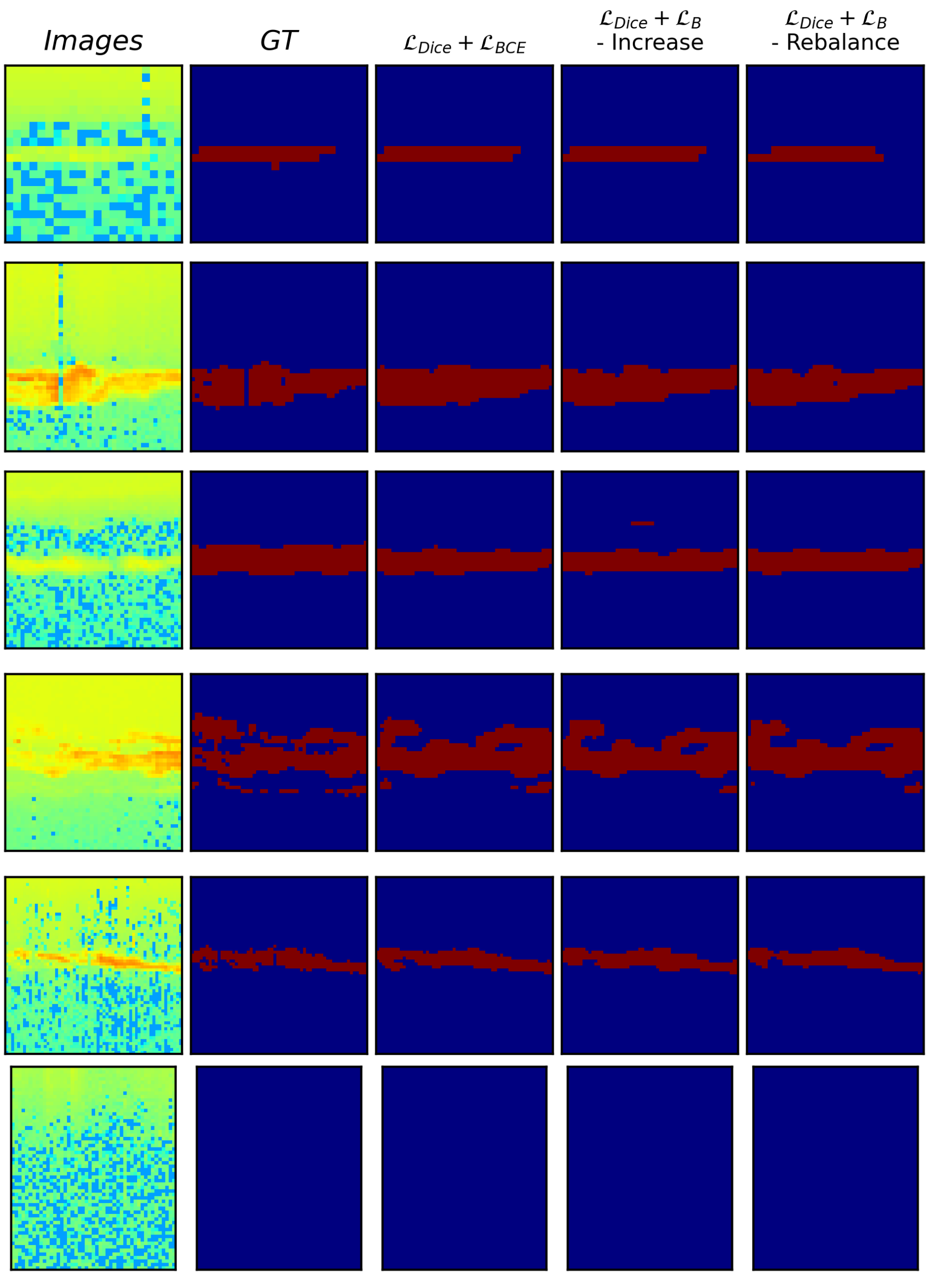

Figure A-21: Easy Samples. Qualitative comparison between the different loss functions $\mathcal{L}_{Dice} + \mathcal{L}_{BCE}$ and $\mathcal{L}_{Dice} + \mathcal{L}_{B}$ - *Increase* -and *Rebalance* using a random initiated UNet++[64] architecture. The images and their ground truth label is shown in the first and second column, respectively.

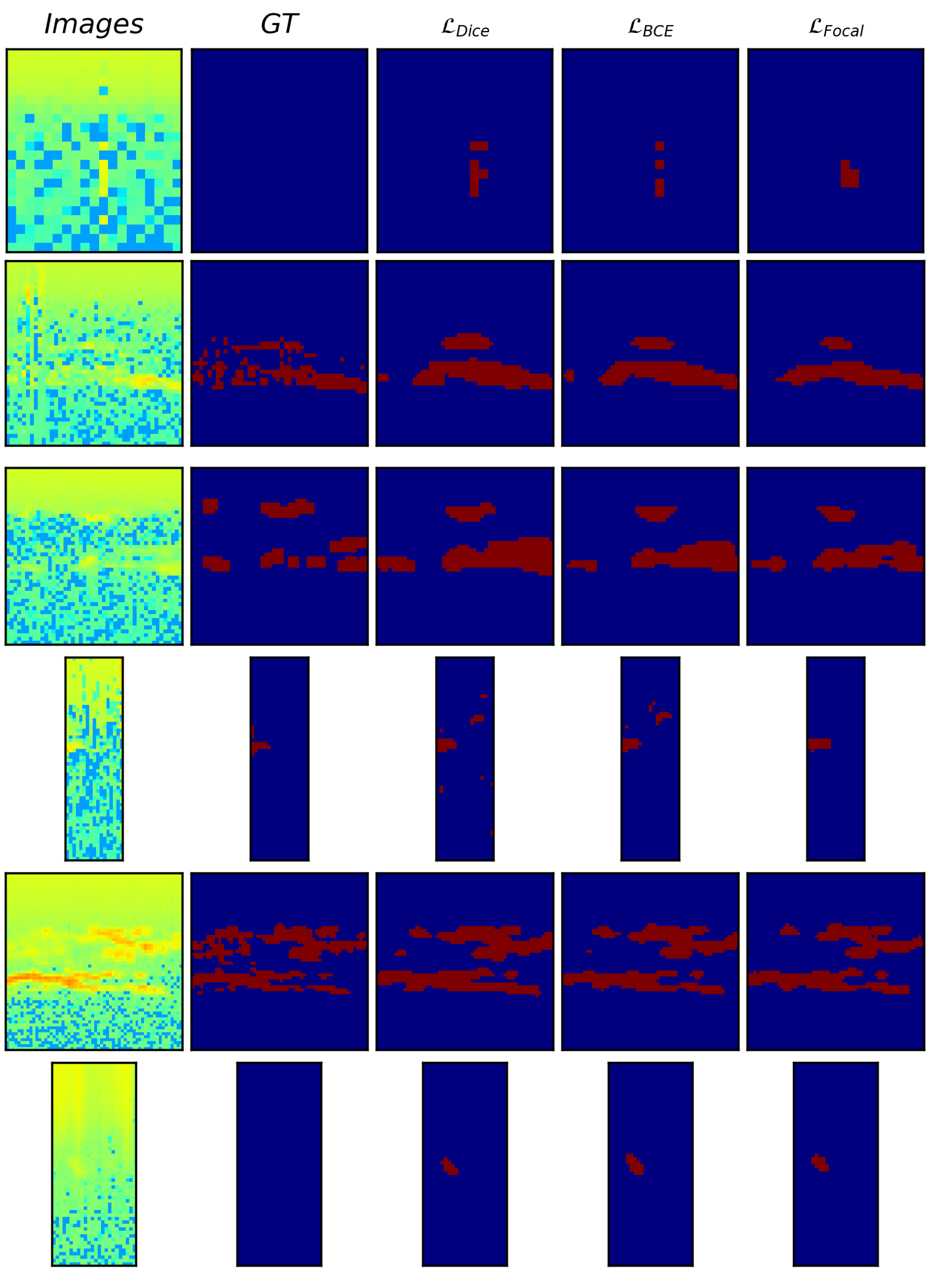

Figure A-22: Difficult Samples. Qualitative comparison between the different loss functions $\mathcal{L}_{Dice}$, $\mathcal{L}_{BCE}$ and $\mathcal{L}_{Focal}$ using a random initiated UNet++[64] architecture. The images and their ground truth label is shown in the first and second column, respectively.

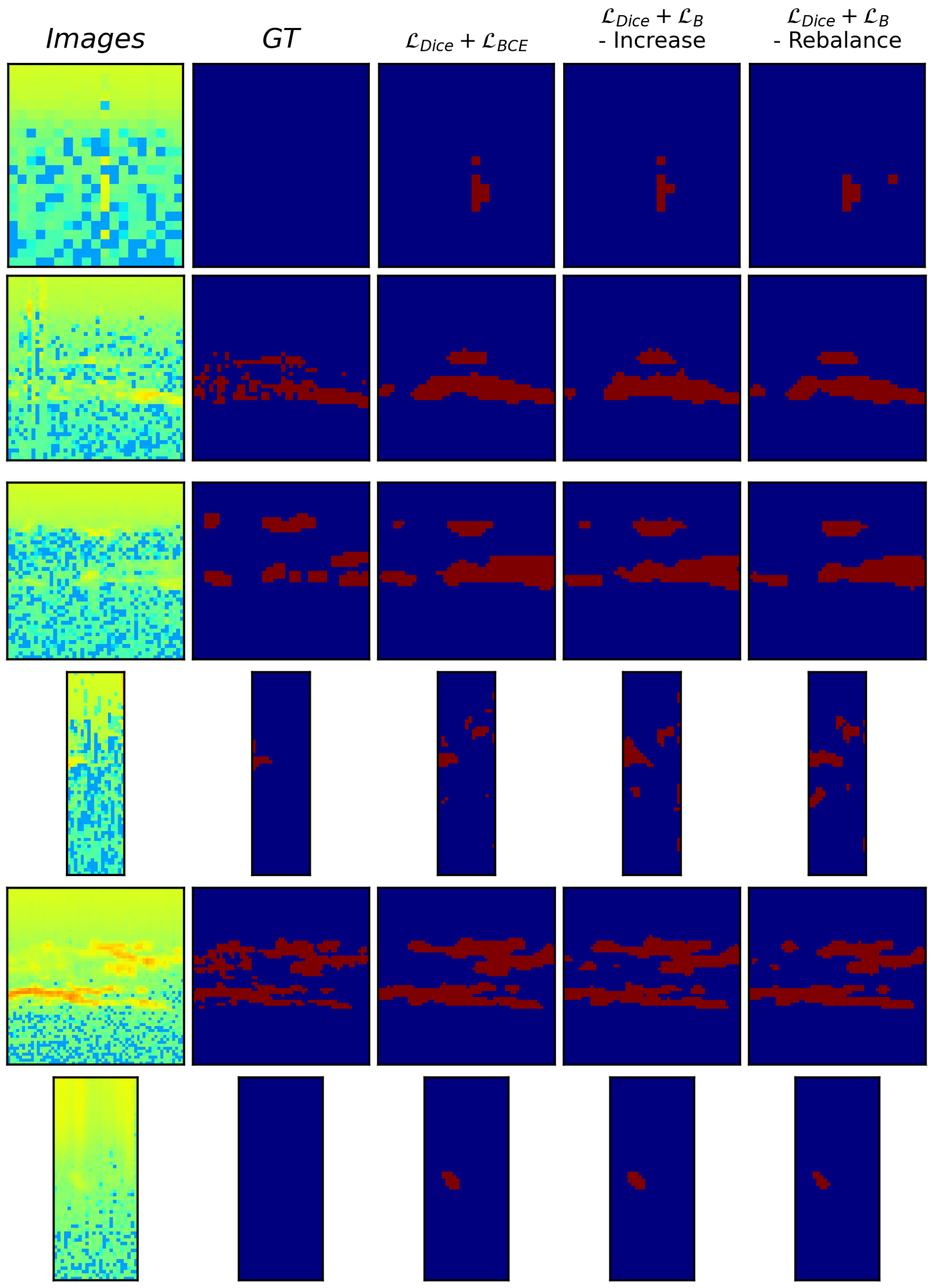

Figure A-23: Difficult Samples. Qualitative comparison between the different loss functions $\mathcal{L}_{Dice}+\mathcal{L}_{BCE}$ and $\mathcal{L}_{Dice}+\mathcal{L}_{B}$ - *Increase* -and *Rebalance* using a random initiated UNet++[64] architecture. The images and their ground truth label is shown in the first and second column, respectively.

Sample Number	Dates	Start Time in UTC	End Time in UTC
(1..18)	(d/m/yyyy)	(hh:mm:ss)	(hh:mm:ss)
1	30 June 2008	07:59:38	12:07:30
2	02 July 2008	10:24:30	11:59:02
3	10 June 2009	09:03:42	11:56:09
4	14 July 2009	08:19:33	11:33:15
5	16 July 2009	08:47:30	10:06:26
6	17 July 2009	07:49:44	11:59:30
7	30 July 2009	12:15:29	15:59:08
8	07 July 2010	00:00:30	21:59:27
9	08 July 2010	09:00:42	12:59:03
10	09 July 2010	09:00:24	12:59:09
11	01 June 2011	08:34:31	10:02:07
12	08 June 2011	07:23:50	13:01:07
13	09 June 2011	08:01:45	12:59:26
14	01 July 2014	09:00:36	13:00:24
15	10 August 2015	09:14:40	16:12:28
16	12 August 2015	20:04:40	23:59:28
17	13 August 2015	00:00:28	01:59:26
18	20 August 2015	00:00:28	01:59:26

Table A-1: The table lists the observation dates (day, month, year) of the samples used in this study. The observation time is given in UTC with the format *Hours:Minutes: Seconds*.