The Power of Privilege: Enhancing Land Cover Classification with Privileged Information

Agnes Eriksson, Malte Åhman

Master's thesis
2024:E23

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

The Power of Privilege: Enhancing Land Cover Classification with Privileged Information

Agnes Eriksson och Malte Åhman

2024-06-05

Abstract

Given the pressing environmental crisis, marked by climate change, biodiversity loss, and deforestation, precise land cover monitoring has become crucial. In this thesis, we explore the use of privileged information to enhance land cover classification using deep neural networks. Our study specifically focuses on how to best utilise all available data, by leveraging a machine learning framework known as *learning using privileged information*, or LUPI for short. The main idea of the LUPI paradigm is to exploit extra information during training, which might not be available during inference. Based on the the recently released FLAIR 2 dataset, we evaluate three main LUPI strategies, of which a teacher student setup yields the most promising results. Our proposed approach achieves an mIoU of 0.606, compared to the baseline score 0.588. These scores are also on par with the relevant state-of-the-art models. The teacher student architecture is mainly based on a U-Net with EfficientNet B4 as an encoder, but the results also generalise to other networks. In conclusion, we successfully exploit the available privileged information during training to improve inference results.

Acknowledgements

First and foremost, we would like to take the opportunity to thank our supervisors, Aleksis Pirinen, Martin Willbo and Olof Mogren at RISE, and Mikael Nilsson from LTH. You have all helped us through this thesis with helpful insights, and we have greatly appreciated your genuine interest in making the best our of this project. Our discussions about what to focus on, and your help when we faced issues, have been invaluable.

We would also like to express our gratitude to both RISE and LTH, in general, for making this thesis possible. We have learnt so much during the last five years, and especially during this final semester.

Contents

1 Introduction

In this section, we discuss relevant background and the objectives of this master thesis. It includes a motivation to why the topic is important and a short introduction to the main areas of the report. Hopefully, after this section the reader will have a better understanding of the field we are working in, why it matters and what to expect from this thesis.

1.1 Background

The environmental crisis is characterized by a range of pressing issues, from climate change and biodiversity loss to pollution and deforestation, each contributing to the destabilization of natural systems and the depletion of vital resources. Urgent and coordinated global actions are essential to mitigating these effects and preserving the planet for future generations. To handle this, we need to understand which actions that are most efficient, and to continuously evaluate them. One crucial part of this is to study the Earth's land cover. If we are able to follow the deforestation, urbanization and melting glaciers live and in detail, we possess a greater chance of avoiding the crisis. For this, we need data.

The gathering of information about our planet's physical, chemical, and biological systems via remote sensing technologies, is referred to as Earth observation (EO). If we collect many images of the land cover of our planet, these images can over time tell us how water levels are changing, forests are developing and cities are growing. One part of conducting such environmental monitoring is to understand which pixels of a certain image that show a forest, a building or some other relevant category. Pixel wise classification of such images is known as land cover classification, and is a type of semantic segmentation.

Deep neural networks are well suited to perform land cover classification, as the amount of data often is immense. When executing the classification, we also want to use as much information as possible. That includes information that might not always be available for all data points, and during inference. Such information is referred to as privileged. The main purpose of this thesis is to understand how learning with privileged information (LUPI) might improve the performance of land cover classification using deep neural networks. The fundamental idea is that one sometimes has more information per data point available during training, than when actually using the model. This can be the case when we have training data collected from Sweden, which monitors some set of values, but the model is intended for usage in France, where a different set of values are available. Ideally, the LUPI paradigm helps the model understand the underlying function during training, so that the model does not need the privileged information during inference.

1.2 Objectives

The objectives of this thesis are

- To understand if the privileged information available, near-infrared and elevation data, actually contains information that is useful for the model.
- To study the potential of LUPI in the context of land cover classification. More specifically, we will study noise architectures, teacher student architectures and predicting/generating privileged information architectures.
- To understand which of NIR and elevation is the most helpful privileged information.
- To study the usage of available metadata and satellite images.

To the best of our knowledge, LUPI within deep neural networks for EO is a fairly unexplored area. The basic ideas of privileged learning are discussed in this paper [27] from 2009, but it focuses on support vector machines. These principles are further developed in an article [13] from 2016, which studies the usage of privileged information for deep neural nets. Our main objective is to develop and apply these principles for land cover classification. For this, we have three main ideas which will be our starting point. These are

- Noise architectures
- Teacher student architectures
- Predicting/generating privileged information architectures.

The specifics of these architectures will follow in the methods section. However, before actually exploring privileged learning, we need to see if the available privileged information is useful. That is to say, we need to train one model with privileged information, and another one without. From this, we hope to conclude that we receive better results from the model with access to privileged information. If not, it is unlikely that the privileged information is useful at all, and hence any further LUPI efforts are deemed to fail. This initial check is a sub target to the above mentioned main objective of studying LUPI in the context of land cover classification. During these experiments, we will also study which type of privileged information that is most useful for land cover classification, NIR or elevation. Furthermore, we will examine the usage of available metadata and context giving satellite images.

1.3 Restrictions

Throughout this thesis, we have been forced to prioritize, and to refrain from exploring several ideas due to the lack of time and computational resources. In general, machine learning is time-consuming, and relies on heavy computation. As mentioned, we will look at LUPI within the use of land cover classification. In a best case scenario, we would try many different LUPI architectures, and several different networks, and look at how these could be combined. Ideally, we would even use multiple datasets. Furthermore, for most of our experiments, we have to set multiple hyperparameters. The best would be to do this through thorough experiments, but we, at times, have to make qualified guesses. It should also be mentioned that LUPI has applications within other domains, but we focus exclusively on land cover classification.

1.4 Disposition

Following this introductory section, we delve into the theoretical foundations. For readers already familiar with the basics of neural networks, or those less interested in technical specifics, it may suffice to briefly review the dataset discussion (see Section 2.6). Subsequently, the methods section outlines the conducted experiments. This is followed by a comprehensive results section, which provides an extensive presentation of our findings. Readers less concerned with granular details may prefer to consult only the summary of results (see Section 4.12), where the key findings are summarized and the most interesting results are shown.

After presenting the results, we analyze their implications and reflect on limitations in the discussion section. The thesis concludes with a generalisation of the most relevant results utilizing three additional network architectures. Finally, we outline potential ideas for future research and then summarize the thesis.

The document concludes with the references and an appendix, providing supplementary materials and sources cited throughout the research.

1.5 Division of Work

All experiments were planned and conducted in joint consultation between the two authors. Furthermore, all sections in the report have been written in close collaboration, and to explicitly separate the sections between the authors is not doable.

2 Theory

The theory section includes an in-depth exploration of neural network basics, the specific architectures we utilise, the concepts of privileged learning, our evaluation metrics, loss functions, and a detailed overview of the dataset employed. As mentioned, for readers already familiar with the basics of neural networks, or those less interested in technical specifics, it may suffice to focus on the section dedicated to the dataset (see Section 2.6).

2.1 Neural Networks

Using machine learning, we can automate many tasks which would be tedious, if not impossible, to do by hand. Machine learning evolves around the idea of presenting data (a dataset) for a model and to let it learn by studying it. Very briefly, the learning happens by presenting data, letting the model return some type of output based on the input data and then evaluate the output. The evaluation is then utilised to update the parameters of the model. This description is wide, and indeed there are many ways of doing machine learning. It ranges from simple models such as linear regression, commonly with fewer than a dozen trainable parameters, to deep neural nets which can have many hundreds of billions of trainable parameters. Which type of machine learning model to employ depends heavily on the problem. A very crass rule is that the more complex data (commonly in the sense of data being high dimensional), the more complex model (commonly by having more parameters) we want. In this study, the data consists of images, which are high dimensional as each pixel can be seen as a dimension. Hence, we will leverage deep neural networks, and in this section we will develop the fundamental theory regarding them. We will not be able to cover everything, and will focus on giving the reader an intuition for main ideas of what a neural net is, and how it can be trained. The interested reader can find more information in the Deep Learning book written by Ian J. Goodfellow, Yoshua Bengio and Aaron Courville [7].

2.1.1 Supervised Learning

There are in general three main ways to train a model. They are called supervised learning, unsupervised learning and reinforcement learning. Supervised learning requires both input data and labels for the target for the model. That is to say, if we want to train a supervised model to tell if a picture shows a cat or a dog, we will need all cat and dog images to come with a label which says either "cat" or "dog". A unsupervised model would instead only get the cat and dog images, and without any feedback on it attempts try and search for underlying features. This approach is convenient in the sense that it does not need labeled data - which is nice since it is time consuming to tag 10000 images as either "dog" or "cat". However, it is sometimes more difficult to yield good results with unsupervised learning. Reinforcement learning is structured a bit differently. It has an agent, which learns to make decisions by taking actions in an environment to maximize some notion of cumulative reward. So, a reinforcement model gets some feedback on its performance, and just like unsupervised learning does not require labeled data. However, constructing a suitable reward function is often challenging, and in the end requires a lot of work as well. Hence, when labels are available, a common approach is to use supervised learning, and so is done in this study as well. So during training we have input data (images x) and labels (the ground truth y) to be used for comparison between the models prediction and the truth [7]. The comparison is the utilised to create a loss function, which we will let you know more about shortly.

2.1.2 Training, validation and test data

The common procedure when training a model is to split up the available dataset into a training set, a validation set and a test set. The training set is used to train the model, and is allowed to alter it. To understand how well the model would perform on general data (drawn from the same distribution as the training set), it is common to take one part of the dataset as a test set. The model should not be altered based on the test set results, it should only be seen as an evaluation. Hence, it is appropriate to let the relation between the test set and training set well replicate the relationship between the training data and the expected real world data. As an example, if the model is trained on images from 10 provinces of France, and is then supposed to be used to make predictions for other provinces of France, it is suitable to let the test set consist of images from provinces not available in the training set.

The validation set is in between the training set and test set. The model should not be directly altered on it, and its general format should be similar to the test set. However, during training, the model is run on the validation set to see how well it performs and especially how well it generalizes to unseen data, such as the test set. Hence, the validation set can be seen as an indicator of how well the model will perform on the test set, and it is common practice to use the model which performs best on the validation set. The reader may also wonder why the model would not perform as well on the validation or test set as on the training set. The main reason for this is that the model can "over fit" on the training data. This means that the model over adjusts its parameters to particularly suit the training data. As deep neural nets have so many parameters, this happens quite easily, and in the extreme the model might even be able to "memorize" the training set, if the data complexity is low and the model complexity high. Therefore, it is important to have a training set, validation set and test set to best prepare for real world usage.

2.1.3 The Perceptron

The perceptron, introduced by Rosenblatt in 1958, is the building block of neural nets [20]. A perceptron, more commonly known as a neuron, takes an input vector x and takes the scalar product of x and a weight vector w, of the same size as x. It also has one bias parameter b, which is added to the scalar product as $x \cdot w + b$. This is referred to as the weighted sum, and the weights w and the bias b are trainable parameters. This means that w and b will during training be updated by an algorithm trying to reach some goal. The algorithm is called backward propagation, the goal is to minimize the so-called loss, and we will further develop these topics in the following sections. These neurons can be stacked both next to each other and "on top" of each other. A layer is typically some number of neurons next to each other, and the "depth" of a network is broadly speaking the number of layers of neurons stacked on top of each other. Any layer that is not the input or the output layer is referred to as a hidden layer. Briefly, you can think of each layer in a neural network as a matrix that takes in an input vector and produces an output vector. The whole network is formed by chaining such matrix multiplications together.

2.1.4 The Activation Function

There is a second important step in the neuron, after the calculations of the weighted sum. The weighted sum is in general not passed on as input to the next layer just as it is. Instead, we first apply an activation function. Some activation functions commonly used nowadays are rectified linear unit (ReLU), softmax, and the sigmoid, which you will get to know more about shortly. There are two main reasons to not pass on the weighted sum as it is, but instead, first pass it through an activation function. The first one is that the activation function introduces non-linearity, which might help the network better approximate any non-linear functions. The second one is that it can limit the output range of any given layer, and hence the stability in the network increases.

We will now quickly go over the definitions of ReLU, sigmoid and softmax. ReLU was first introduced in 1969 [5] and is defined as

$$\mathrm{ReLU}(x) = \begin{cases} x, x > 0 \\ 0, \text{otherwise.} \end{cases}$$

The motivation behind ReLU specifically lies in that this behaviour is believed to resemble the behaviour of the neurons in an animal brain. As seen, it is quite simple but still introduces non-linearity. The observant reader may notice that ReLU is in fact not differentiable everywhere, more specifically at zero. However, this is handled by setting the derivative there to either 0 or 1.

The sigmoid $\sigma(x)$ is another common activation function. It is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The sigmoid was introduced in the context of neural networks in this paper [22] and the main difference between the sigmoid and ReLU is that sigmoid keeps all activation values within an interval of $[-1, 1]$, which sometimes is preferable. Any deeper theory regarding which activation function to use when is however out of the scope of this report.

Softmax has been used for a long time, but its use was formalized in for instance this book [8]. Softmax converts a vector of k real numbers into a probability distribution of k outcomes. This means it rescales the vector so that all values are zero or greater (as negative probabilities are undefined) and assures it sums to one (as the sample space of all possibilities must sum to one). Hence, softmax is commonly used in the last layer of a network to give some prediction in terms of probabilities of the different classes. Softmax is defined as

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}.$$

2.1.5 The Loss Function

Now that we know about the neuron - the fundamental building block of a neural net - it is time to introduce the "goal" for the network. First, the network designer will have to decide what it wants from its network. As an example, say we want to be able to correctly identify cats from images of 10 different animals. Then we want to create some metric to say how good the network is at that, and as an example metric m we can take how many percent of shown cats are identified as cats. Now, it seems natural to train the network to try and maximize this metric. It is, however, more common to minimize some negative version of the metric. This function is called the loss function, and is basically created as described above. However, the loss function needs to be differentiable, so if the goal metric is not, it needs to be generalized to a differentiable function. This is again for the learning algorithm, backpropagation, to work. The most commonly used loss function is called Cross Entropy (CE) and will be formally introduced later. There are also some loss functions more specifically adapted to semantic segmentation, and some of these will also be mentioned.

2.1.6 Forward Propagation

Forward propagation in a neural net is when we give an input and feed it forward through the network, using all the weights and biases. Most supervised tasks include trying to predict something. For example, it could be to predict which animal is shown in an image. In that case, by the end of the forward propagation, the network will give some probability for the image belonging to each class. Once the network has given a probability, we use this to calculate the loss function by comparing it to the label (which will be 1 for the correct class and 0 for all others). After that, we conclude the forward propagation, by taking the class with the highest probability to be our prediction $\hat{y}$.

2.1.7 Backpropagation

We know how the network is built, and what goal it has. Now we just need some clever algorithm to work towards that goal. The, by far, most commonly used algorithm is called gradient descent. One might wonder why we can not try a brute force approach, i.e. try every single combination of learnable parameters, and the reason for that is simply computational restrictions. Assuming a network of a couple of 100 million parameters trying every possible permutation of parameter values, even to only a few decimals and within a tight interval, takes a tremendous amount of time. The advantage of brute force is obviously that we will find the global minima of the loss function, instead of some local minima which commonly is the case with gradient descent. There is currently no other known way of being sure of finding a global minima, due to limitations of current optimization theory.

Gradient descent is constituted by five main steps. The first step is initialization, which means to initialize the parameters to some value. These values are commonly taken randomly between 0 and 1. Step two is to run the input through the network and calculate the value of the chosen loss function (this is the forward propagation step). The third step is the most complicated, but also most important one. This is the "backwards" step, in which we start by computing the gradient for the loss function. As the gradient points in the direction of the steepest ascent, and we want to minimize the loss function, we want to move in the opposite direction. The gradient of the loss is then propagated backward through the output layer to compute the gradients with respect to the weights and biases of the output layer. The gradient is further propagated backward through the hidden layers. For each layer, starting from the last hidden layer and moving towards the input layer, the algorithm computes the gradient of the loss with respect to each weight and bias. The computation involves applying the chain rule of calculus repeatedly to propagate the gradients through the layers. The activation functions' derivatives are crucial at this step because they modulate the gradients as they are propagated back through the network. Also, it is hopefully clear at this point why the activation function and loss function should be differentiable. After all computing, the fourth step is to simply update all parameters according to the gradients computed in the previous step. The gradient computation tells in which direction we should change the parameters, but there is also a hyperparameter called the learning rate, which decides how large each step will be. The fifth, and last, step is to repeat from step two until the learning has converged, i.e. until it seems the network cannot learn more. An overview of gradient descent optimization algorithms can be found in An overview of gradient descent optimization algorithms by Sebastian Ruder from 2016 [21].

2.1.8 Batches

When training a net using backpropagation one can take one input sample at a time. However, it is more common to do it in batches, which means taking a batch of inputs at the same time calculating a loss, and doing the backward propagation for these together. The batch size is decided partly by computational restrictions - the larger the batch size, the more memory is needed by the computational resource. The other thing to have in mind is that the network tends to converge slower when having too large batch sizes since more data has to be passed through the network for each weight update. Typically, it also impacts the generalisation of new data.

2.1.9 Hyperparameters and Fine-Tuning

We have now been through the fundamentals of neural networks. Before moving on to topics that are more specialised for this thesis we also want to introduce some concepts which are important to the functionality and performance of a neural network. These are

1. Data Augmentation: Data is often scarce, and by altering available data one can get more data, for "free". Such transformations commonly include rotating the image, and changing the saturation and contrast of an image. By introducing more data, data augmentation also helps prevent overfittning.
2. Optimizers: To move in the loss landscape as cleverly as possible there are various optimizers to help, i.e. ways of improve the learning further than only taking a step with the size of the learning rate in the direction of the gradient. Two common ones are Adam, introduced in 2014 [4] and Stochastic Gradient Descent (SGD), introduced in 1951 [18].
3. Hyperparameters: There are many parameters, apart from the above mentioned, which have to be set "manually" as they are parameters for how the model should behave during training. Such parameters are known as hyperparameters.
4. Regularization: This is a fundamental technique in machine learning and statistics used to prevent overfitting. Regularization methods introduce additional information or constraints into a model to discourage learning overly complex models.
5. Epoch: The number of times the model ran through the entire dataset. So if the model was trained for 10 epochs, it means the model was exposed to all input data 10 times.

2.1.10 Convolutional Layers

The simplest type of layers constructed by neurons are fully connected layers. These consist of neurons next to each other, which take an input vector of some length and then returns a vector of the same size as the number of neurons in that layer. However, when working with image data, as we do, the networks almost always contains at least some convolutional layers. This is since the image data commonly is quite high dimensional, and also often has many channels. As an example, an RGB image with width and height of 512 pixels has three channels (R: red, G: green, B: blue), and the shape of the matrix representing this image would be $3, 512, 512$. To efficiently combine the values of these three channels it is suitable to use convolutional layers.

The convolutional layers were introduced already in 1989 [14], and are built on the convolutional operation. In this context, the convolutional operation refers to a sliding filter (often called kernel) over the image, and computing the dot product of the filter with the image regions it covers. The idea is that this operation is able to extract certain features from the image, such as edges, corners and other patterns. The kernel is commonly quadratic with a few pixels width and height, but span over all channels of the image. As this kernel slides over the images, it moves a certain number of pixels at the time, and this number is called the stride. The result of the convolution operation is a feature map that represents the input's filtered version, highlighting specific features detected by the filter. Different filters can detect different types of features. After the convolution operation, an activation function is typically applied to the feature map, for which ReLU is the most commonly used. Often, after convolutional layers, a pooling layer is applied to reduce the spatial size (width and height) of the representation, reduce the number of parameters, and computation in the network, and hence also control overfitting. The most common pooling method is max pooling, which reduces the input's dimensions by taking the maximum value over a spatial window. Networks containing convolutional layers are often called convolutional neural nets (CNNs).

2.1.11 Encoders and Decoders

Some CNNs have a special structure called an encoder-decoder architecture, such as proposed by [1], and this includes several of the networks that we use for this thesis. The main idea of this architecture is to create multiple representations of an image and then use these to perform pixel-wise classification. Such a structure is shown in figure 1. The first part, the encoder, takes the input and generates something called feature maps. These maps are of a lower dimension than the input image, and the idea is that only the most important features of the image remain in this new representation. In the case of image segmentation, these are constructed by passing the image through several convolutional, pooling and downsampling layers. Depending on the complexity of the encoder, the features can represent simpler things such as edges and corners or more complex objects. The further an image is passed through layers, the more abstract the feature representation.

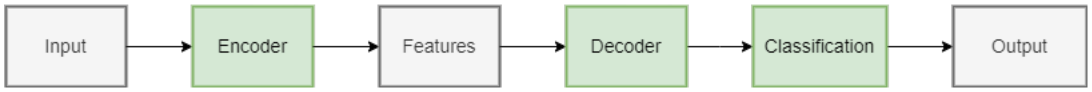


Figure 1: A simple flowchart that illustrates how a simple encoder-decoder architecture works. This image has been constructed using draw.io.

Once the image has been passed through the encoder, the network has generated several feature representation that need to be treated. This is where the decoder comes in. The overall goal of the decoder is to recreate a representation of the image that has the same dimensions as the original input image. Similar to how the encoder works, this is done in several steps using convolutional layers, but upsampling instead of downsampling. By using all the different feature maps from the encoder, a segmentation mask is produced that can be used for the final classification. This last layer usually consist of a softmax layer to produce probabilities as output. In short, the encoder-decoder structure extracts useful features of an image to then construct a, hopefully, accurate pixel-wise prediction of an image.

2.2 Neural Networks Architecture

Despite focusing exclusively on state-of-the-art architectures for semantic segmentation, there remains a vast selection of network models to choose from. These architectures often build upon traditional CNNs, with modifications tailored to the semantic segmentation task. Due to time constraints, we concentrated primarily on a single network, U-Net, while excluding others early on. The U-Net architecture will be comprehensively detailed in the following section.

Additionally, we will briefly touch upon ResNet50, FCN-8s, and UNetFormer as they are utilised in smaller parts of this thesis. This section is not essential for readers uninterested in technical details, but to understand our subsequent adjustments to U-Net for fitting our task, some understanding of its architecture is beneficial. Notably, the U-Net architecture is of more interest than the others.

2.2.1 U-Net

U-net was first introduced in 2018 as a CNN built for biomedical image segmentation [19]. However, biomedical images are not fundamentally different from Earth observations, and hence the architecture applies also to our dataset. The fundamental idea of U-Net is the encoder-decoder structure described earlier. The encoder follows a typical architecture of a convolutional network. The encoder can be any network that returns features. In the original U-Net implementation, the encoder was constituted by repeated applications of two 3×3 convolutions, each followed by a ReLU and a 2×2 max pooling operation with stride 2, for downsampling (”making the image smaller”). At each downsampling step, i.e. after each pair of convolutions, the number of feature channels is doubled. The feature channels are initially the number of input channels, e.g. three for an RGB image with input channels red, green, and blue. Generally speaking, U-Net shifts the images from being spatially large to becoming deeper. In our study, we used EfficientNet-B4 as the encoder, which was introduced in 2019 by Mingxing Tan and Quoc V. Le [25]. This architecture is slightly more complex than the original U-Net encoder, but the main idea of having convolutional layers and downsampling to get deeper and deeper features is still the same. The difference is primarily that EfficientNet-B4 includes mobile inverted bottleneck MBConv [24] and squeeze-and-excitation optimization [11]. The interested reader can learn more in the respective papers, as the details are not relevant to this thesis.

Once the encoder has produced five sets of features, which successively are ”deeper” (more channels) and smaller (fewer pixels in width and height), with the fifth feature being the deepest, it is time to decode these features. The decoder has to convert the deep image into an image of full width and height again. Every step in the decoder consists of an upsampling of the feature map followed by a 2×2 convolution (“upconvolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the encoder, and two 3×3 convolutions. Each of these is followed by a ReLU activation. The final layer of U-Net is a segmentation head, which is a 1×1 convolution, used to map each feature vector to the desired number of classes.

2.2.2 ResNet50

ResNet50, short for Residual Network 50 layers, is a variant of the ResNet model that is (yes, you guessed it) 50 layers deep. It was introduced as part of a family of Residual Networks in 2017 [9], and is a CNN. The ResNets were developed to address the vanishing gradient problem in very deep neural networks (that is to say, the problems of gradients going towards zero). The main innovation of ResNet is the introduction of ”residual blocks”, which allow for the training of much deeper networks by using skip connections or shortcuts to jump over some layers. These shortcuts effectively enable the network to learn identity functions, ensuring that the added layers can at least achieve the performance of the shallower model, thereby not harming the training process.

2.2.3 FCN-8s

Fully convolutional network, FCN, is a specific type of convolutional network, where exclusively convolutional layers are used. This type of network was proposed in 2014 [15] for the use of semantic segmentation. The network itself is not too different from the U-net architecture, but has a decoder that produces only one feature map, meaning there is no "U shape". The 8 in FCN-8s refers to the fact that the resulting segmentation has a resolution that is 8 times lower than that of the input image.

2.2.4 UnetFormer

The UnetFormer is another network with an encoder-decoder structure, that was proposed in 2022 [28]. What separates it from the other networks mentioned above, is that the decoder is a transformer network, meaning no convolutions are involved. A transformer network has the advantage of being good at extracting global features from the image, meaning it can use information that might be far away in an image. The hope is that this will improve the segmentation results relative to CNNs.

2.3 Privileged Learning

We will continue by formally defining the LUPI paradigm, and also providing a general theory for the teacher student architecture, one of the main LUPI methods examined. Regarding the other two main architectures, they have been developed empirically, and are not based on literature studies. Therefore, they are instead introduced in the methods section.

2.3.1 LUPI Paradigm

Learning using privileged information, LUPI, was introduced by in 2009 by [27], and proposed a new way of using privileged data to improve model performance. It was further developed for deep neural network by Jonschkowski et al. in [13]. Based on these two papers, we present a theoretical foundation for the LUPI paradigm. To do this, we must first present the standard way of training a segmentation model. Given some input-label pair

$$(x_i, y_i), \quad x_i \in X, \quad y_i \in Y, \tag{1}$$

where X is some input space and Y is some label space, the goal is to find a function f such that $y_i = f(x_i, \alpha)$, where α are the model parameters. The same type of data is available during both training, and inference. The LUPI paradigm is slightly different. Imagine that there is privileged data x_i^* available during the training of a model. We now have triplets of data

$$(x_i, x_i^*, y_i), \quad x_i \in X, \quad x_i^* \in X^*, \quad y_i \in Y, \tag{2}$$

where X^* is some input space with privileged information. The aim is still the same, we want to find a function f such that $y_i = f(x_i, \alpha^*)$, where α^* are the model parameters from training with privileged information. The difference, however, is that we can find α^* using both x_i and x_i^*. In a less formal way, we say that we have access to the privileged information during training, but not during inference.

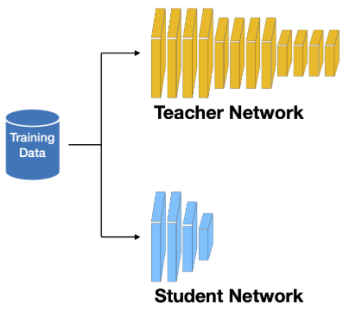
Jonschkowski et al. [13] provides an illustrative example of how the LUPI paradigm can be useful. The example itself cannot be transferred directly to the domain of semantic segmentation as the example is trivial but provides some intuition behind the use of privileged information. “Suppose we want to estimate a function from the input/output samples: 3→14, 5→30, and 2→9. From looking at these data, it is not immediately obvious what the true underlying function is. However, if we provide additional information and the prior that they correspond to intermediate values that f computes, in this case, 3→9→14, 5→25→30, and 2→4→9, we see that the function first squares its input and then adds five to the intermediate result, $f(x) = x^2 + 5$. Side information together with a prior about how they relate to f reveal the underlying function.”That is to say, the privileged information ideally helps the model understand the true underlying function, such that the model will not be affected by the lack of privileged information during inference. In real-life applications, an example of a situation where privileged information could be available could be if we train a model using data collected in Sweden, where we track more values than in e.g. France, where the model is intended for usage. Then we have all of this extra information available during training, and we want to utilise all available data.

One should also notice that privileged information occurs in most, if not all, fields. Hence, even though this thesis specialises on land cover classification the fundamental ideas might very well hold to completely different types of data.

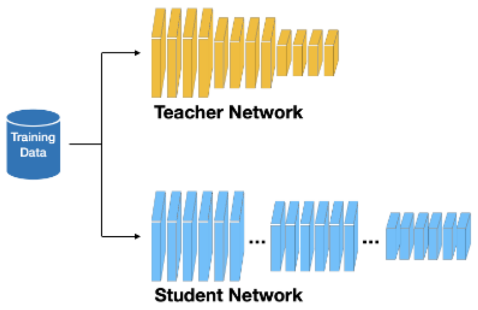
2.3.2 The Teacher Student Method

One way of training according to the LUPI paradigm is with a teacher student setup. As the name suggests, this architecture consists of a teacher network and a student network. The skills of the respective networks and their mutual relationship can vary depending on the specific situation.

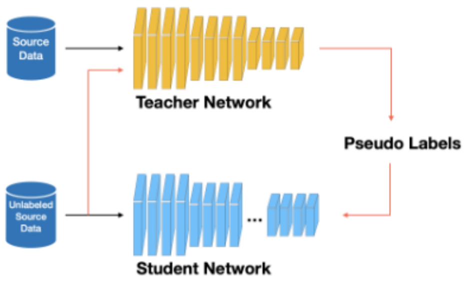
According to Chengming Hu et al. [10], there are four main usages for teacher student setups. The first one is knowledge distillation, which aims to reduce the size of the student network. The second one is knowledge expansion, which aims to expand the size of the student network. The third is called knowledge adaptation and aims to transfer knowledge from the source domain to the target domain by training a student network. The fourth, and last, is denoted multi-task learning, and aims to compress knowledge from multiple domains into one domain by training a single student network. In figure 2, we can see illustrations of these four types of use cases. These can be overlapping and it may be so that not any of the four suits a specific setup exactly. In our situation, we want to pass on information from the privileged channels from the teacher to the student. This is closest to knowledge adaptation, but in some sense also knowledge distillation as the student network will be slightly smaller than the teacher network as it takes fewer inputs.



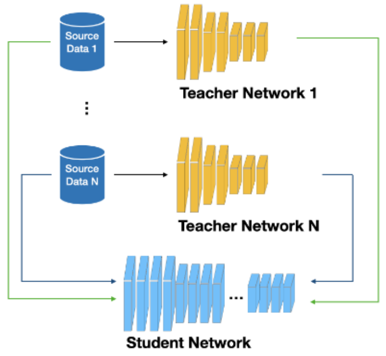
(a) Knowledge distillation: aims to reduce the size of the student network.



(b) Knowledge expansion: aims to expand the size of the student network.



(c) Knowledge adaptation: aims to transfer knowledge from a source domain to a target domain.



(d) Multi-task learning: aims to compress knowledge from multiple domains into one domain by training a student network.

Figure 2: The four main categories for teacher student architectures. All illustrations from [10].

The most important question stands; how does the teacher help the student learn? The common procedure is to link the two networks together through the loss function of the student network. Inspired by these papers [26], [16], we can describe this knowledge transfer in four steps.

1. First, train the teacher network, using input-output triplets (x_i, x_i^*, y_i), where x_i* in our case is the privileged information, which the student will not have access to.

2. After finishing training the teacher, take each input x_i, x_i*, and compute the predictions $\hat{y}_i$ for the teacher. Then, take the softmax of the prediction as $\sigma(\hat{y}_i)$ to get the predicted probabilities per class. Recall, softmax normalises the input so that it sums to one. At this stage, it is also possible to introduce a hyperparameter $T_T > 0$, called the temperature. It is used to smooth the teacher's prediction as $\sigma(\hat{y}_i/T_T)$.

3. It is now time to train the student, and this procedure contains two steps. Assume we pass an input-output pair (x_i, y_i) through the student network. From x_i, the student makes a prediction $\tilde{y}_i$, and between $(\tilde{y}_i, y_i)$, we calculate a loss $\mathcal{L}_S$. Normally, we would use that loss to backpropagate and update the weights. In the teacher student setup, however, there is an intermediate step. To compute this

intermediate step, we need to take the softmax also of $\tilde{y}_i$. Such as for the teacher, we can introduce a smoothing parameter $T_S > 0$ for the student, so that we get $\sigma(\tilde{y}_i/T_S)$.

4. At this stage, we know the smoothed, softmaxed predictions from both the teacher ($\sigma(\tilde{y}_i/T_S)$) and the student ($\sigma(\hat{y}_i/T_T)$). We use these to compute another loss $\mathcal{L}_T$, typically a measure of the distributional differences between the two predictions. This loss component $\mathcal{L}_T$ is then added to the student's own (”regular”) loss $\mathcal{L}_S$. The weighting between the two loss components is set by $\alpha \in [0, 1]$, and a rescaling parameter R is introduced to make the components match in magnitude. The final loss is given as $\mathcal{L} = \alpha R \mathcal{L}_T + (1 - \alpha)\mathcal{L}_S$, and is utilised to backpropagate through the network.

To summarize, the teacher makes a prediction using the privileged information, and the student makes one without it. This all means that the student will be nudged in some direction by the teacher, by always comparing its own prediction with the teacher's.

2.4 Evaluation Metrics

Three of the most common metrics for semantic segmentation are mean intersection over union (mIoU), precision and recall. These are all in some sense variations of the accuracy, which is commonly used for classification tasks. Which of these metrics that are most relevant depend on the specific task. As an example, if the objective is to detect the occurrence of some decease infecting trees it is more important to not have any false negatives than false positives, as it might be fine to check one time too much. On the other hand, if the goal is to understand how urbanisation has developed, one might care equally much about false negatives and false positives. All three metrics are calculated for a the entire validation set and test set respectively, by comparing our predictions with the true labels. Since our study concerns privileged information in general, rather than solving one specific task, we will evaluate our results based on all these metrics. However, we will especially focus on mIoU as it is the prevailing metric within the field.

2.4.1 Mean Intersection Over Union (mIoU)

The name ”mean intersection over union” is quite self explanatory, but to be explicit we will provide a formal definition. In equation 3 below, defining intersection over union (IoU), y_c denotes the label for a class c and $\hat{y}$ the prediction the class c. This expression is equivalent to all true positives (TP), divided by the sum of all true positives (TP), all false positives (FP) and all false negatives (FN). These calculations are done over all pixels, per class, i.e. the union returns 1 for a pixel i and a class c if either the prediction or the true label was class c. Likewise, the intersection returns 1 for an pixel i and class c if the prediction and the true label had the value associated with the class c. The $|X|$ operation refers to the cardinality and takes the size of the set X. The mIoU is derived by taking the mean of IoU over all (relevant) classes. This might be considered as the most general metric as it in some sense penalizes all types of errors equally.

$$\text{IoU} = \frac{|y_c \cap \hat{y_c}|}{|y_c \cup \hat{y_c}|} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{3}$$

2.4.2 Precision

The second metric we use to evaluate the performance of our model is precision. As for mIoU, the precision is calculated over all pixels per class, and the final value is the mean of the precision over all classes. Hence, the precision is actually the mean precision. In equation 4 below we can see the formal definition of the precision metric as the cardinality of the intersection between the true labels y_c for a class c and the predicted labels $\hat{y_c}$ for the class c divided by the number of predictions made of that class. This is equivalent to the the number of true positives divided by the sum of true positives and false positives. That is to say, this metric can be interpreted as when we actually guess class c, how often we are correct. However, it says nothing about the case where we never guess that specific class. Again, the definition below is for one specific class c and the final value, reported in the results, is the mean of the precision over all classes.

$$\text{precision} = \frac{|y_c \cap \hat{y_c}|}{|\hat{y_c}|} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

2.4.3 Recall

The third and final metric is recall. Similarly to the other two metrics, recall is calculated over all pixels per class, and again the recall is reported as the mean of the recall over all classes. The formal definition of recall can be seen in equation 5 below. There we see that recall (for a class) is calculated as the intersection between the true labels y_c for a class c and the predicted labels $\hat{y_c}$ for the class c divided by the number of true occurrences of that class. This is the same as dividing the number of true positives by the sum of the true positives and false negatives. It can be interpreted as how often the model actually predicts a class once it occurs. Recall is in some sense an opposite of precision, as it will give a very poor score for a class c if the model never predicts it, however, it will for a specific class c give a perfect score if the model predicts only that class c.

$$\text{recall} = \frac{|y_c \cap \hat{y_c}|}{|y_c|} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

2.5 Loss Functions

In this section, we formally define the loss functions utilised in this project. Recall, the goal of the neural network is to minimize the loss function. The choice of loss function, therefore, has a great impact on the model. There is a plethora of possible functions, and which to choose depends on the application and structure of the model.

2.5.1 Dice Loss

A loss function commonly used for semantic segmentation is the Dice loss. Minimizing the Dice loss closely resembles maximizing the IoU. However, since the IoU is not differentiable, the Dice coefficient was introduced as a differentiable extension of the IoU. Another positive trait of the Dice loss is that it handles class imbalances particularly well. According to [12], the Dice coefficient is defined as

$$d_c(y_i, \hat{y}_i) = \frac{2y_i\hat{y}_i}{y_i + \hat{y}_i + \epsilon}, \tag{6}$$

where ϵ is a small constant used for numerical stability. By taking the sum over all pixels and averaging over all C classes, we define the Dice loss as

$$\mathcal{L}_D = 1 - \frac{1}{C}\sum_{c=1}^{C} d_c. \tag{7}$$

2.5.2 Cross Entropy

Cross entropy, CE, is the most commonly used loss and can be leveraged both for normal classification and pixel-wise classification. It is a measure of how similar two probability distributions are, and for any two distributions p and q, it can be defined as

$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\log(q(x)), \tag{8}$$

where $\mathcal{X}$ is the set of all possible outcomes. The above expression can be used to define a loss function if we consider our predictions and our labels to be probability distributions. For labels y_i, predictions $\hat{y_i}$ and the number of samples n, [12] defines the cross-entropy loss as

$$\mathcal{L}_{CE} = -\sum_{i=1}^{n}\log(\hat{y_i}) \tag{9}$$

2.5.3 Focal Loss

Focal loss is a semantic segmentation loss, related to cross-entropy. [12] defines it as

$$\mathcal{L}_F = -\sum_{i=1}^{n}\alpha(1-\hat{y_i})^{\gamma}\log(\hat{y_i}) \tag{10}$$

where α and γ are constants. The focal loss is the same as the cross-entropy loss, but with a factor to focus the training on difficult samples.

2.5.4 Kullback-Leibler Divergence

The Kullback-Leibler divergence, or KL divergence, is closely related to the CE loss as it can be defined from cross-entropy and regular entropy. Regular entropy for a probability distribution p is defined as

$$H(p) = -\sum_{x\in\mathcal{X}} p(x)log(p(x)), \tag{11}$$

Using this, [3] defines KL divergence for p and q as

$$D_{KL}(p||q) = H(p) - H(p,q) \tag{12}$$

2.5.5 Mean Square Error or L2 Loss

One loss function that cannot directly be used for semantic segmentation is the mean squared error loss (MSE). Although it cannot compare predictions to the true label of a pixel, it is still useful when comparing numerical variables. For some variable pair x_1 and x_2, [3] defines the mean squared error loss as

$$\mathcal{L}_{MSE} = \sum_{i=1}^{n}(x_{i,1} - x_{i,2})^2 \tag{13}$$

2.5.6 L1 loss

Finally, we have the L1 loss, which just as the L2 loss is suited for comparing numerical values rather than predictions and labels. L1 is, according to [3], defined as

$$\mathcal{L}_{L1} = \sum_{i=1}^{n} |x_{i,1} - x_{i,2}| \tag{14}$$

2.6 Dataset

Throughout this project, we have used the FLAIR 2 dataset (French Land cover from Aerial ImageRy), which includes Earth Observation data from both aerial and satellite imaging. The dataset was released in this paper [6] from 2023, and was established to address the critical need for monitoring land cover and soil conditions, which are essential for ecosystem sustainability. In response to these challenges, the French National Institute of Geographical and Forest Information is leveraging high-quality EO data and AI tools to monitor land cover in France. The dataset builds upon its predecessor, FLAIR 1, by incorporating Sentinel-2 satellite image time series and introducing a new test dataset. These enhancements aim to improve the accuracy and comprehensiveness of land cover monitoring, supporting the production of the French national land cover map reference.

2.6.1 Spatial and Temporal Domains

The data has been collected from 50 different domains in France, where each domain corresponds to a unique administrative area called *département*. These domains are spread out all over France, and are used to split the data into a training, validation and test set. The placement of these domains can be seen in figure 3, along with the time during which the respective data was acquired. As is showed in the figure, the time of acquisition differs between the three datasets.

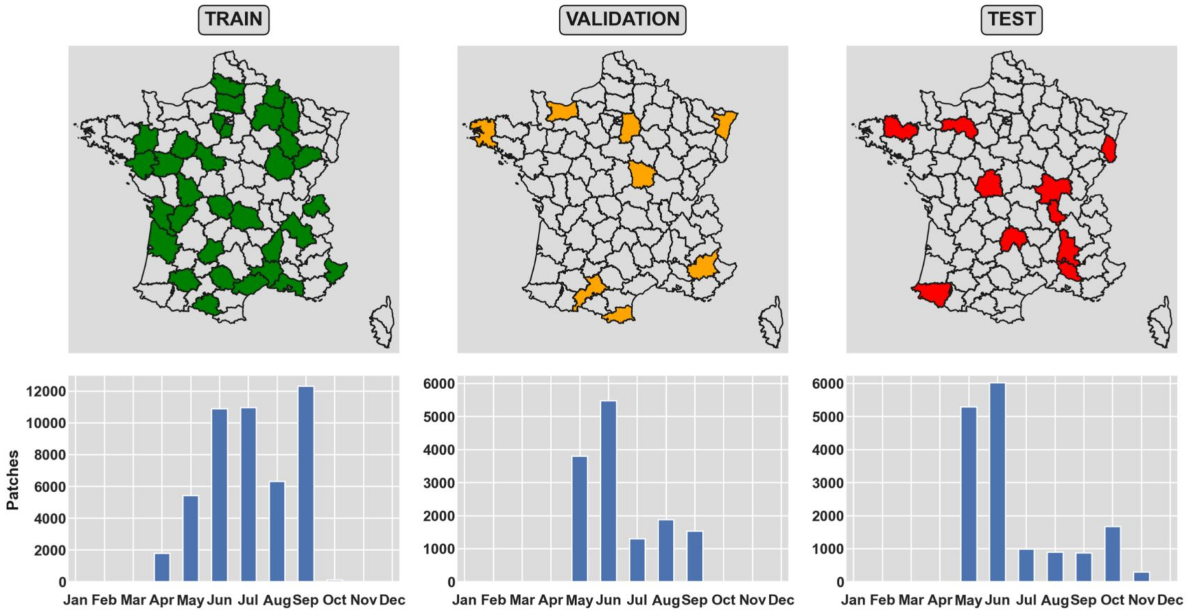


Figure 3: Spatial and temporal distribution of train, validation and test data. The images are taken from [6].

Within each domain, images have been taken from 916 areas and cover a total of 817 km^2. The areas vary in size, but are all split into patches of equal size. Furthermore, these areas have been chosen so that different landscapes are all well represented [6]. Figure 4 depicts how the domains relate to the areas and the patches.

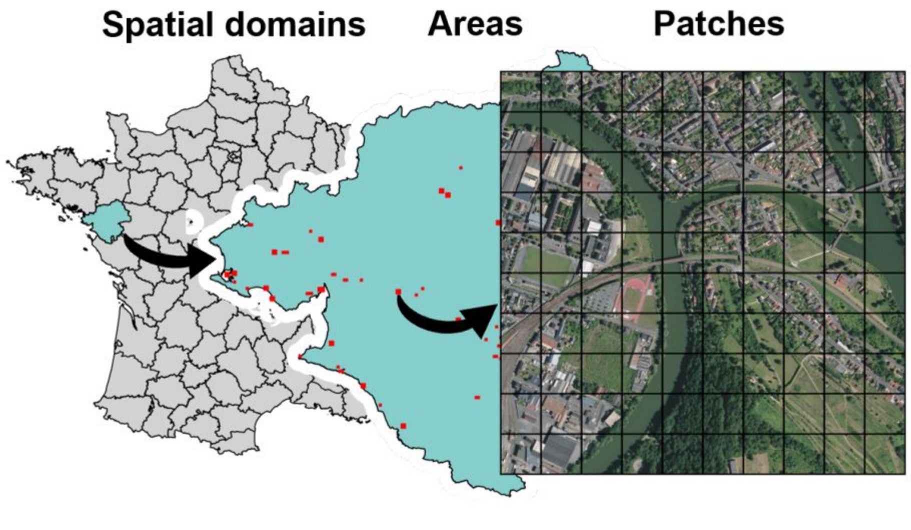


Figure 4: Spatial domains, areas, and patches. From [6].

Given that data was acquired over a vast area, there are differences in seasonality between domains, but also within each area. All training data has been acquired between April and September whereas the test data has been acquired between May and November. Due to the change in season, the same land cover class can have different appearance depending on both the location and the time of acquisition [6].

2.6.2 Aerial Imaging

Our main input data consists of aerial patches (images). In total, the dataset consists of 77,412 patches, each having 512×512 pixels. With a spatial resolution of 0.2 meters, each patch covers approximately 10486 m^2 on the ground. These patches have 5 channels that correspond to red (R), green (G), blue (B), near-infrared (NIR), and elevation. Depending on the domain, two different cameras were used in the acquisition, so slight variations in spatial resolution might occur between images, even though pre-processing has been performed to reduce this. The images have also been corrected to reduce any inconsistencies due to sunlight and contrast. Note also that the fifth class, elevation, is not the result of direct measurements but has been derived from the other 4 channels [6].

2.6.3 Metadata

The FLAIR 2 dataset provides metadata for each aerial patch. That means that each patch is associated with a date and time of acquisition, the geographical location of the center pixel, the type of camera used, and the average altitude of the image.

2.6.4 Satellite Imaging

In addition to the aerial imaging that is provided, FLAIR 2 contains satellite time-series data that comes from the Sentinel-2 satellite. This data is different from the aerial data when it comes to spectral and spatial resolution. In terms of spectral properties, the satellite data has 10 channels that are spread out from visible light (490 nm) to medium infrared light (2190 nm). All of these channels have a spatial resolution of either 10 or 20 m. Together with each satellite image, snow and cloud masks are provided that contain the probability of snow and cloudiness for each pixel. These can be used to filter out images that have a high probability of either snow or cloudiness. All channels that have a spatial resolution of 20 m are interpolated to match a spatial resolution of 10 m. Another difference from the aerial imaging, is that the satellite data contains several images for each patch of land, taken at different times. This way, temporal information can be extracted in a way that is not possible when only considering the aerial data. On average, each aerial patch has 50 corresponding satellite patches, but the number of satellite images that can be used varies due to snow and cloudiness [6].

2.6.5 Land Cover Classes

The labeling of all patches has been done manually by the help of experts in geography so that each pixel can be mapped to a land cover class. There are a total of 19 different classes in the dataset. Here follows a list of all land cover classes with descriptions, taken from [6].

- **Anthropized surfaces without vegetation (1, 2, 3, 13 and 18)**
  - Class 1 – building includes not only buildings, but also other type of constructions such as towers, agricultural silos, water towers and dams. Greenhouses (class 18) are an exception.

- Class 2 – pervious surface defined as man-made bare soils covered with mineral materials (e.g. gravel, loose stones) and considered to be pervious. It includes pervious transport networks (e.g. gravel pathways, railways), quarries, landfills, building sites and coastal ripraps.
- Class 3 – impervious surface is defined as man-made bare soils that are impervious due to their building materials (e.g. concrete, asphalt, cobblestones). It includes roadways, parking lots, and certain types of sports fields.
- Class 13 – swimming pool is defined as man-made artificial (open-air) swimming pools. It is not included in class 5 (water).

- **Natural areas without vegetation (4, 5 and 14)**
  - Class 4 – bare soil defined as natural permanently bare soils. These natural soils remain without vegetation throughout the year and generally are covered with sand, pebbles, rocks or stones. Examples of natural bare soils are frequently found in coastal, mountainous and forested areas.
  - Class 5 – water is defined as areas covered by water, such as sea, rivers, lakes and ponds. An exception are swimming pools (class 13).
  - Class 14 – snow refers to surfaces covered by snow. It is an extremely rare class as the images are taken in the summertime and only very few regions in France are covered with snow year-round.
- **Woody natural vegetation surfaces (6, 7, 8, 15, 16 and 17)**
  - Class 6 – coniferous, is defined as trees identifiable as coniferous (pines, firs, cedars, cypress trees, ...) and taller than 5 m.
  - Class 7 – deciduous is defined as trees identifiable as deciduous (oaks, beeches, birches, chestnuts, poplars, ...) and taller than 5 m.
  - Class 8 – brushwood refers to natural woody surfaces with a vegetation less than 5 m high. It includes short and young trees, brushwood, shrublands, mountain moors and abandoned agricultural lands.
  - Class 15 – clear-cut, is defined as forest areas, in which the trees have been cut down and harvested.
  - Class 16 – ligneous is an extremely rare class used to describe forest areas with a homogeneous representation of either coniferous or deciduous trees.
  - Class 17 – mixed is an extremely rare class used to describe forest areas with heterogeneous trees for which the types of trees (coniferous/deciduous) cannot be determined with sufficient certainty.
- **Agricultural surfaces (9, 11 and 12)**
  - Class 9 – vineyard despite being an agricultural use of the land, are assigned a class apart, a reason being their rather distinctive land cover characteristics.
  - Class 11 – agricultural land encompasses various different agricultural classes. For example, besides major crops, it also includes permanent and temporary grasslands with agricultural use. Vineyards (class 9) are not included in this class.
  - Class 12 – plowed land is defined as agricultural land with no visible vegetation (e.g. recently plowed and freshly harvested land).
- **Herbaceous surfaces (10)**

- Class 10 – herbaceous vegetation defines herbaceous surfaces that are not intensively exploited for agriculture purposes. This class includes ornamental lawns (e.g. gardens, public parks), recreational fields (e.g. used for sport), natural herbaceous areas in forested or mountainous areas, non-cultivated grass in agricultural areas or along transportation networks.

Not all classes are equally distributed in the dataset, and as mentioned in the description, some classes are even deemed to be extremely rare. Figure 5 shows the distribution of pixels over the classes for the dataset. The last class '*other*' is here an aggregation of class 13 through 19, which all are very small.

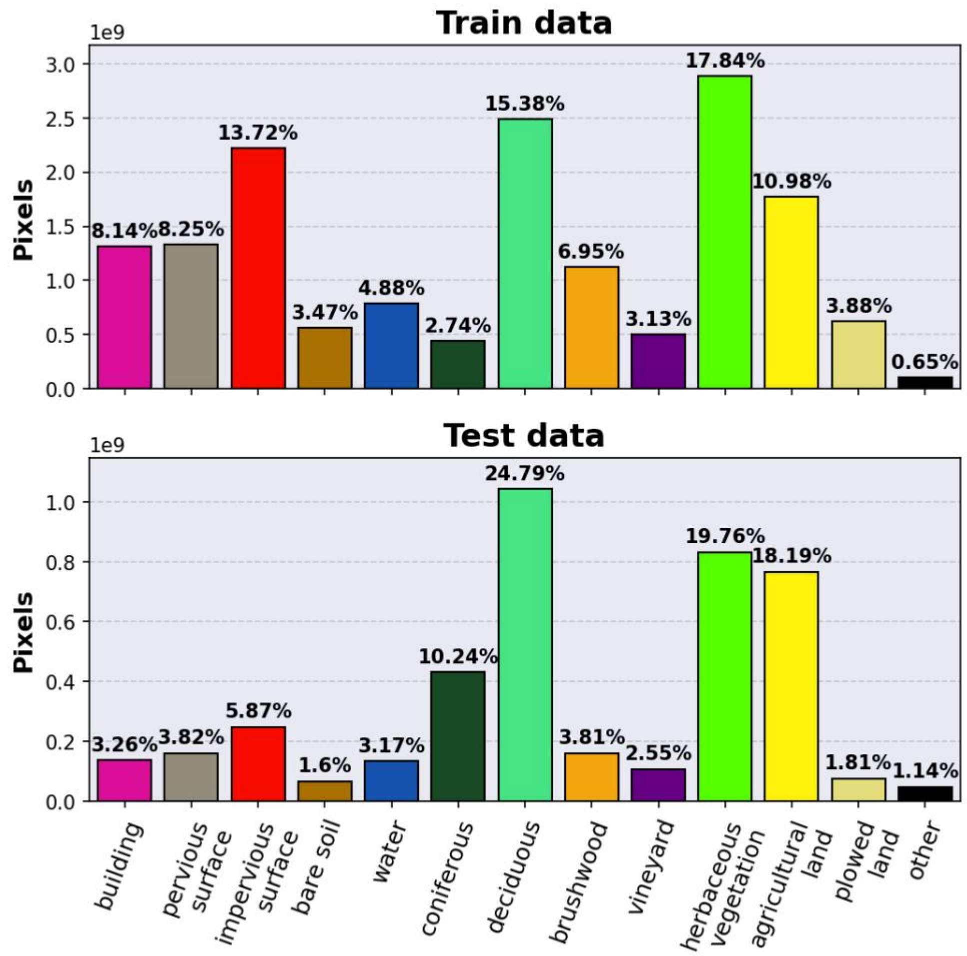


Figure 5: The distribution of pixels on the training and test set. Here, the train set includes the validation set. The image is taken from [6].

2.6.6 Privileged Information

Most images are based on RGB which is why near-infrared and elevation data cannot be assumed to be accessible at all times. In this case, we have access to this specific data, and that is the reason why we choose to define these channels as privileged information when it comes to the aerial patches.

3 Method

In the method section, we present which experiments were conducted, and the technical details of how they were run. The section starts with some general notes on our experimental setup, which holds for all following experiments. After that, we delve into the specific architectures, which all have a corresponding subsection in the results section.

3.1 Experimental Setup

Throughout all experiments, the training and validation were conducted on the full-size images (512, 512). All input data was also normalised to have a mean of 0 and standard deviation of 1, where the mean and standard deviation were calculated from only the training data to not corrupt the validation and test sets. Besides normalisation, some data augmentation was done. It was randomly applied and replaced the original image in that batch. This consisted of up-down and left-right flips with a probability of 0.5, and color jittering with a probability of 0.2. These are similar to the one used in the baseline from the release of the dataset [6].

The training, validation, and test set splits were taken from FLAIR 2 [6]. This means all three sets contained mutually exclusive domains of France. The model utilised for testing was the model that received the lowest validation set loss, which was checked every epoch. When validating and testing, we always use Dice loss between $\hat{y}$ and y to assure comparability between the results for all architectures, regardless of which loss was used for training. The training batch size was 8, and the validation and test set batch size was 2. To receive reproducible results, we used random seed 42 for all randomised values. In terms of optimiser, we used Adam with a learning rate of 0.0002.

All experiments were run on both 100 % and 5 % of the dataset. The 5 % always consisted of the same subset and was drawn randomly from the entire dataset. We ran experiments on a subset to resemble the situation where less data is available, and to study if privileged information might have a greater impact when data is scarce. Similar procedures can be observed in [2]. Furthermore, using less data gives greater practical possibilities to run many experiments. For all 5 % trainings the model was also validated on 5 % of the dataset, but tested on the full test set. This is since the validation set actually impacts the model, while the test set is only used to test the performance of a model.

The number of times our models were trained on the input dataset, i.e. the number of epochs, varied throughout all experiments. The time it took to reach convergence simply depended on the choice of architecture, loss and network, to mention a few things. In general, most experiments were ran for 50-300 epochs before convergence was achieved, with some margin.

Since the dataset consists of 12 common classes, and the seven last classes, $13-19$, together constitute less than 1% of all data, we have chosen to focus on the first 12 classes. For all experiments, the mIoU, mean precision and mean recall will be reported. Unless otherwise is stated, this will refer to the average for the first 12 classes. We still have results for the last six classes, but we choose not to report these as they are less relevant. All models were, however, trained on all classes, meaning all classes were included in the loss function.

3.2 Choice of Loss Function and Neural Network

We needed to decide on an appropriate loss function and a suitable neural network. When choosing, we wanted a combination that performed well, could make use of the privileged information, and all within reasonable time. Since cross entropy loss and Dice loss are two commonly used loss functions within semantic segmentation, we limited our experiments to these two. We also chose to try two pretrained networks, ResNet50 and U-Net. Using these losses and networks, we trained a model on 100% of the data using all five channels (RGB, NIR and elevation). The model that yielded the highest performance was then used throughout the rest of the experiments.

3.3 Added Value of Privileged Information

As previously discussed, our initial step was to verify that the available privileged information could indeed add value. We achieved this by training a model with access to privileged information during both training and inference, referred to as a privileged model. This should be distinguished from a LUPI model, which only has access to privileged information during training. Additionally, we trained a model, which had access only to RGB data, and no privileged information during either training or inference. Such as model will be denoted as an RGB model. The objective was to determine if there was a performance gap between the privileged model and the RGB model. If no such gap was found, further efforts to develop a LUPI model would likely be futile.

To investigate the existence of this performance gap, we trained both a privileged model and an RGB model using 5 % of the dataset, and again using 100 % of the dataset. The performance of the privileged model was then compared to that of the corresponding RGB model. Any observed performance gap would establish the upper and lower bounds for the subsequent application of the LUPI paradigm.

3.4 Choice of Privileged Information

One of our objectives was to assess which channel of privileged information that is most useful. This was done by training our model on RGB + NIR, and RGB + elevation separately. By doing this, we could separate how the addition of each type of privileged information affected the base model. This was done on the entire dataset.

3.5 Metadata

Apart from attempting the LUPI setups, we wanted to assess the usage of metadata. This was not used as privileged data, used as additional information to improve the model. One challenge concerning the metadata is that it is low-dimensional compared to the image data (6 data points per image, compared to $5 \times 512 \times 512$ for the image itself). It is not obvious how to best rebuild the network architecture to utilise this type of data. We focused on mid fusion and late fusion. In short, our mid fusion attempts consisted of passing the metadata through a small linear net, and then inserting it in the middle of the U-Net by using SE blocks. Read this paper [11] for more information on SE blocks. However, after initial tests we decided to focus on late fusion as the performance was better, and the general idea was more intuitive. Late fusion instead means having a small linear net (three layers deep) which outputs a vector with the same length as the number of classes. This vector represents a probability for each class, based only on the metadata. As the metadata contains no pixel specific information we can not make pixel wise predictions using this, only predictions that are general for the entire image. There are various ways of inserting this information into the main net, but in the end we went with a simple linear sum for the predictions. This means we took the probabilities based on the metadata and reconstructed it to fit the size of a full image, but containing the same value in every pixel. We then took this metadata probability and multiplied it with a weight w, and added it to the (pixel wise) predictions made by the U-Net multiplied with $(1 - w)$. w was set to 0.3 after running some quick tests. It was also attempted to normalise the metadata predictions by the class sizes to achieve a ”scaling” of class probabilities, but it performed worse.

The intuition behind why this method might be useful can be illustrated by thinking about the two classes agricultural land and plowed land. These are physically the same place, but agricultural land has crops growing, whereas plowed land has no visible plants. Hence, knowing it is July and not December might greatly increase the probability of the farm lands having growing plants.

3.6 Satellite Data

In addition to metadata, we also used the satellite data to improve the results of our best model. Each aerial patch is mapped to a series of satellite images, captured over the course of a year. These images were normalised and augmented in the same way as the aerial images. Since there could be up to 100 satellite images covering a large area on the ground per aerial patch, the images were cropped and processed to reduce the amount of data. The cropping was done so that the aerial and satellite patches were concentric, but the satellite image was always overlapping the aerial image. Trying a similar approach as in [6], the satellite patches were cropped to 40x40 resulting in a 400x400 meters on the ground, compared to 100x100 meters in the aerial images. Even though the resolution of the satellite images are lower than the aerial images, the idea was that pixels surrounding the aerial images can be useful for segmentation within it. The satellite images were further processed by first discarding all images that had a high risk of snow or cloudiness. If either risk was higher than 50 % for 60 % of the pixels or more in an image, it was discarded. We then computed the average over all images per month. This way, one aerial image corresponded to a time series of 12 separate satellite images. This time series was fed through the encoder and decoder of a UTAE net and then fed into SE blocks in the U-Net used for the aerial images. The output probabilities from the UTAE were also used to make predictions, and to calculate a Dice loss $\mathcal{L}_{sat}$ based only on the satellite images. We then trained our model using a loss function $\mathcal{L} = \mathcal{L}_{sat} + \mathcal{L}_{aerial}$, where $\mathcal{L}_{aerial}$ is the regular Dice loss on the output from the U-Net. For more detail about the UTAE net and this setup in general, see the original FLAIR 2 paper [6].

3.7 Naive Model

The most straightforward approach to leveraging privileged information involves training the model with the privileged data and then withholding this data during inference. In this project, we had two main ways of ”withholding this data”, where one was to provide noise instead of the real channels, and the other was to zero out all weights from the additional channels in the first convolutional layer. When we set all weights to zero, we tried doing only that, but also to scale up the remaining three channels with a factor of 5/3. When it comes to providing noise, there are various types to give and we focused on what we will call salt and pepper noise and normally distributed noise. These will be further discussed below, but in short, salt and pepper noise was defined as randomizing each pixel to the value -5 (”white” or ”salt”) or 5 (”black” or ”pepper”). The normally distributed noise instead sets each pixel to a value taken from a normal distribution with mean 0 and standard deviation 1, and hence the pixel values will be less extreme and more similar to the values the input data normally has.

3.8 Noise Architectures

Another quite naive approach is an extension of the above idea. This is done by providing noise to the model during training, of the same type that is provided during validation. The noise level is not set to

100% immediately, as we want the model to learn from the privileged information first, to then be able to remember it and manage without it. However, the noise level for the validation set is always set to 100%, with the same type of loss as in training, as we want the validation results to resemble the test conditions. The various types of noises the model was trained with closely resemble the ones described above, but are further developed here. To the best of our knowledge, this has not been done before, so we hope to shed some light on a still unexplored area. Also, as the reader will notice, there are many possible combinations from the below derived ideas of how to introduce noise. Hence, we will not be able to present results for all possible combinations but will focus on the most interesting results and observations.

3.8.1 Salt and Pepper

The first type of noise is salt and pepper. As mentioned, the idea of this approach is to set the value of the pixels to either -5 or 5 (representing white or "salt" and black or "pepper" respectively). These are quite extreme values as our data is normalised to mean 0 and standard deviation 1, and hence a value of absolute value 5 or larger should only occur in 0.00003% of cases. The intuition behind this noise is that the model should be able to realize that these extreme values are useless, and learn from the smaller and "more realistic" values, and basically be able to ignore the noisy pixels both in training and validation. Assuming a noise level nl, a fraction nl of the pixels would be randomized according to the salt n pepper scheme.

3.8.2 Image Wise Fade

The second approach we looked at was an image wise fade of the noise. In this method we introduced noise as taken from a normal distribution with mean 0 and standard deviation 1, since the input data is normalised to mean 0 and standard deviation 1. From this an entire image of noise can be created, by sampling from this distribution for each pixel. Assuming a noise level of nl, and denoting the above create noise image as x_{noise} we then get an image with nl level of noise $\tilde{x}$ as $\tilde{x} = x \cdot (1 - nl) + x_{noise} \cdot nl$. This implies that at 100% noise, $\tilde{x}$ will only be noise.

3.8.3 Pixel Wise Fade

The third approach, called pixel wise fade is quite similar to the image wise fade. We draw the noise pixels from the same distribution, but instead of taking the input image and noise image and merging them, we replace the input pixels with noise. This means that if we have a noise level of nl, a fraction nl of the pixels of the original channel will be replaced with pure noise, drawn from the normal distribution with mean 0 and standard deviation 1, while the remaining fraction $1 - nl$ of pixels will be unaltered. This also implies that the image wise fade and pixel wise fade are equivalent once the noise level is 100%.

3.8.4 Zero Out

The last approach, denoted zero out, is quite different from the above mentioned, as we do not really introduce noise in the input, but rather alter the model. This approach is inspired partly by the idea of dropout and pruning. In each batch, we take a fraction of the input weights from the privileged channels, set by the noise level nl, and set them to 0. The weights are chosen randomly in each batch. This leads to that these input weights are forced to 0, and hence might struggle to learn from the privileged data. How well this works also depends on with which speed we introduce the noise, which will be further discussed below.

3.8.5 Batch Wise v.s. Image Wise Noise Enforcement

We have discussed the different types of noise. However, the salt n pepper, image wise fade and pixel wise fade can be introduced in two architecturally very different ways. The first one is batch wise, in which a noise level of *nl* implies that a fraction *nl* of the images in a batch are set to pure noise. Again, in this case the pixel wise fade and the image wise fade are equivalent. The intuitive upside in this is that the model gets to see privileged channels consisting of only noise from the start, which might prepare it better for the transition to full noise. The second way of introducing the noise is denoted as image wise, and sets *nl* of the pixels to noise in all images of a batch.

3.8.6 Noise Level Functions

The final part of this noise model architecture is to decide with which speed the noise should be introduced, i.e. set a noise level function. There are an infinite number of functions available, but for obvious reasons this study will not have time to go through them all. During the training, we have focused on two different types; step wise linear functions and variants of sine functions. There are various things to take into considerations, such that the function significantly impacts the speed of convergence, that the model needs to be trained with 100 % noise at some point. Initially, it also seemed like the performance on validation was poor for noise levels lower than 25%. Based on these considerations and various experiments, we ended up working with the following functions:

- A step wise linear function, shown in figure 6 below. In the image, the x-axis is percentage of number of epochs and the y-axis is the noise level.

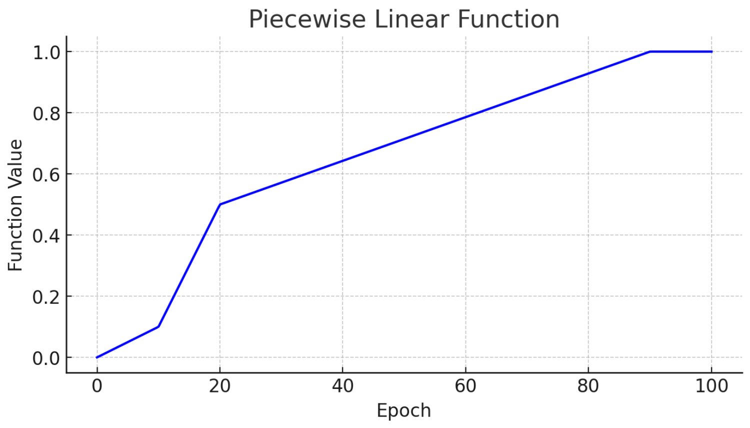


Figure 6: Illustration of the step wise linear noise function.

- A custom sine function, shown in figure 7. Now the x-axis is just epochs, meaning the function had a periodicity of 20 epochs, and the y-axis is again noise level.

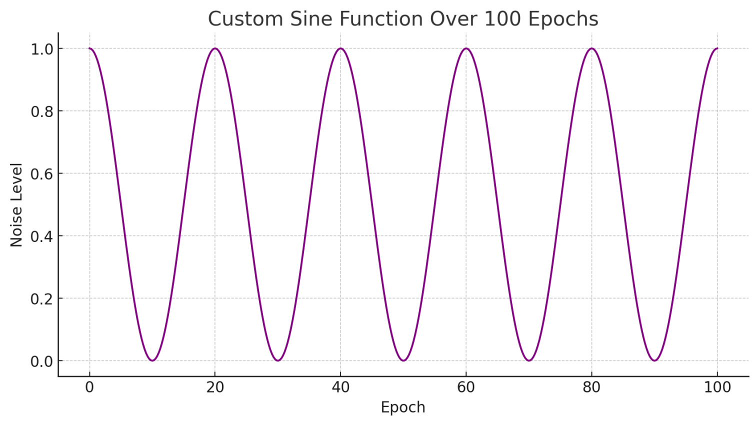


Figure 7: Illustration of the custom sine noise function.

3.9 Teacher Student Architectures

The second main type of LUPI architecture is the teacher student model. Before diving into the technical details of this, we would like to remind the reader of the overall goal of this setup. During testing and inference, the student network will not have access to any type of privileged information. By comparing the teacher's privileged predictions with the student's unprivileged prediction during training, however, the student can find a better approximation of the underlying function that maps RGB images to class probabilities. With that said, we will in this section clarify how the teacher student architecture described in the theory section has been adapted to fit our task.

3.9.1 Our Setup

We started by looking into the choice of teacher model. The first thing we studied was comparing a teacher trained on all five channels, and one trained on only the two privileged channels. The network trained on only privileged information was a lot worse than the one trained on all channels, but the intuition behind why it might work is that the network should then be more different to the student, why it might be easier to transfer knowledge. So, the first round of experiments was to see whether the teacher should have all five channels, or only the privileged channels. These two models were trained on 5% of the dataset.

Secondly, we experimented with some different loss functions for the loss between the teacher and the student. Those attempted were KL divergence, MSE and CE, for which all models were trained on 5%. Once the best loss was found, we experimented a bit with the hyperparameters T_T and T_S. Also, the different losses take values within different intervals and changes with different speeds, why these losses were divided with the reweighting parameter R, to get the losses in approximately the same scale as the Dice loss coming from the student. This parameter is closely related to α, the weight of the teacher loss $\mathcal{L}_T$, but will be reported separately for easier interpretation. α was set to 0.4 for all experiments. We then tried the most promising results on the entire dataset.

Furthermore, since the teacher student architecture resembles an ensemble model (a model where multiple models get to vote), we tried the teacher student architecture on two teacher student pairs where both the teacher and the student had the same input channels. That is to say, we attempted training a student on RGB only, with a teacher who had only been trained on RGB, and one student got access to RGB, elevation and NIR, with a teacher who also had access to all five channels. This experiment was conducted to understand if there was any ensemble effect coming from the architecture itself, and not from the privileged information.

To clarify, when using a teacher that was trained on all channels, we re-used the same teacher for all experiments, and likewise for the teachers trained on only RGB, and only NIR, elevation respectively. Furthermore, when training on 5% of the data, both the teacher and the student model had access to 5%, to to properly simulate a situation with access to less data. Finally, the student was evaluated using the Dice loss, so all losses shown in resutlt section are directly comparable. To be clear, all results show the student's performance, as that is the model we care about.

3.9.2 Representation Layer

A general aim of the teacher-student setup is to make sure that the teacher knowledge is properly transferred to the student model during training. In addition to the teacher-student loss that is computed with respect to the outputs, we also attempted adding a loss computed at the deepest layer in our network. Inspired by [23], we compute the loss between the deepest feature map from both the student and the teacher model. We will refer to this as a representation layer. This loss is then added to the total loss. By doing so, we encourage the student model to mimic the features of the teacher model, and hopefully to perform even better. In practice, we let an image forward propagate through both the networks until it reaches the deepest feature stage, and we then calculate the loss. For the representation layer loss, we tried KL divergence, MSE and CE as loss functions. It would have been best to optimize the hyperparameters together with the choice of loss function, but we limited ourselves to fixating all hyperparameters when finding the best loss. Once the best loss function for the representation layer had been found, we ran one more experiment where we combined the representation layer with the optimized hyperparameters.

3.10 Predicting and Generating Privileged Information Architectures

The third, and final, group of architectures studied were various methods for predicting or generating the privileged information. That is to say, assuming we want our final model to work without privileged information, we train a model which can predict or generate how the privileged information would have looked, if it existed. There are surely many ways of predicting or generating, privileged information, and we will in the following sections go over the few which we have studied more closely. It should be noted that we have not found these architectures in the litterature, but rather constructed them ourselves.

3.10.1 The Loss Components

In all of the below discussed models, there will be a few different loss components. We will explain these components in this section, and for the specific architectures it will be specified exactly how these are utilised.

1. $\mathcal{L}_r$: the regular loss. This loss is the Dice loss calculated based on the prediction $\hat{y}$ and the ground truth y, and is the same as the loss used in all previous experiments. This component is a part of all below architectures.

This first loss component is natural, and needed for the model to perform well. The rest of the loss components are related to the predicted or generated privileged information $\hat{y}_{\text{priv}}$ and the true privileged information y_{priv}. Note that y_{priv} is the same as x^*, but will in this section and the corresponding results section be denoted as y_{priv}, since we use it as labels. These losses are how we actually enforce the privileged information into the net. The three components utilised are

2. $\mathcal{L}_{\text{priv}}$: the loss component directly connected to how similar the predicted or generated privileged $\hat{y}_{\text{priv}}$ is to the true privileged information y_{priv}. $\hat{y}_{\text{priv}}$ consists of one scalar per pixel for the two privileged channels. These values are then compared to y_{priv}, where a MSE, L_1 or KL loss function was used. MSE and L_1 are suitable losses since we need to compare scalars to scalars, and not classes to classes. In other words, a prediction of 3 if the ground truth is 4 is better than 2, while it does not hold for class comparison (there is no reason class 3 would be any more similar to class 4 than class 2 would be.

3. $\mathcal{L}_{\text{priv: mean}}$: a loss component which compares the mean of $\hat{y}_{\text{priv}}$ with the mean of y_{priv} (batch wise). This loss component is inspired by [17]. It is calculated by first taking the mean per image and channel over all pixels from $\hat{y}_{\text{priv}}$, leaving a matrix of size (batch size ×2), where times two is for the two channels. This matrix is denoted $\hat{y}_{\text{mean}}$. The same operation is done on y_{priv}, giving the marix y_{mean}. The entire loss component is then calculated as

$$\mathcal{L}_{\text{priv: mean}} = \frac{1}{\text{batch size} \times \text{num channels}} \sum_{i=1}^{\text{batch size}} \sum_{c=1}^{\text{num channels}} (\hat{y}_{\text{mean}_{i,c}} - y_{\text{mean}_{i,c}})^2.$$

   Here, num channels will in general be 2 as we have 2 privileged channels. The batch size is, as usual, 8. The square is included to assure the value is always positive. This means that we take the mean over all channels and images to get $\mathcal{L}_{\text{priv: mean}}$. Recall, since all channels are normalised, the true mean over the entire dataset is exactly 0, but it might differ slightly for specific batches.

4. $\mathcal{L}_{\text{priv: std}}$: a loss component which compares the standard deviation of $\hat{y}_{\text{priv}}$ with the standard deviation of y_{priv} (batch wise). This loss component is, as the above one, inspired by [17], and is calculated similarly but by taking the standard deviation instead of the mean. This means we first calculate the standard deviation per image and channel over all pixels, $\hat{y}_{\text{priv}}$, leaving a matrix of size (batch size ×2), again times two for the two channels. This matrix is denoted $\hat{y}_{\text{std}}$. Again, we do the same operation on y_{priv}, giving the marix y_{std}. The loss component is then calculated, just as above, as follows

$$\mathcal{L}_{\text{priv: std}} = \frac{1}{\text{batch size} \times \text{num channels}} \sum_{i=1}^{\text{batch size}} \sum_{c=1}^{\text{num channels}} (\hat{y}_{\text{std}_{i,c}} - y_{\text{std}_{i,c}})^2.$$

   Just like before, this means that we take the mean of the standard deviation over the channels and images to get $\mathcal{L}_{\text{priv: std}}$. Recall, since all channels are normalised, the true standard deviation over the entire dataset is exactly 1, but again, it might differ slightly for specific batches.

These are the four loss components that were utilised. It is probable that $\mathcal{L}_r$ and $\mathcal{L}_{\text{priv}}$ are the most important ones, whereas $\mathcal{L}_{\text{priv: mean}}$ and $\mathcal{L}_{\text{priv: std}}$ help the model focus on correct statistical properties. As is hopefully clear, if $\mathcal{L}_{\text{priv}} = 0$, then $\mathcal{L}_{\text{priv: mean}} = 0$ and $\mathcal{L}_{\text{priv: std}} = 0$, while the opposite does not hold.

3.10.2 Multi Decoder Architecture

The first, and maybe simplest approach, will be referred to as a multi decoder architecture, and an illustration is shown in figure 8. Using this approach, we add two extra decoder heads to the standard U-Net, one for NIR prediction and one for elevation prediction. As usual, the RGB image is passed into the encoder which generates features. These are then passed into the "regular" RGB decoder, which decodes the features, to then pass them through a segmentation head, which makes a prediction. This decoder head is denoted ”land cover decoder”, as it is the decoder utilised to do the regular land cover classification. This prediction is compared to the ground truth to generate a loss (the ”regular” loss) $\mathcal{L}_r$. Furthermore, we also pass the features calculated from the RGB input into the NIR decoder and NIR segmentation head. This part then uses these features to make a pixel-wise prediction of the NIR channel. The elevation decoder works in the exact same way, but for the elevation channel. These predictions are concatenated into one image with two channels, constituting $\hat{y}_{\text{priv}}$. $\hat{y}_{\text{priv}}$ and y_{priv} are then utilised to create $\mathcal{L}_{\text{priv}}$, $\mathcal{L}_{\text{priv: mean}}$ and $\mathcal{L}_{\text{priv: std}}$ as described in section 3.10.1. These three losses are then combined into $\mathcal{L}_{\text{all priv}}$ as $\mathcal{L}_{\text{all priv}} = (\mathcal{L}_{\text{priv}} + \mathcal{L}_{\text{priv: mean}} + \mathcal{L}_{\text{priv: std}})/3$. The final loss $\mathcal{L}$, utilised for the backpropagation is then given by $\mathcal{L} = (\mathcal{L}_{\text{all priv}} + \mathcal{L}_r)/2$. All of this means that we indeed predict the privileged information, but never use the prediction as an input to another net (but do not worry, we will). Instead, this approach can be seen as a complex regularization method, as its predictions of the privileged information forces the weights to overfit less.

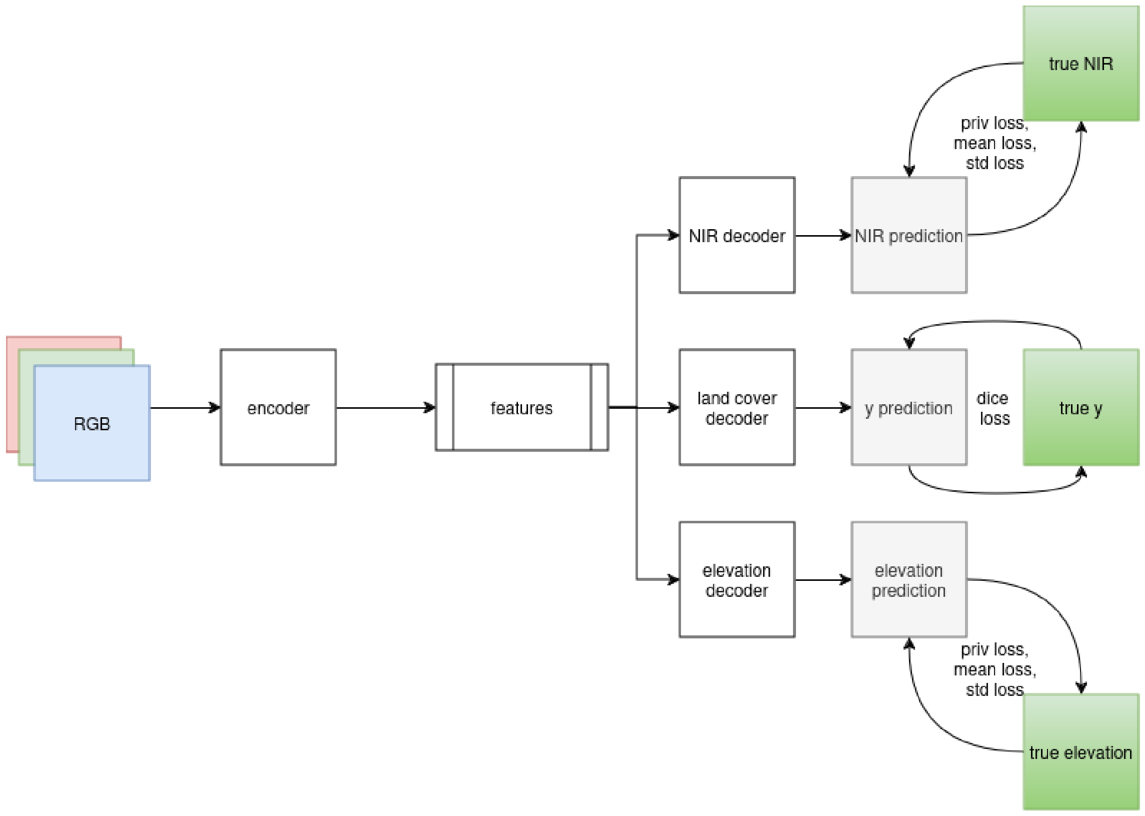


Figure 8: A sketch over the multi decoder architecture. The image was created using the tool draw.io.

3.10.3 Generating Privileged Information in Two Steps

We move on to a second architecture, where we actually use the generated privileged information as input. It consists of two separate networks, one which is called the generator, and one which is called the predictor. A sketch of the architecture can be seen in figure 9. The generator generates the privileged information as one scalar per pixel and channel, based on the RGB input. It also tries to predict the ground truth y as $\hat{y}_{\text{generator}}$. The generator consists of a U-Net with three decoder heads, one for generating NIR, one for generating elevation and one for predicting $\hat{y}_{\text{generator}}$. As in the multi decoder architecture, the generated NIR and elevation are concatenated into $\hat{y}_{\text{priv}}$, which in turn is utilised to calculate $\mathcal{L}_{\text{priv}}$, $\mathcal{L}_{\text{priv: mean}}$ and $\mathcal{L}_{\text{priv: std}}$ (as defined in section 3.10.1). This means that the generator is exactly the multi decoder described in the above section. The tree losses are combined in the same way as above: $(\mathcal{L}_{\text{priv}} + \mathcal{L}_{\text{priv: mean}} + \mathcal{L}_{\text{priv: std}})/3$. Then, we add a loss component $\mathcal{L}_{\text{generator prediction}}$ from $\hat{y}_{\text{generator}}$ and y, calculated using Dice loss. Finally, all of these loss components are combined into the full generator loss $\mathcal{L}_{\text{generator}}$ as $\mathcal{L}_{\text{generator}} = ((\mathcal{L}_{\text{priv}} + \mathcal{L}_{\text{priv: mean}} + \mathcal{L}_{\text{priv: std}})/3 + \mathcal{L}_{\text{generator prediction}})/2$. This loss is utilised to train the generator network.

Now to the second part: the predictor. This is just a normal U-Net as we know it from before, which takes the normal, true, RGB image and the generated privileged information from the generator is input. These are concatenated along the channel dimension, forming an image of the exact same shape as used before. The predictor does the regular land cover classification, and is trained by using the Dice loss on $\hat{y}$ and y.

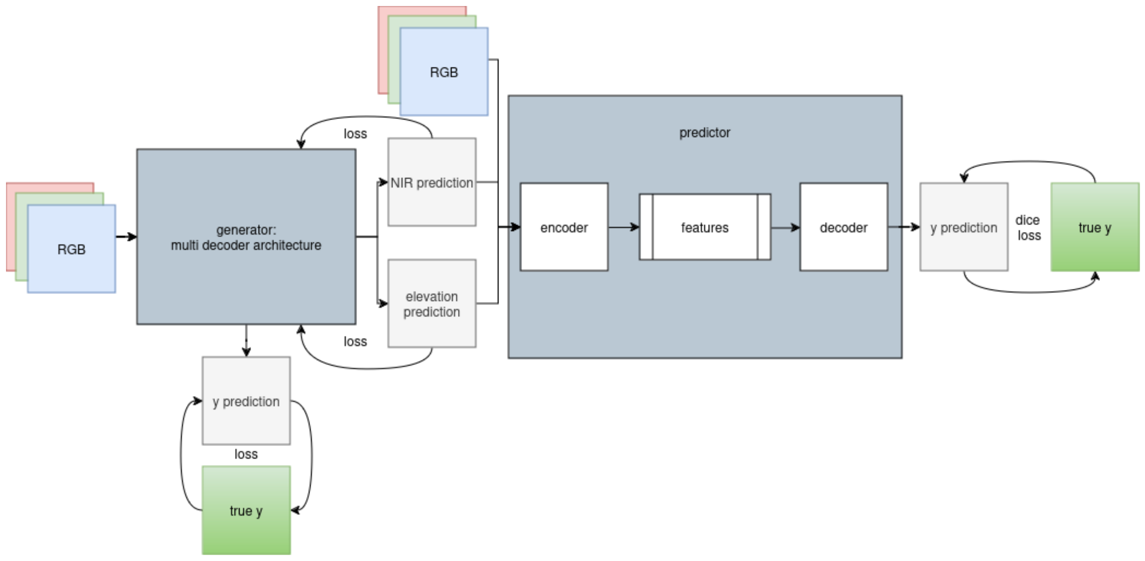


Figure 9: A sketch of the architecture to generate privileged information in two steps. The image was created using the tool draw.io.

3.10.4 Generating Privileged Information in One Step

After reading the previous section, some reader may have wondered: would it not be ideal to let the generator network generate privileged information based on how well the predictor actually performs? At least, so did we. Therefore, we built such a network, very similar to the above architecture, apart from it being only one network, with only one backpropagation. This network is still constructed by one generator and one predictor, but they are now connected. This makes the network twice as deep as before, with the disadvantage that it might be more difficult to train, but with the upside that it might generate better results.

The generator is constructed similarly as in the two step generation above, but it now only has two decoder heads, one for each privileged channel, while the third one for predicting y is removed. This means that it is a regular U-Net, but with two decoder heads, instead of one. An illustration of this network architecture can be seen in figure 10 below. As before, each of these generate one scalar per pixel per image based on the RGB input, which are then concatenated along the channel dimension to $\hat{y}_{\text{priv}}$. This is, in turn, concatenated along the channel dimension with the RGB image, the same RGB input as given to the generator. This matrix, now with 5 channels is then inputted into the second part of the net: the predictor. The predictor is a U-Net performing land cover classification on the input, as we have seen many times before. This outputs a prediction $\hat{y}$, which is compared to the ground truth y using Dice loss. This is the first loss component $\mathcal{L}_{\text{prediction}}$. As there is one network, there is also only one loss, but one with many components. In the middle of the network (in the end of the generator part), we generate $\hat{y}_{\text{priv}}$, which is utilised to calculate $\mathcal{L}_{\text{priv}}$, $\mathcal{L}_{\text{priv: mean}}$ and $\mathcal{L}_{\text{priv: std}}$, as defined in section 3.10.1. These are, very similarly to before, combined into the loss $\mathcal{L}$ which is used to backpropagate through the network during training. It is defined as $\mathcal{L} = ((\mathcal{L}_{\text{priv}} + \mathcal{L}_{\text{priv: mean}} + \mathcal{L}_{\text{priv: std}})/3 + \mathcal{L}_{\text{prediction}})/2$. This architecture is quite interesting as it in some sense can be seen as the predictor training the generator by trying to force the generator to give as relevant information as possible - which is not necessarily the same as the true privileged information.

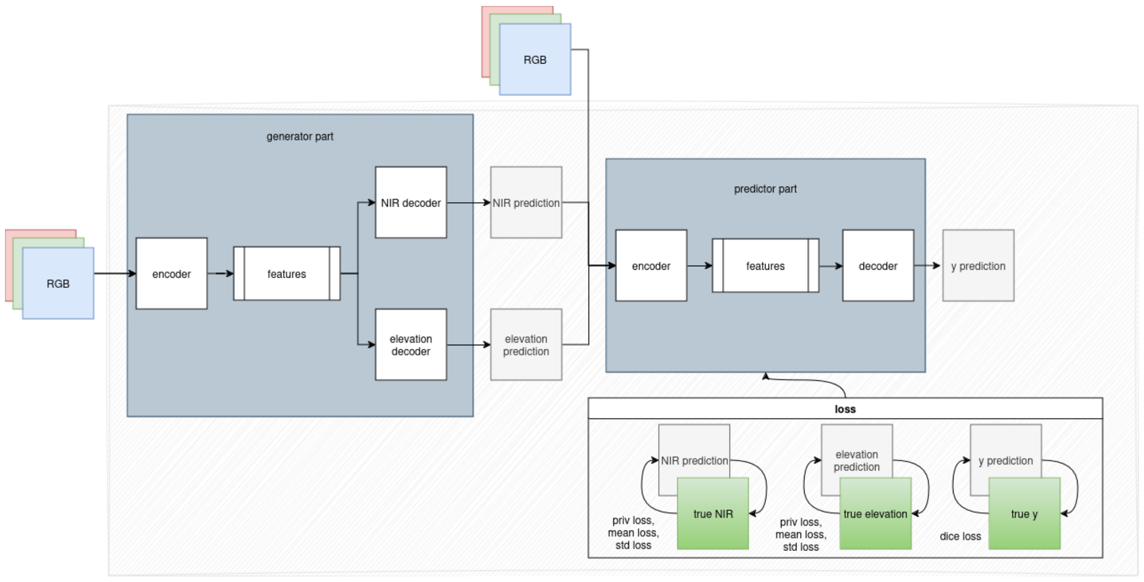


Figure 10: A sketch of the architecture to generate privileged information in one step. The image was created using the tool draw.io.

4 Results

In this section we will present all results. The results are divided into subsections for each architecture type. For any reader not interested in the details, it suffices to read Summary of Results (Section 4.12).

4.1 Choice of Loss Function and Neural Network

We begin by presenting the results which motivated our choice of network and loss function.

4.1.1 Validation Set Results

In table 1, we can see the validation results where we compare ResNet50 to U-Net and Dice loss to CE loss, as well as the epoch from which the result are obtained. All networks are trained on 100% of the data, and on all five channels. As we can see, U-Net outperformed ResNet50, and is also faster to train. Taking these two things into consideration, we chose to use the U-net for all further experiments. In terms of loss function, the Dice loss gives better results than the CE loss, especially on the mIoU, why all further experiments will be conducted using Dice loss. These models will not be ran on the test set, as we only conducted these trainings to decide which network and loss function to use, and we do not want the test set results to impact any choices.

Network	U-Net	U-Net	ResNet50	ResNet50
Loss Function	CE	Dice	CE	Dice
Epoch	6	47	6	50
Loss	0.674	0.148	0.672	0.152
mIoU	0.642	**0.650**	0.626	0.635
Precision	0.780	**0.784**	0.772	0.780
Recall	0.775	**0.781**	0.758	0.769

Table 1: Table over results for the four (net, loss function) pairs. Note that the values of different loss functions are not comparable. The best values are marked in bold.

4.2 Added Value of Privileged Information

Now that we have chosen a network and a loss function, we will study how large the gap is between the models trained and validated with and without privileged information. These experiments have been conducted using both 5 % and 100 % of the data. Going forward, the models presented here will be referred to as the RGB baseline, and the privileged baseline. As the name suggests, the RGB baseline is trained and validated on the RGB channels, and the privileged baseline on all five channels.

4.2.1 Validation Set Results

Table 2 below shows the results on the validation set for the four models. We can see that the results are better when using privileged information in almost all metrics. It is especially worth noticing that the mIoU is better on both 5% and 100% of the data when using privileged information, and that the gap is somewhat larger between the 5% models.

Input Data	RGB	RGB, NIR, elevation	RGB	RGB, NIR, elevation
Dataset Size	100 %	100 %	5 %	5 %
Epoch	35	47	23	59
Loss	0.156	**0.148**	0.172	**0.167**
mIoU	0.639	**0.650**	0.580	**0.597**
Precision	**0.784**	**0.784**	0.723	**0.741**
Recall	0.762	**0.781**	0.723	**0.734**

Table 2: Table of results for models trained with and without privileged information on 5 % and 100 % of the dataset. The best results are marked in bold, for the 5 % and 100 % models separately.

4.2.2 Test Set Results

In this section we present the test set results. As a reminder, all results are obtained from the entire test set, regardless of whether the model has been trained using 5 % of the training dataset or 100 %, as these are only to evaluate the performance of the model. We can see, in table 3, that the results hold from the validation set. More specifically, the gap increases between the 100% models, but gets slightly smaller between the 5% models.

Input Data	RGB	RGB, NIR, elevation	RGB	RGB, NIR, elevation
Dataset Size	100 %	100 %	5 %	5 %
Loss	0.147	**0.141**	0.166	**0.159**
mIoU	0.579	**0.596**	0.543	**0.552**
Precision	0.712	**0.725**	0.679	**0.682**
Recall	0.735	**0.748**	0.697	**0.719**

Table 3: Table of results for the models trained with and without privileged information on 5 % and 100 % of the dataset. The best results are marked in bold, for the 5 % and 100 % models separately.

In addition to the presented metrics, figure 11 shows the class-wise IoU for the 100 % models on the test set. The privileged model outperforms the RGB model by a bit in most classes, but there is no particular class which is outstanding. Nevertheless, the RGB model does outperform the privileged model in class 2, 4, 9 and 10 by a small margin.

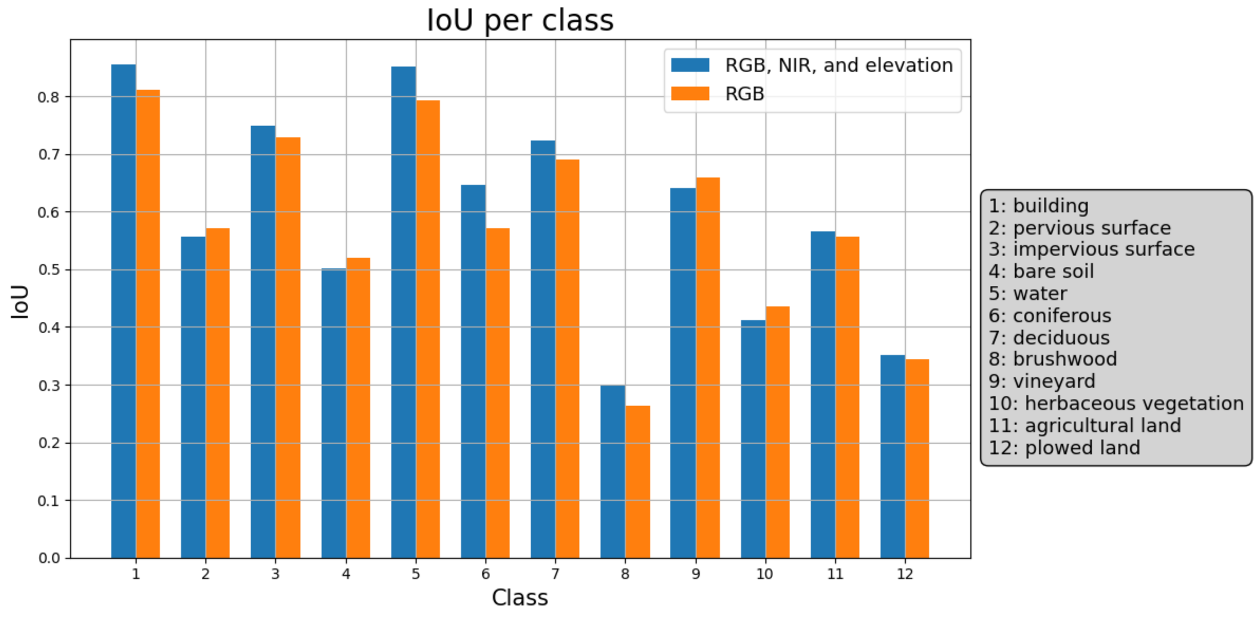
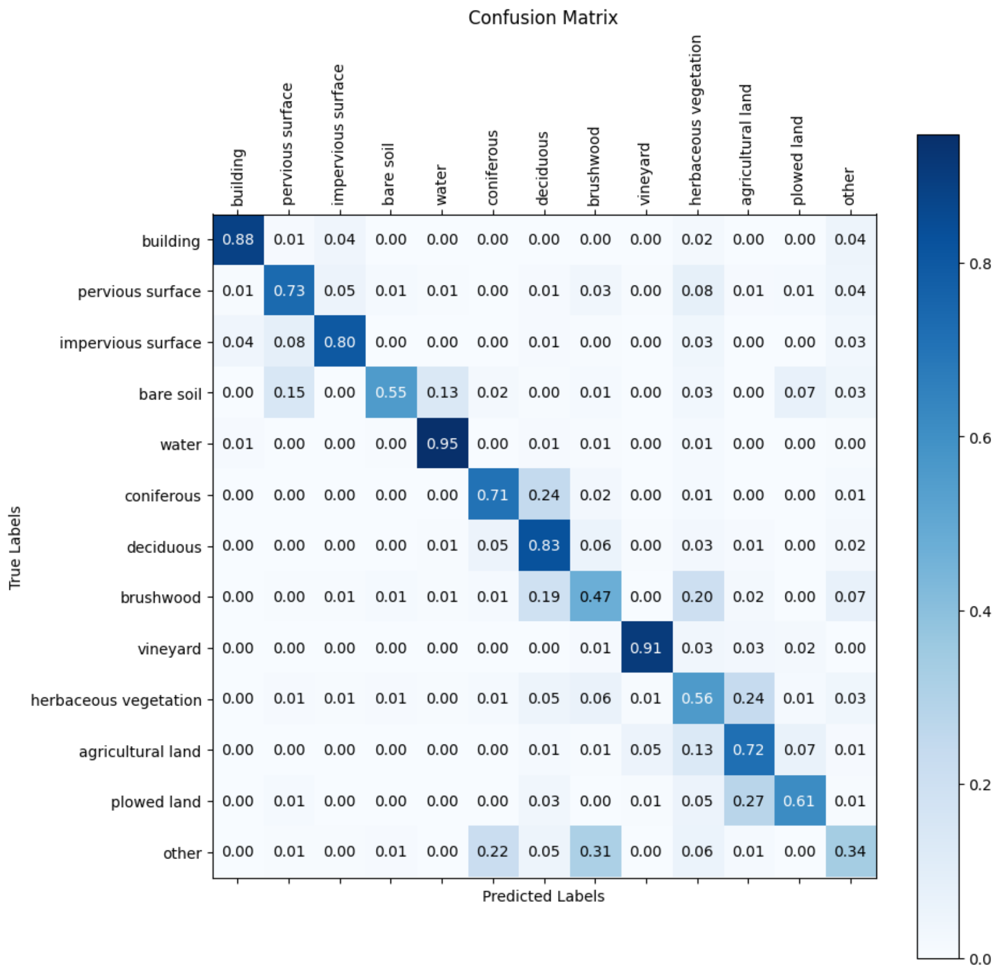


Figure 11: IoU per class comparing the model with only RGB and the one with RGB, NIR and elevation trained on 100 %.

Finally, we also show confusion matrices for the 100 % models, but only for 5 % of the test set, due to computational limitations. The 13th row and column is denoted ”other” and represents the seven last classes, grouped together. The confusion matrix for the RGB model can be seen in figure 12 and the confusion matrix for the privileged model can be seen in 13. As we can see, both models struggle to separate herbaceous vegetation, agricultural land and plowed land from each other, which is not very surprising as these classes are similar. Recall, agricultural land and plowed land only differ depending on whether there are plants growing or not. Another thing is that he privileged model struggles less than the RGB model with separating coniferous, deciduous and brushwood.



	building	pervious surface	impervious surface	bare soil	water	coniferous	deciduous	brushwood	vineyard	herbaceous vegetation	agricultural land	plowed land	other
building	0.88	0.01	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.04
pervious surface	0.01	0.73	0.05	0.01	0.01	0.00	0.01	0.03	0.00	0.08	0.01	0.01	0.04
impervious surface	0.04	0.08	0.80	0.00	0.00	0.00	0.01	0.00	0.00	0.03	0.00	0.00	0.03
bare soil	0.00	0.15	0.00	0.55	0.13	0.02	0.00	0.01	0.00	0.03	0.00	0.07	0.03
water	0.01	0.00	0.00	0.00	0.95	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00
coniferous	0.00	0.00	0.00	0.00	0.00	0.71	0.24	0.02	0.00	0.01	0.00	0.00	0.01
deciduous	0.00	0.00	0.00	0.00	0.01	0.05	0.83	0.06	0.00	0.03	0.01	0.00	0.02
brushwood	0.00	0.00	0.01	0.01	0.01	0.01	0.19	0.47	0.00	0.20	0.02	0.00	0.07
vineyard	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.91	0.03	0.03	0.02	0.00
herbaceous vegetation	0.00	0.01	0.01	0.01	0.00	0.01	0.05	0.06	0.01	0.56	0.24	0.01	0.03
agricultural land	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.05	0.13	0.72	0.07	0.01
plowed land	0.00	0.01	0.00	0.00	0.00	0.00	0.03	0.00	0.01	0.05	0.27	0.61	0.01
other	0.00	0.01	0.00	0.01	0.00	0.22	0.05	0.31	0.00	0.06	0.01	0.00	0.34

Figure 12: Confusion matrix for the RGB baseline trained on 100 % of the data.

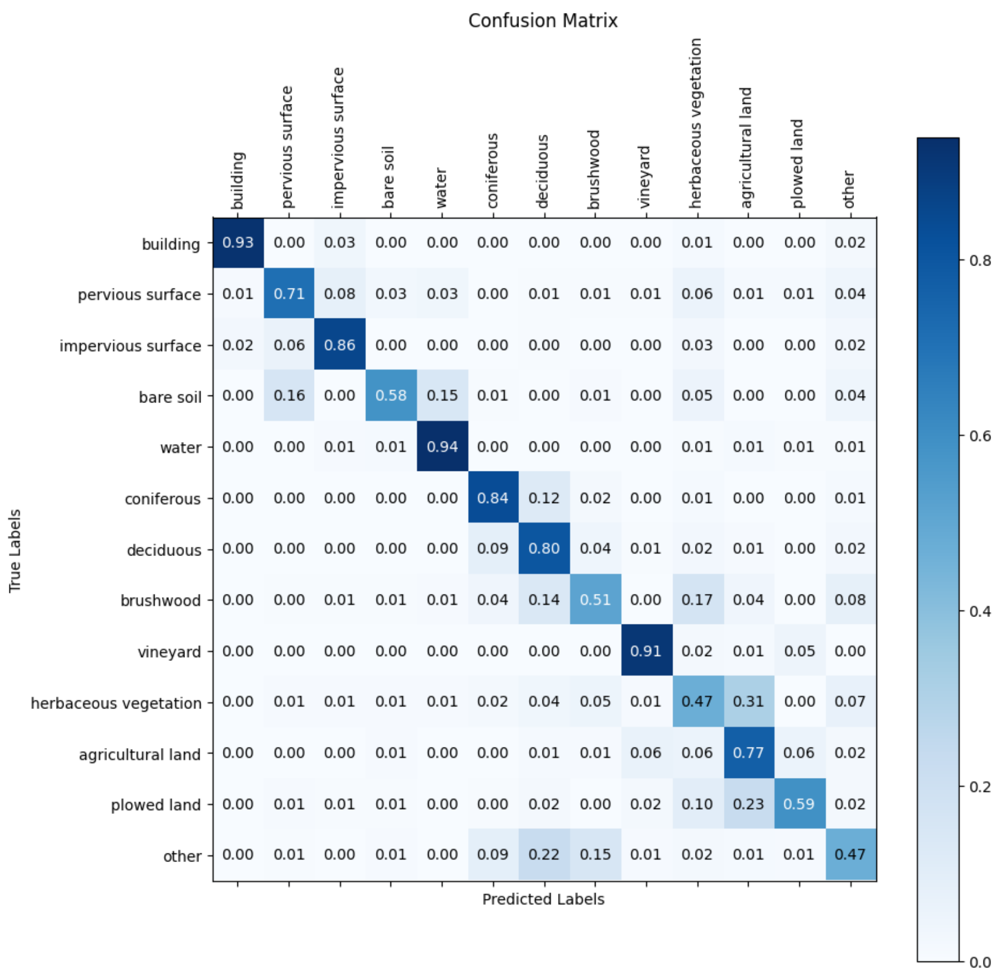


Figure 13: Confusion matrix for the privileged baseline trained on 100 % of the data.

4.3 Choice of Privileged Information

For the above section, both NIR and elevation were used as privileged information. In this section, we will present results from models that trained with only NIR or elevation as privileged information. The hope is to understand which type of privileged information that is most useful.

4.3.1 Validation Set Results

Table 4 below shows the validation set results from using the different channels of privileged information on the entire dataset, compared to the baselines obtained in 4.2. We can see that the model that only has NIR as privileged information performed better on almost all metrics compared to using NIR and elevation. In terms of mIoU, which is our most important metric, the difference was merely 0.001 (0.651 vs. 0.650). Given that this difference is so small, we chose to include both NIR and elevation in our future models, hoping that the other architectures would be better at using this additional information that the elevation channel provides.

A possible explanation to the decrease in performance, is that the elevation channel is computed from RGB and NIR, and does not provide new information. Although we do not include any class-wise metrics here, it seems like the addition of NIR as input increases the mIoU especially for vegetation classes.

Input Data	RGB, NIR, elevation	RGB, NIR	RGB, elevation	RGB
Epoch	47	56	87	35
Loss	**0.148**	**0.148**	0.154	0.156
mIoU	0.650	**0.651**	0.634	0.639
Precision	0.784	**0.787**	0.777	0.784
Recall	**0.781**	0.779	0.767	0.762

Table 4: Table of results for models trained with different privileged information. The best results are marked in bold.

4.4 Metadata

In this section we will study the usage of metadata, according to the principles described in the methods section. From these results, we hope to be able to conclude if metadata should be included in our model or not.

4.4.1 Validation Set Results

The following section presents the validation set results from incorporating metadata into our model trained on all five channels. These are shown in table 5 for both 5 % and 100 % of the dataset. To choose whether or not to include metadata, we compare it to our previously best model with all five channels of input but without metadata. As can be seen in the table, the mIoU improves for the 5% model, but remains the same for the one on 100%. Given that the improvement does not hold for the entire dataset, we chose to not include metadata, since it gives us a slightly simpler model. One interesting result for the 100 % model is that the metadata improved the IoU for agricultural and plowed land, despite the general lack of improvement on the mIoU. Since agricultural and plowed land have a time-dependency, it could be that the time stamps in the metadata helps in distinguishing the two classes. Again, these models will not be run on the test set since we only wanted to decide whether to include metadata in the model, or not.

Metadata	False	True	False	True
Dataset Size	100 %	100 %	5 %	5 %
Epoch	47	65	59	184
Loss	0.148	**0.147**	0.167	**0.163**
mIoU	0.650	0.650	0.597	**0.615**
Precision	0.784	**0.788**	0.741	**0.760**
Recall	0.781	0.776	0.734	**0.746**

Table 5: TValidation set results for models trained with and without metadata on 5 % and 100 % of the dataset. Any results where the metadata model beats the corresponding baseline are marked in bold.

4.5 Satellite Data

Moving on from metadata, we here present the experiments that were performed with satellite images as input, in addition the the aerial images. As for the metadata, the hope of this study is to conclude if satellite data should be included in our model or not.

4.5.1 Validation Set Results

Table 6 shows the results for models trained with and without satellite data, and we compare it to the best model from the aerial input on both 5 % and 100 % of the data. All models are trained using all five channels. To summarize, all metrics get worst when including the satellite images on 100% of the data, but slightly improves the mIoU and precision on 5%. Since mIoU drops by almost 2 percentage units when using the entire data set, the satellite images will not be part of the input to our model moving forward.

Satellite Data	False	True	False	True
Dataset Size	100 %	100 %	5 %	5 %
Epoch	47	19	59	76
Loss	0.148	0.186	0.167	0.202
mIoU	**0.650**	0.631	0.597	**0.601**
Precision	**0.784**	0.779	0.741	**0.769**
Recall	**0.781**	0.762	**0.734**	0.728

Table 6: Validation set results for models trained with and without satellite data on 5 % and 100 % of the dataset. The best results are marked in bold, for 5 % and 100 % separately.

4.6 Naive Model

This section presents the results from our first naive LUPI approach. Recall, this trivial idea was simply to train the model with all five channels, and then substitute the NIR and elevation channels with noise when evaluating.

4.6.1 Validation Set Results

The results for the validation set can be found in table 7. All results are obtained from the privileged baseline model that was trained on 100% of the data. None of the metrics were even close to as good as those of the RBG baseline, hence the lack of numbers in bold. Even though the zero out models do not beat the RGB baseline, they still outperform the other noise models with a great margin in all metrics. As the results are so poor, we will not present any test set results.

Noise Type	N/A	s & p	normal noise	zero out	reweighted zero out
Loss	**0.156**	0.361	0.362	0.246	0.229
mIoU	**0.639**	0.025	0.050	0.377	0.392
Precision	**0.784**	0.059	0.239	0.634	0.605
Recall	**0.762**	0.080	0.131	0.535	0.558

Table 7: Validation set results for the model trained with privileged information on 100 % of the dataset, evaluated without the privileged information. S & p stands for salt n pepper. In the first column we see the RGB baseline results, which all are marked in bold since they outperform all other models.

4.7 Noise Architectures

In this section we will study our second, slightly less naive, LUPI approach: the noise models. The noise is now part of the training, and not only introduced at evaluation.

4.7.1 Validation Set Results

In table 8, we can see the results for the various noise models trained on 5 % of the data, compared to the RGB baseline seen in the first column. As we can see, the only model which might outperform the baseline is the zero out model, which is better than the baseline for most metrics, especially for mIoU. Hence, we discard all models except for the zero out model at this stage, and continue by training the zero out model on the entire dataset. In general, all noise models take quite many epochs to converge. This is likely since we do not enforce full noise during training until very late, and before that the model is still unprepared for receiving only noise. Furthermore, the zero out model converges faster and seems less sensitive to the training noise level, which is aligned with the results from the naive model in 4.6.

Noise Type	N/A	pixel fade	img fade	img/pix fade	s & p	s & p	s & p	zero out
Image/Batch	N/A	image	image	batch	image	batch	batch	N/A
Noise Function	N/A	linear	linear	linear	linear	sine	linear	linear
Epoch	23	183	179	191	197	181	198	102
Loss	0.172	0.178	0.182	0.178	0.181	0.181	0.180	0.173
mIoU	0.580	0.556	0.548	0.553	0.546	0.542	0.552	**0.585**
Precision	0.723	0.722	**0.731**	0.717	0.717	0.716	0.714	**0.737**
Recall	0.723	0.689	0.677	0.697	0.686	0.683	0.691	0.720

Table 8: Validation set results for the noise models, compared to the RGB baseline. Results that outperform the baseline score are marked in bold. S & p stands for salt n pepper and linear is short for the stepwise linear function described in 3.8.6.

In table 9 below, we can see the result for the zero out model trained on the full dataset, compared to the 100% RGB baseline. The zero out model is outperformed by the baseline, even though the performances are comparable.

Noise Type	N/A	zero out
Noise Function	N/A	linear
Epoch	35	43
Loss	0.156	**0.155**
mIoU	0.639	0.631
Precision	0.784	0.771
Recall	0.762	0.759

Table 9: Test set results for the zero out model, compared to the RGB baseline. Results which outperform the baseline scores are marked in bold.

4.7.2 Test Set Results

We now move on to the test set results for our best model, the zero out model, trained on 100% and 5% of the data. We compare them to the corresponding RGB baselines, seen in columns one and three respectively. The results are shown in table 10, where we see that none of the zero out models outperform their corresponding RGB baseline.

Noise Type	N/A	zero out	N/A	zero out
Dataset Size	100 %	100 %	5 %	5 %
Loss	0.147	0.148	0.166	**0.165**
mIoU	0.579	0.577	0.543	0.543
Precision	0.712	0.712	0.679	**0.689**
Recall	0.735	0.733	0.697	0.686

Table 10: Test set results for the zero out models trained on 100% and 5% of the data respectively. Any metric that outperforms the RGB baseline on the corresponding dataset size is marked in bold.

For more detailed information on how well the zero out model performs on the test set, we include a plot of the class-wise IoU in figure 14. There are no notable differences in performance of the two models. Compared to the RGB baseline, the zero out model performs better on half of the classes.

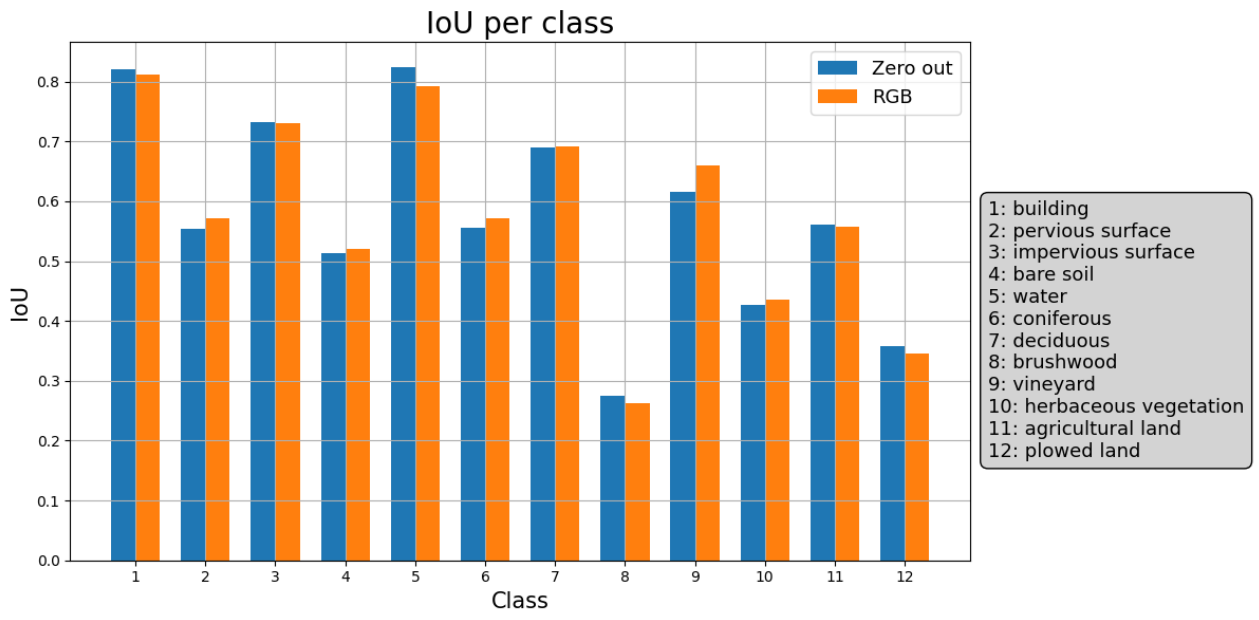


Figure 14: IoU per class comparing the model with only RGB and the zero out model on 100 %.

Finally, we again show confusion matrix for the 100 % model, but only for 5 % of the test set, due to computational limitations. The 13th row and column is denoted ”other” and represents the seven last classes grouped together. The confusion matrix for the 100% zero out model can be seen in figure 15. The confusion matrix does not differ from the RGB baseline confusion matrix (in figure 12) in any remarkable way.

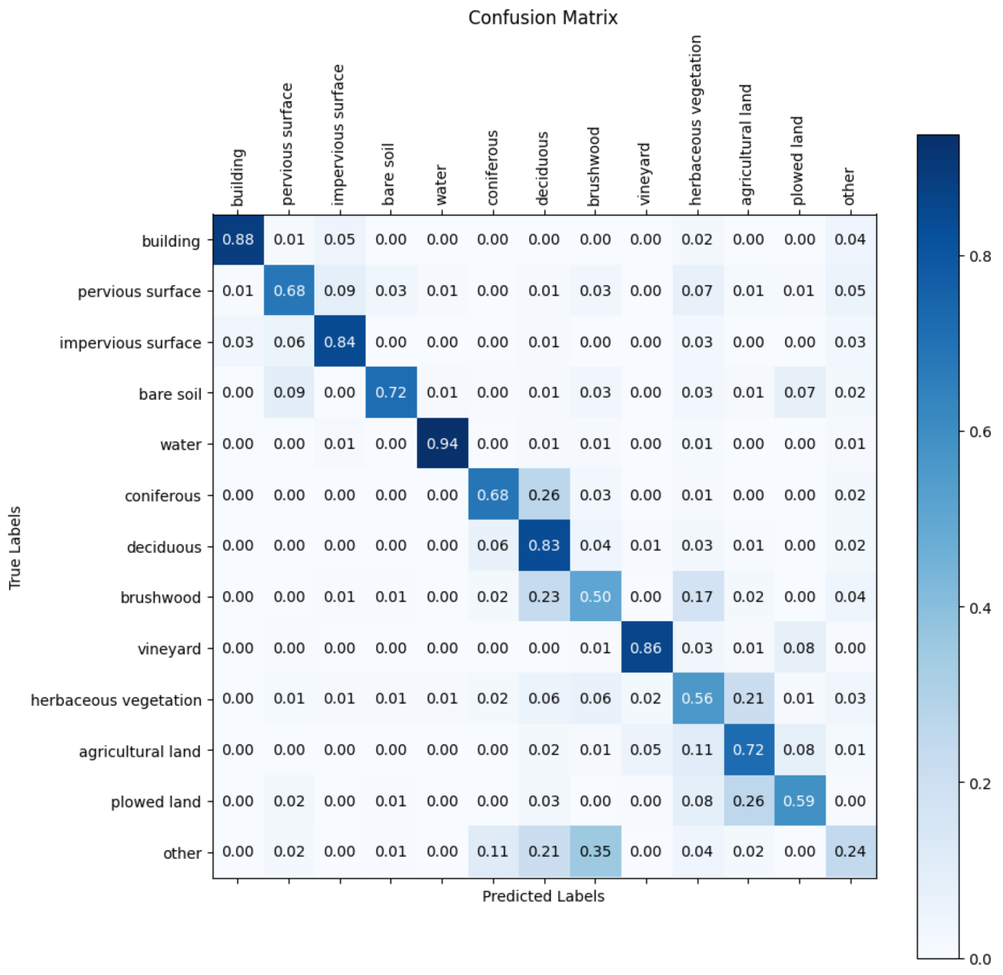


Figure 15: Confusion matrix for the zero out model trained on 100 % of the data.

4.8 Teacher Student Architectures

In this section, we will present results for the teacher student (TS) architectures described in section 3.9. This is our second main category of LUPI experiments. We will present validation results for five rounds of experiments. The first four rounds are run only on 5% of the dataset. Round one is to decide the input to the teacher model, round two to decide the loss function between the teacher and the student, round three to set hyperparameters, and round four to introduce the concept of representation layers. Finally, in round five we train the most promising models on the full dataset. In the second round we will also present results for models where the teacher and the student had access to the same information. Finally, we will present the results for the most promising models on the test set.

4.8.1 Validation Set Results: Choosing Teacher

In table 11, below we see the results for the first round of tests for the teacher student architecture. In this experiment we try different inputs to the teacher model, one with RGB, NIR and elevation, and one with NIR and elevation only. Both models are trained with KL divergence as their teacher student loss, on 5% of the dataset, with $R = 1/100000$, and no temperature T_T or T_S. We also compare these results to the 5% RGB baseline, seen in the first column. As we can see, both models clearly outperform the baseline in all metrics except for loss. The two different inputs to the teacher give quite similar results, but due to the lower loss of the model with a teacher trained on all channels, we will focus on that model. Throughout the training, that model had a more stable validation behaviour, leading us to believe that it generalizes better even though performing slightly worse in some metrics. Hence, going forward all teachers will have all five channels as input.

Teacher Input	N/A	RGB, NIR, elevation	NIR, elevation
TS Loss	N/A	KL	KL
Epoch	23	184	121
Loss	0.172	0.175	0.185
mIoU	0.580	**0.600**	**0.601**
Precision	0.723	**0.736**	**0.751**
Recall	0.723	**0.746**	**0.733**

Table 11: Table of results for teacher student models trained with different inputs to the teacher. Also, we present the baseline trained with only RGB for comparison, and mark the results in bold when the models outperform the baseline. TS loss is short for the loss function between the teacher's and the student's prediction.

4.8.2 Validation Set Results: Choosing Teacher Student Loss Function

Having chosen the input to our teacher model, we moved on to the second round of experiments for the teacher student architecture. In table 12 below, we can see one model where both the teacher and the student have RGB as input, and one model where the teacher and the student have RGB, NIR and elevation as input. We choose to include these models where the teacher and the student have the same input, to separate how much of the model improvements that come from the teacher student architecture itself, and how much that comes from using LUPI. As we can see in the first four columns, the architecture itself improves the baseline quite a lot. There can be many reasons for this, and these will be developed in the discussion. We will considered these models as baselines for all teacher student model, and are called TS (teacher student) baselines below. Table 12 also includes three LUPI models with a teacher that is trained on RGB, NIR, elevation and the student trained on only RGB, but with three different types of losses. Column 1 and 2 show the results for the regular baselines, for comparison. Note that since the architecture itself improves the baseline, we will mark the results in bold only if it beats the result of the teacher-student model, with both the teacher and student trained on only RGB. All models are trained on 5% of the dataset. Again, no temperature is used and the reweighting factors R are set to match the scale of the teacher student loss function and the "regular" Dice loss function.

Compared to the RGB TS baseline, the LUPI teacher student models are even better. It also seems like the KL loss is the best, why that is the one which will be used going forward. Another observation is that the losses are in general higher for the teacher student setups, even though the results are improved. Possible reasons for this will be discussed later.

	Baseline		TS Baselines		LUPI		
Teacher Input	N/A	N/A	RGB	RGB, priv	RGB, priv	RGB, priv	RGB, priv
Student Input	N/A	N/A	RGB	RGB, priv	RGB	RGB	RGB
TS Loss	N/A	N/A	KL	KL	KL	MSE	CE
R	N/A	N/A	1/100000	1/100000	1/100000	5	1
Epoch	23	59	295	90	184	160	91
Loss	0.172	0.167	0.172	0.166	0.175	0.172	**0.169**
mIoU	0.580	0.597	0.599	0.639	**0.600**	**0.600**	0.597
Precision	0.723	0.741	0.740	0.780	0.736	**0.748**	0.735
Recall	0.723	0.734	0.738	0.766	**0.746**	0.739	**0.746**

Table 12: Validation results for teacher student models trained with different losses between the teacher student prediction. We also present TS baselines and the two regular baselines, for comparison. The best metric out of the three LUPI models is marked in bold.

4.8.3 Validation Set Results: Setting the Temperature

In table 13 below we can see the results for the third round of teacher student experiments, where we bring in the temperatures T_T and T_S. All models have KL loss between the teacher and student prediction, and the reweighting parameter R is again set to match the scales between the two loss components. For comparison, we also show the previously best LUPI teacher student model. All models are trained on 5% of the dataset, using a teacher with all channels as input, and RGB as the input to the student. This means that all models in the table below are LUPI models.

As we can see, temperature seems to be useful, as suggested by literature. Exactly how much, and if both the student's and the teacher's predictions should be smoothed, is a more complicated question. However, the model with the temperature pair $(2, 7)$, shown in the last column, performs best, why those temperatures primarily will be used forward. Furthermore, we can see that the model where only the teacher's prediction is smoothed has a relatively high validation loss (recall, the validation loss is the usual Dice loss, hence all losses shown here are directly comparable), but still quite good results. As a side note, we also attempted different temperature for the TS baselines, but they were far less sensitive to temperature changes so the results are omitted in this table.

R	1/100000	1/50000	1/70000	1/65000
$\mathbf{T_S, T_T}$	(1,1)	(5,5)	(1,5)	(2,7)
Epoch	184	205	265	232
Loss	0.175	**0.167**	0.207	0.181
mIoU	0.600	0.617	0.611	**0.623**
Precision	0.748	0.760	**0.762**	0.760
Recall	0.739	**0.752**	0.738	**0.752**

Table 13: Validation results for teacher student models with various values for the teacher's temperature T_T and student's temperature T_S. The first column shows the previously best LUPI model without temperature ($T_S = T_T = 1$), for comparison. The best performance in each metric is marked in bold.

4.8.4 Validation Set Results: Adding a Representation Layer

As a last step, we added a representation layer to the teacher student setup for 5 % of the data. Table 14 shows the results from trying different loss functions when adding this layer. All three teachers are trained with RGB, NIR, and elevation as input, and the reweighting R_2 is used to keep the new loss component on the same scale as the TS loss and regular loss. For the other parameters, we set $R = 1/65000$, $(T_S, T_T) = (2, 7)$ for all below models. As can be seen in the table, none of the different loss functions for a representation layer rendered a mIoU larger than that of the previously best LUPI model in table 13 (0.623). The MSE loss came closest with a mIoU of 0.607 for the 12 first classes. Since the representation layer did not improve the results, we chose to not look into it any further.

Rep. loss	KL	MSE	CE
$\mathbf{R_2}$	0.003	5000	1
Epoch	158	256	205
Loss	0.184	0.184	0.183
mIoU	0.597	0.599	0.607
Precision	0.757	0.753	0.759
Recall	0.728	0.733	0.733

Table 14: Results for teacher student models with different loss functions for the representation layers. None of the metrics are better compared to the best model without a representation layer.

4.8.5 Validation Set Results: 100 %

Finally, we used the best model trained on 5 %, but used it to train on on the full dataset. In table 15, we can see these results. Notably, the results from the 5% experiments hold well to the 100% experiments, as the LUPI model outperforms the RGB TS baseline.

	TS Baselines		**LUPI**
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S, T_T}$	(2,7)	(2,7)	(2,7)
R	1/65000	1/65000	1/65000
Epoch	102	138	193
Loss	0.154	0.146	**0.151**
mIoU	0.651	0.663	**0.656**
Precision	0.791	0.791	**0.790**
Recall	0.773	0.793	**0.781**

Table 15: Results for teacher student models trained on 100% of the data. The LUPI model is shown in the last column and wherever it beats the RGB TS baseline, the metrics are marked in bold.

4.8.6 Test Set Results

As a final step of the teacher student architecture, we present the test set results for our best LUPI model. This is the model with KL divergence as TS loss, the temperatures $(2, 7)$, and no representation layer, trained on 5% and 100% of the data. We also present the test results of the corresponding TS baselines. The results for the model trained on 5% of the data can be seen in table 16, while the results for the model trained on 100% of the data can be seen in 17. As we can see, the results from the validation set holds to the test set, and gets even stronger for the 100% LUPI model, which is almost as good as the privileged TS baseline.

	TS Baselines		**LUPI**
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
Loss	0.166	0.160	0.174
mIoU	0.545	0.558	**0.554**
Precision	0.687	0.687	0.685
Recall	0.707	0.725	**0.718**

Table 16: Test set results for TS baselines and the best LUPI teacher student model trained using 5 % of the dataset. Where the LUPI model beats the RGB TS baseline, the score is marked in bold.

	TS Baselines		**LUPI**
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
Loss	0.148	0.140	**0.143**
mIoU	0.588	0.607	**0.606**
Precision	0.717	0.728	**0.736**
Recall	0.744	0.764	**0.753**

Table 17: Test set results for TS baselines and the best LUPI teacher student model trained using 100 % of the dataset. Where the LUPI model beats the RGB TS baseline, the score is marked in bold.

For the 100 % models, we also plot the IoU per class as a diagram in figure 16. This is done for the LUPI model and the two TS baselines for comparison. We can see that the LUPI model actually outperforms both the TS baselines on class 7, 11, and 12. Furthermore, the LUPI model outperforms the RGB TS baseline in all classes apart from 2, which is the only class where the RGB baseline is the best.

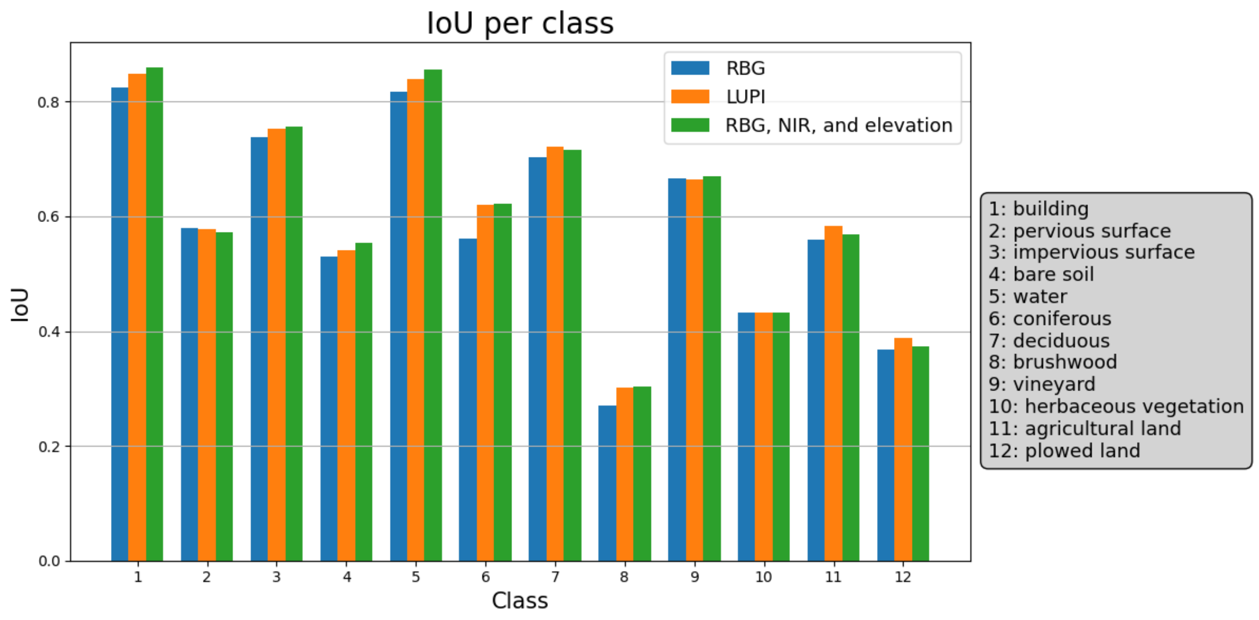
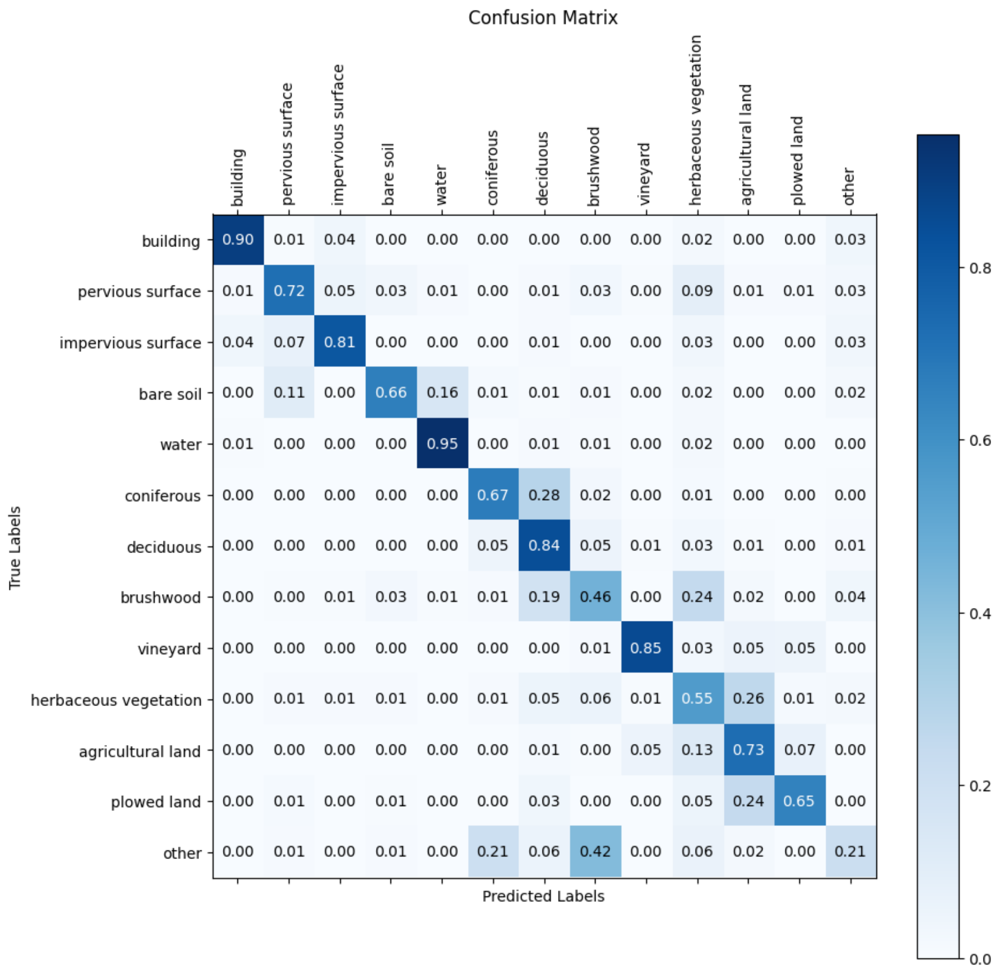


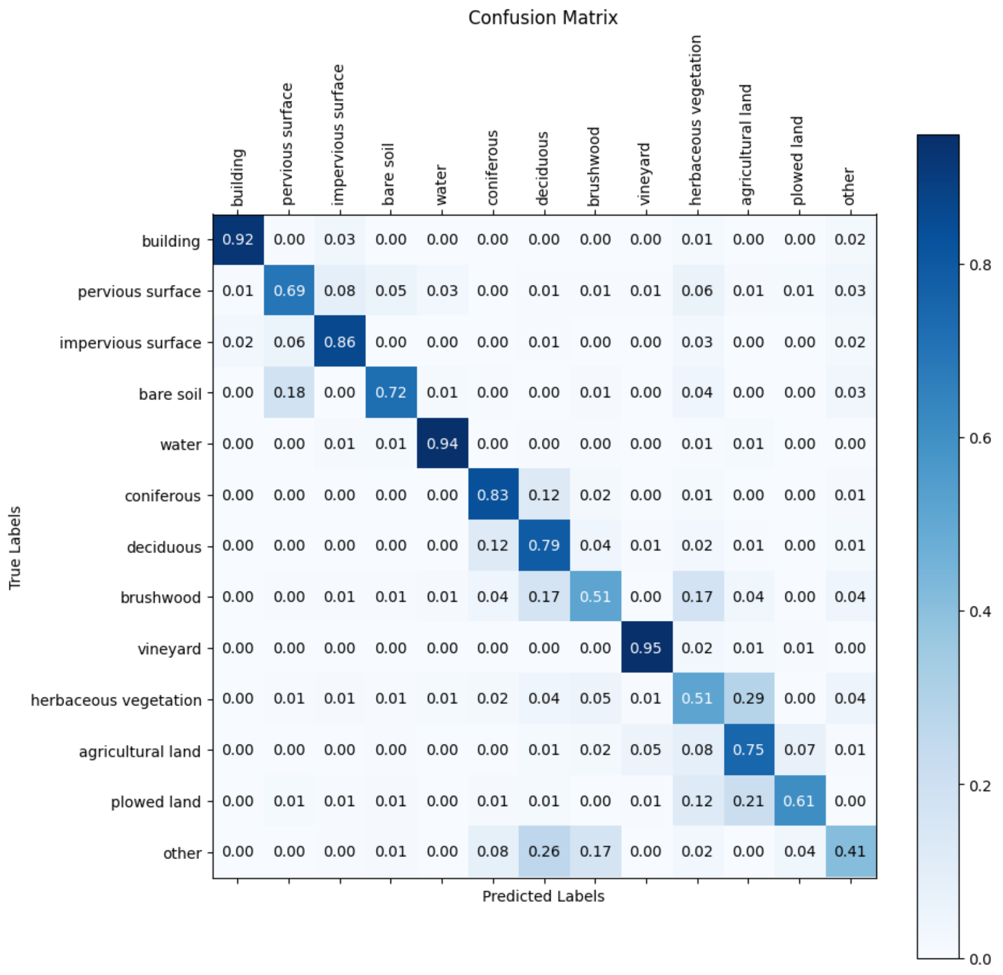
Figure 16: IoU per class for the best teacher student model together with the relevant baselines.

Finally, we again show confusion matrices for the 100 % models, but only for 5 % of the test set, due to computational limitations. The 13th row and column is denoted ”other” and represents the seven last classes grouped together. The confusion matrix for the RGB TS baseline can be seen in figure 17, for the privileged TS baseline in figure 18, and finally for the LUPI model in figure 19. There are no notable differences in the confusion matrices between the three models.



	building	pervious surface	impervious surface	bare soil	water	coniferous	deciduous	brushwood	vineyard	herbaceous vegetation	agricultural land	plowed land	other
building	0.90	0.01	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.03
pervious surface	0.01	0.72	0.05	0.03	0.01	0.00	0.01	0.03	0.00	0.09	0.01	0.01	0.03
impervious surface	0.04	0.07	0.81	0.00	0.00	0.00	0.01	0.00	0.00	0.03	0.00	0.00	0.03
bare soil	0.00	0.11	0.00	0.66	0.16	0.01	0.01	0.01	0.00	0.02	0.00	0.00	0.02
water	0.01	0.00	0.00	0.00	0.95	0.00	0.01	0.01	0.00	0.02	0.00	0.00	0.00
coniferous	0.00	0.00	0.00	0.00	0.00	0.67	0.28	0.02	0.00	0.01	0.00	0.00	0.00
deciduous	0.00	0.00	0.00	0.00	0.00	0.05	0.84	0.05	0.01	0.03	0.01	0.00	0.01
brushwood	0.00	0.00	0.01	0.03	0.01	0.01	0.19	0.46	0.00	0.24	0.02	0.00	0.04
vineyard	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.85	0.03	0.05	0.05	0.00
herbaceous vegetation	0.00	0.01	0.01	0.01	0.00	0.01	0.05	0.06	0.01	0.55	0.26	0.01	0.02
agricultural land	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.05	0.13	0.73	0.07	0.00
plowed land	0.00	0.01	0.00	0.01	0.00	0.00	0.03	0.00	0.00	0.05	0.24	0.65	0.00
other	0.00	0.01	0.00	0.01	0.00	0.21	0.06	0.42	0.00	0.06	0.02	0.00	0.21

Figure 17: Confusion matrix for the RGB TS baseline, trained on 100 % of the data.



	building	pervious surface	impervious surface	bare soil	water	coniferous	deciduous	brushwood	vineyard	herbaceous vegetation	agricultural land	plowed land	other
building	0.92	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.02
pervious surface	0.01	0.69	0.08	0.05	0.03	0.00	0.01	0.01	0.01	0.06	0.01	0.01	0.03
impervious surface	0.02	0.06	0.86	0.00	0.00	0.00	0.01	0.00	0.00	0.03	0.00	0.00	0.02
bare soil	0.00	0.18	0.00	0.72	0.01	0.00	0.00	0.01	0.00	0.04	0.00	0.00	0.03
water	0.00	0.00	0.01	0.01	0.94	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00
coniferous	0.00	0.00	0.00	0.00	0.00	0.83	0.12	0.02	0.00	0.01	0.00	0.00	0.01
deciduous	0.00	0.00	0.00	0.00	0.00	0.12	0.79	0.04	0.01	0.02	0.01	0.00	0.01
brushwood	0.00	0.00	0.01	0.01	0.01	0.04	0.17	0.51	0.00	0.17	0.04	0.00	0.04
vineyard	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.02	0.01	0.01	0.00
herbaceous vegetation	0.00	0.01	0.01	0.01	0.01	0.02	0.04	0.05	0.01	0.51	0.29	0.00	0.04
agricultural land	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.05	0.08	0.75	0.07	0.01
plowed land	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.12	0.21	0.61	0.00
other	0.00	0.00	0.00	0.01	0.00	0.08	0.26	0.17	0.00	0.02	0.00	0.04	0.41

Figure 18: Confusion matrix for the privileged TS baseline, trained on 100 % of the data.

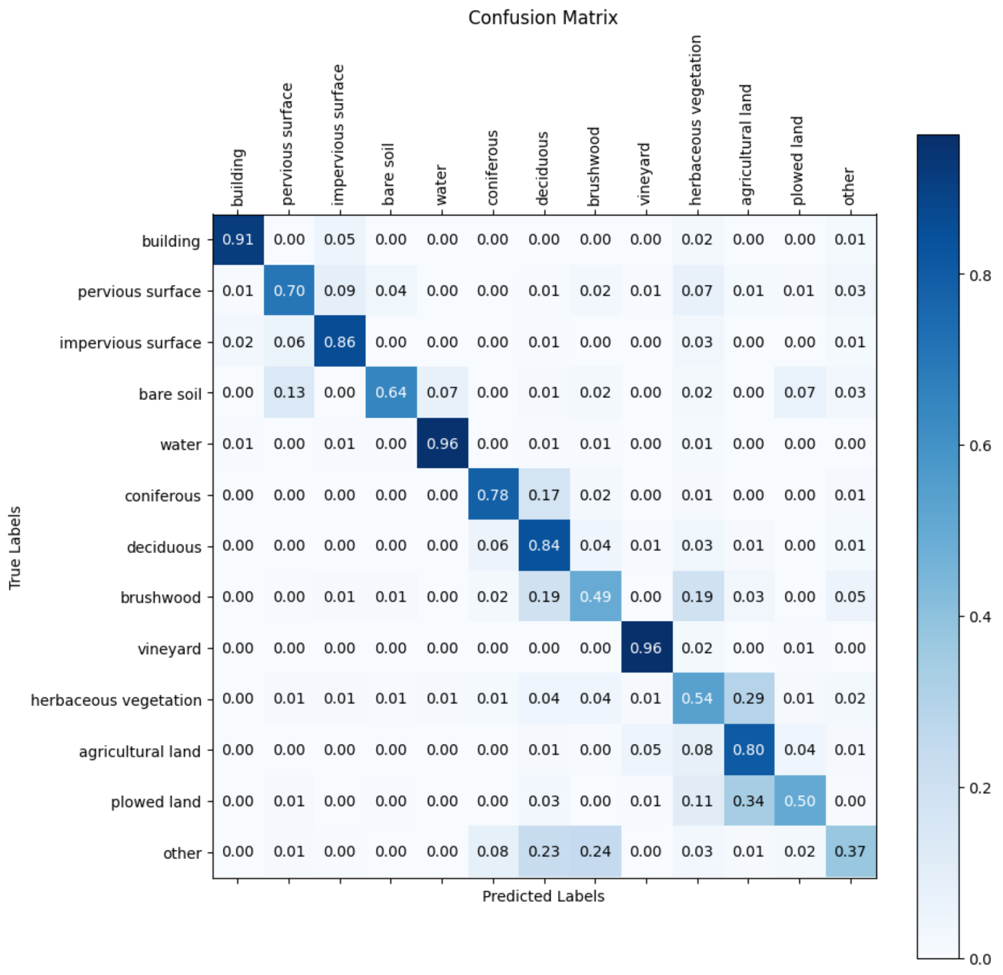


Figure 19: Confusion matrix for the LUPI teacher student model trained on 100 % of the data.

4.9 Predicting and Generating Privileged Information

The third and last category of LUPI architectures aimed to predict or generate the privileged information. As in the previous section, our experiments were conducted in a few rounds. The first round determined which of the different architectures to use and the second round determined which loss function to use for the prediction/generation of the privileged information. During the third round, we set some hyperparameters, and the fourth round consisted of training the most interesting models on 100% of the data.

4.9.1 Validation Set Results: Choosing Architecture

In table 18 below we present the results for the predicting/generating architectures. This experiment was run to determine which of the three architecture to focus on. All trainings are run on 5% of the data and with MSE loss between $\hat{y}_{\text{priv}}$ and y_{priv}. The results are compared with the RGB baseline (not the RGB TS baseline). The results show that at least the multi decoder and generating privileged information in two steps outperform the baseline, where the multi decoder performs the best. Therefore, we will continue this set of experiments focusing on the multi decoder architecture.

Architecture	RGB Baseline	Multi Decoder	Gen. in Two Steps	Gen. in One Step
Epoch	23	188	217	205
Loss	0.172	**0.168**	0.174	0.174
mIoU	0.580	**0.605**	**0.586**	**0.585**
Precision	0.723	**0.759**	**0.738**	**0.739**
Recall	0.723	**0.733**	**0.726**	0.714

Table 18: Validation set results for the predicting/generating architectures, all with MSE loss. The results which outperform the RGB baseline model (first column) are marked in bold.

4.9.2 Validation Set Results: Choosing Loss Function

After evaluating the different architectures, it was necessary to also settle for a loss function between the predicted privileged information $\hat{y}_{\text{priv}}$ and the true privileged information y_{priv}. In the above table, all models were trained using MSE loss, but in table 19 we also evaluate training the model using L1 loss and KL loss. All results are for 5% of the dataset, using the multi decoder. To clarify, the loss between the land cover classification $\hat{y}$ and the ground truth y is, as usual, the Dice loss. As seen in table 19 , it turns out that the MSE loss generates the best results.

Loss Function	MSE	L1	KL
Epoch	188	256	234
Loss	**0.168**	0.171	0.172
mIoU	**0.605**	0.589	0.563
Precision	**0.759**	0.732	0.730
Recall	**0.733**	0.729	0.696

Table 19: Validation set results for the multi decoder architecture using three different loss functions to compare $\hat{y}_{priv}$ and y_{priv}. The best results are marked in bold.

4.9.3 Validation Set Results: 100 %

Having found the multi decoder model using MSE loss to be our best model, we trained it on 100% of the data. The validation set results, compared to the RGB baseline trained on 100% are presented in table 20 below. As we can see, the results do not hold when we utilise the full dataset.

Architecture	RGB Baseline	Mutli Decoder
Loss Function	N/A	MSE
Loss	0.156	**0.154**
mIoU	0.639	0.633
Precision	0.784	0.777
Recall	0.762	0.756

Table 20: Validation set results for the the multi decoder model, compared to the RGB baseline model, both trained on 100 %. The loss function refers to one that compares $\hat{y}_{priv}$ and y_{priv}. Results which outperform the baseline score are marked in bold.

4.9.4 Test Set Results

Even though the results did not hold for the 100 % multi decoder architecture, we chose to still evaluate it on the test set. In table 21, we present the test set results for both the 5% and the 100% model. Unfortunately, neither multi decoder model beats the RGB baseline, in terms of mIoU.

Architecture	RGB Baseline	Multi Decoder	RGB baseline	Multi Decoder
Loss Function	N/A	MSE	N/A	MSE
Dataset Size	100 %	100 %	5 %	5 %
Loss	0.147	0.147	0.166	0.167
mIoU	0.579	0.574	0.543	0.532
Precision	0.712	**0.713**	0.679	**0.680**
Recall	0.735	0.725	0.697	0.685

Table 21: Test set results for the multi decoder models trained on 5% and 100% of the data respectively. The loss function refers to one that compares $\hat{y}_{priv}$ and y_{priv}. Wherever the models beat the baselines the results are marked in bold.

For the 100 % multi decoder model, table 20 shows the class-wise IoU compared to the RGB baseline. The multi decoder model outperforms the baseline on some of the classes, but on others. the baseline is the better one. In general, the differences are however quite small for all classes.

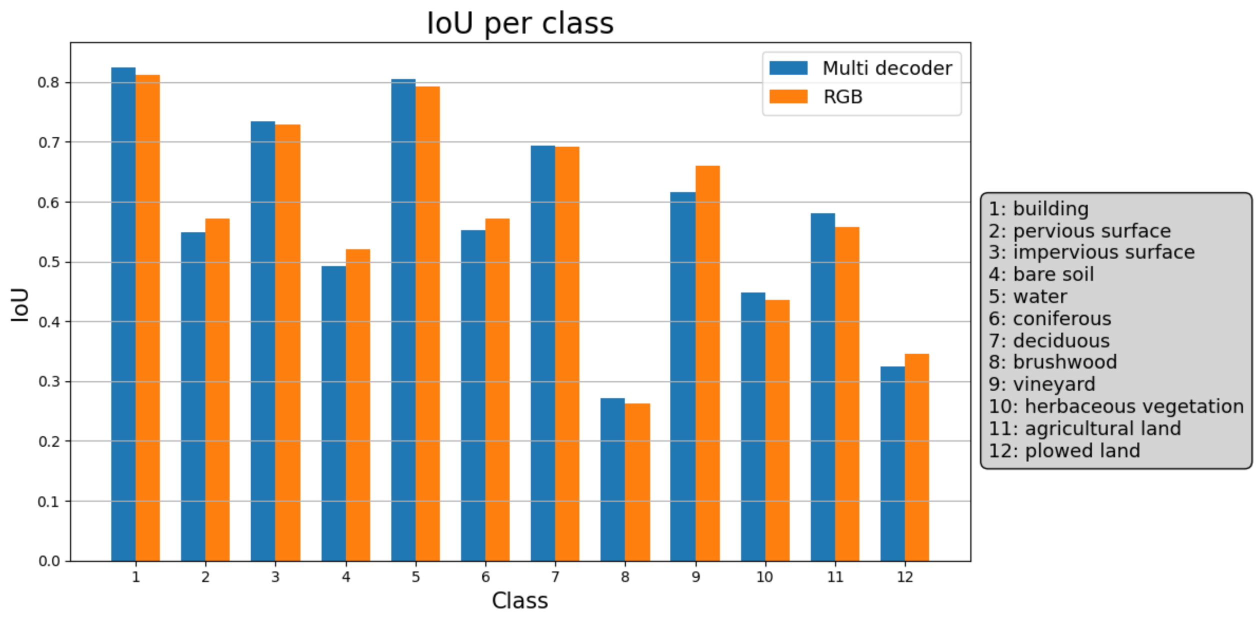


Figure 20: IoU per class on the test set for the multi decoder model trained on 100 %.

Finally, we again show confusion matrix for the 100 % model, but only for 5 % of the test set, due to computational limitations. The 13th row and column is denoted ”other” and represents the last seven, tiny, classes grouped together. The confusion matrix can be seen in figure 21.

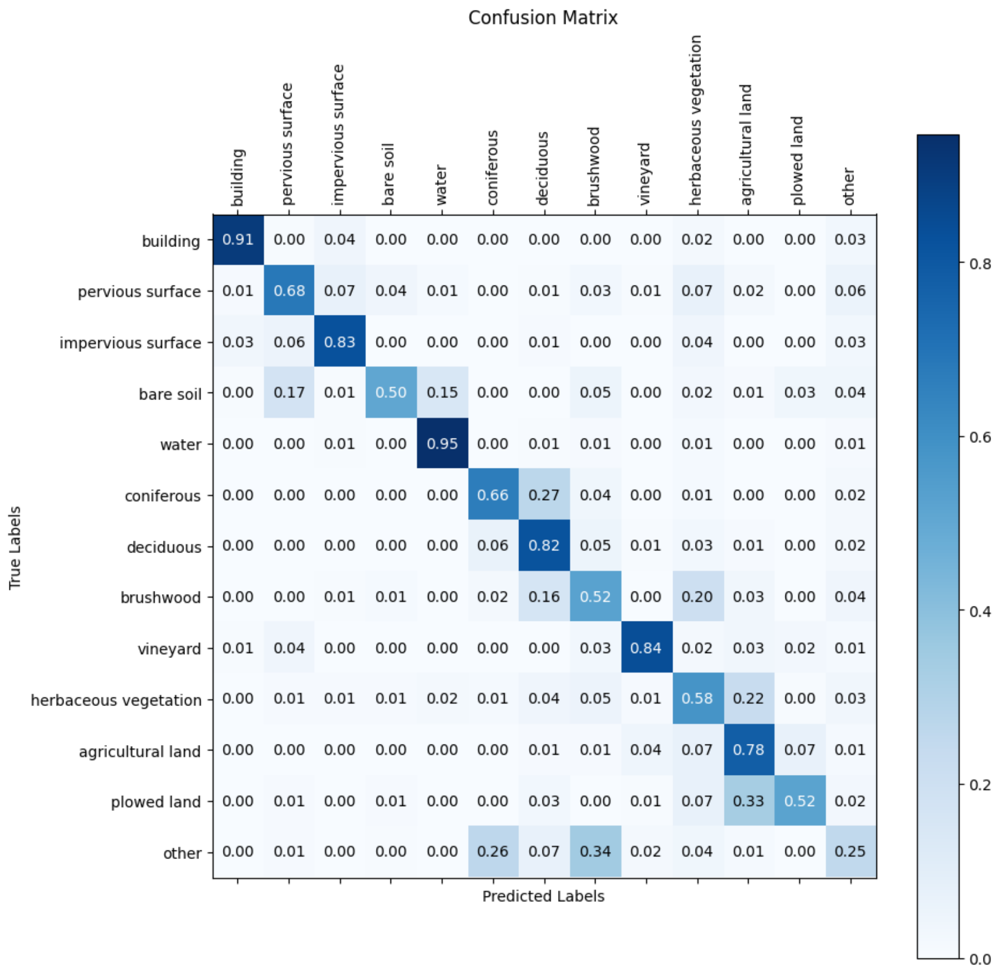


Figure 21: Confusion matrix for multi decoder model trained on 100 % of the data.

4.10 Focusing on NIR Classes

Judging from the results so far, there are no massive improvements for any classes when using LUPI. Rather, there is a small increase in performance for most classes. However, there are a few classes, primarily within vegetation classes, which are improved slightly more by the privileged information. This conclusion is based on initial studies of the models trained with RGB+NIR compared to RGB. NIR is known to be useful for detecting plants, which might explain this. If we only look at the performance for these five classes, 6 (coniferous), 7 (deciduous), 8 (brushwood), 9 (vineyard) and 12 (plowed land), we get a greater improvement from using LUPI. The test set result for these classes can be seen in table 22 for the 5 % models, and 23 for the 100 % models. The networks used are the U-Net TS baselines, and the U-Net LUPI TS model. Note that these results do not come from new trainings, but simply from calculating the metrics for only the five classes for the already existing models. As we can see, there is now a greater gap between the RGB TS baseline and the LUPI model, especially for the 100 % model where the LUPI model even outperforms the privileged TS baseline for most metrics.

Model	RGB TS Baseline	Priv TS Baseline	LUPI TS
mIoU	0.469	0.506	**0.487**
Precision	0.601	0.616	0.597
Recall	0.666	0.716	**0.693**

Table 22: Results for the five classes where the privileged information is the most helpful, for the 5 % U-Net teacher student model. Results are marked in bold where the LUPI model beats the RGB TS baseline.

Model	RGB TS Baseline	Priv TS Baseline	LUPI TS
mIoU	0.514	0.537	**0.539**
Precision	0.632	0.637	**0.656**
Recall	0.707	0.744	**0.723**

Table 23: Results for the five classes where the privileged information is the most helpful, for the 100 % U-Net teacher student model. Results are marked in bold where the LUPI model beats the RGB TS baseline.

4.11 Ensemble Model

The results seen in the teacher student architectures indicate that there is an ensemble effect. Also, as any reader who made it all the way here has surely noticed that we have trained many different models. At this point it then seemed to make sense to pick some of these models and let them all conduct the land cover classification and then take a mean of their prediction. Furthermore, some of the very overly confident models got their predictions smoothed a bit. It should also be noted that this set of experiments is not primarily ran to show the strength of LUPI or privileged information. That is to say, we have in the scope of this thesis not trained as many or diverse RGB models as we have trained LUPI models or privileged models. So, in some sense this is a slightly unfair competition for the RGB models as we have not trained some models which very well could be trained as RGB models. One example of this are the metadata and sentinel models. However, it is clearly so that the set of privileged models contains all LUPI and RGB models, while the set of LUPI models contains also RGB models. So it it not unjust that there are more privileged models than LUPI models, and more LUPI models than RGB models. To summarize, this is mostly an experiment to see how strong results we can achieve based on the models we already have trained. The specifics of which models were included can be seen in the appendix, in Section A.4.

In table 24 we present the test set results for the 5 % ensemble models, next to our previously best 5 % model, the privileged student trained with a privileged teacher. As seen in the table, all ensemble models outperforms the former best model. The privileged ensemble model outperforms the LUPI ensemble model, which in turn outperforms the RGB ensemble model. Furthermore, the privileged ensemble model outperforms the previously best mIoU results by almost 4 p.u., and hence performs almost as well as the best 100 % model.

Type	RGB Ensemble	LUPI Ensemble	Priv Ensemble	Priv TS Baseline
Loss	0.311	0.312	0.312	0.160
mIoU	**0.567**	**0.584**	**0.597**	0.558
Precision	**0.702**	**0.717**	**0.724**	0.687
Recall	0.719	**0.733**	**0.749**	0.725

Table 24: Test set results for the three 5 % ensemble models, next to the previously best 5 % model. Where the ensemble models beats the former best results, seen in column five, the values are marked in bold.

In table 25 we present the test set results for the 100 % ensemble models, next to our previously best 100 % model, the privileged student trained with a privileged teacher. As seen in the table, the LUPI and priv ensemble both outperforms the previously best model. The results for the privileged model are improved by 2 p.u., which would give us an honorable 8th place in the competition released in parallel with the release of the dataset, 1.5 p.u. behind the winner.

Type	RGB Ensemble	LUPI Ensemble	Priv Ensemble	Priv TS Baseline
Loss	0.310	0.310	0.310	0.140
mIoU	0.591	**0.613**	**0.627**	0.607
Precision	0.720	**0.736**	**0.747**	0.728
Recall	0.745	0.763	**0.773**	0.764

Table 25: Test set results for the three 100 % ensemble models, next to the previously best 100 % model. Where the ensemble models beats the former best results, seen in column five, the values are marked in bold.

4.12 Summary of Results

The results produced so far have been extensive, but far from all have been promising. For clarity, we will here provide a short summary of the results, and show the most notable results on the test set. In short, neither the noise architectures or the predicting/generating architectures performed well enough to actually beat the baseline properly. Therefore, the only interesting results come from the teacher student architectures, and those are the only results that will be shown here. These are fascinating for two reasons, firstly because there is an ensemble effect. That is to say, even the baselines are significantly improved by getting trained by a teacher with the same knowledge as itself. Secondly, and most importantly, the teacher student architectures provide meaningful knowledge transfer, as the LUPI model outperforms the baseline. In the following two sections we will provide these results for the 5 % models and the 100 % models respectively.

4.12.1 Test Set Results: 5 %

The best results from our models trained on 5 % of the data, was as mentioned, a teacher student model. Within the LUPI model, the teacher had RGB, NIR and elevation as input, but the student only had RGB. As loss function between the teacher and student predictions, KL divergence was used. We also used two temperature constants, $T_S = 2$ for the student, and $T_T = 7$ for the teacher. This is presented in table 26 together with the teacher student baselines. The RGB TS baseline is a student with RGB input trained by a teacher with RGB input, and the privileged TS baseline is a privileged student, trained by a privileged teacher. The privileged TS baseline can be considered as an upper bound for the LUPI model. Finally, the last column shows the test results on the normal RGB model, which was the initial baseline, to illustrate the ensemble effect of the teacher student architecture. As we can see from the table below, the mIoU of the LUPI model is almost as good as that of the privileged TS baseline, and clearly outperforms the RGB TS baseline. This suggests that the teacher student architecture has successfully made use of the privileged information.

	Baseline	**TS Baselines**		**LUPI**
Teacher Input	N/A	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	N/A	RGB	RGB, NIR, elevation	RGB
Loss	0.166	0.166	0.160	0.174
mIoU	0.543	0.545	0.558	**0.554**
Precision	0.679	0.687	0.687	0.685
Recall	0.697	0.707	0.725	**0.718**

Table 26: The best teacher student model on 5 %. For comparison, the TS baselines and the RGB model is included. Where the LUPI model beats the baseline RGB TS baseline, the score is marked in bold.

4.12.2 Test Set Results: 100 %

The model that performed best on 100 % of the data was also a teacher student model. Table 27 shows the results for this model evaluated on the test set. Just like for 5 %, we include the TS baselines. As above, we also include the RGB baseline, the one without a teacher, to more clearly see the ensemble model effect. Similarly to the 5 % models, the LUPI model clearly outperforms the RGB TS baseline, and almost hits the scores of the privileged TS baseline, which is to be considered an upper bound. Notably, the difference is even greater between the 100 % models than the 5 % models.

	Baseline	**TS Baselines**		**LUPI**
Teacher Input	N/A	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	N/A	RGB	RGB, NIR, elevation	RGB
Loss	0.147	0.148	0.140	**0.143**
mIoU	0.579	0.588	0.607	**0.606**
Precision	0.712	0.717	0.728	**0.736**
Recall	0.735	0.744	0.764	**0.753**

Table 27: The best teacher student model on 100 %. For comparison, the TS baselines and the RGB model is included. Where the LUPI model beats the baseline RGB TS baseline, the score is marked in bold.

Our best LUPI model reaches a mIoU of 0.606, compared to the teacher student baseline with a mIoU of 0.588. Even though this difference might be too small for a human eye to notice when comparing predictions, we want to show two example images where the LUPI model outperforms the baseline. Figure 22 shows a clear example of this. The LUPI model correctly labels the majority of the pixels as deciduous (light green), whereas the RGB model fails to do so.

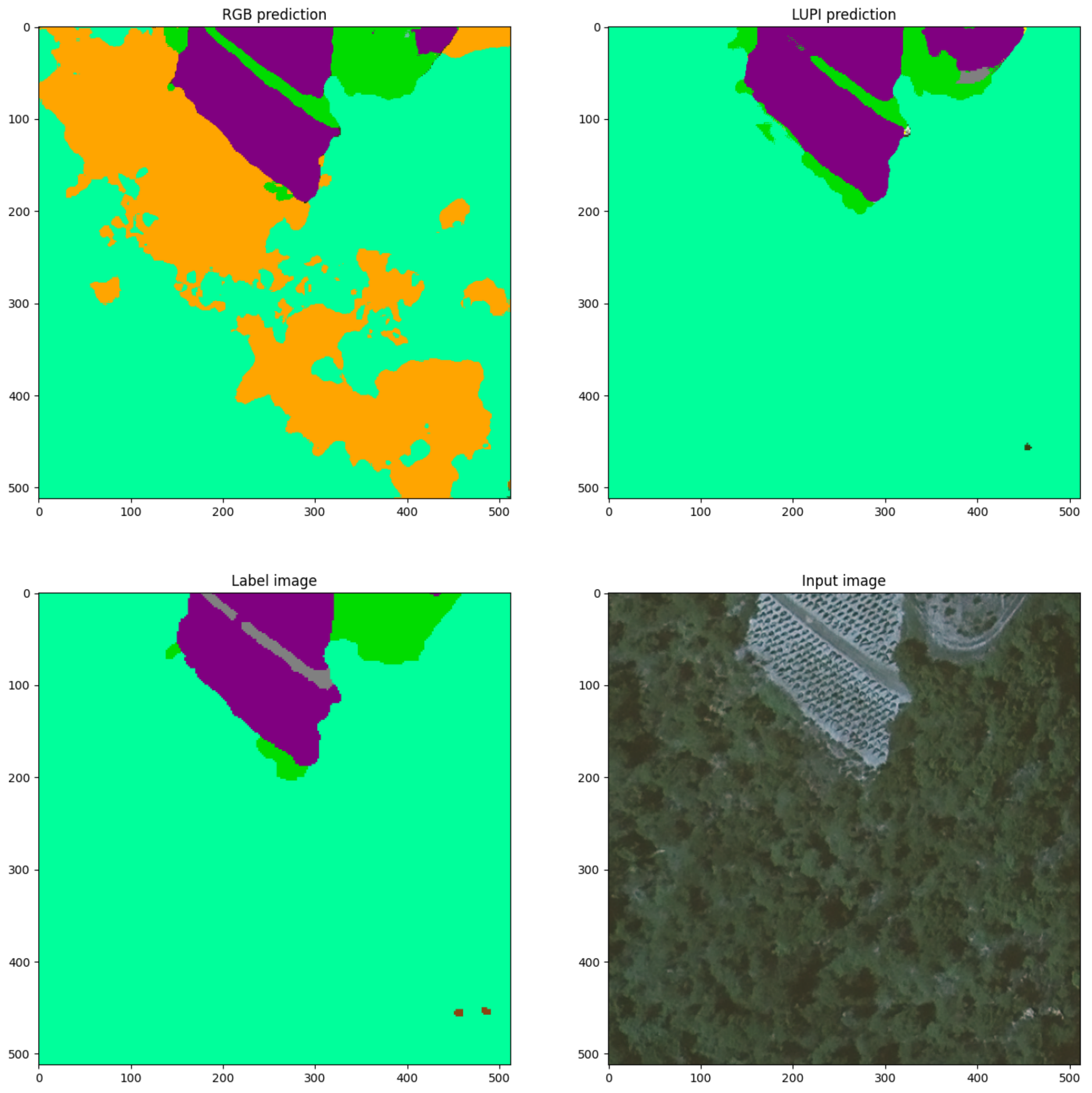


Figure 22: Example image of input, label and prediction using both the baseline model and the best LUPI model.

In figure 23 below, we can see a second example, in which the LUPI model correctly predicts the upper right part of the image to be herbaceous vegetation, while the RGB model predicts it to be agricultural land.

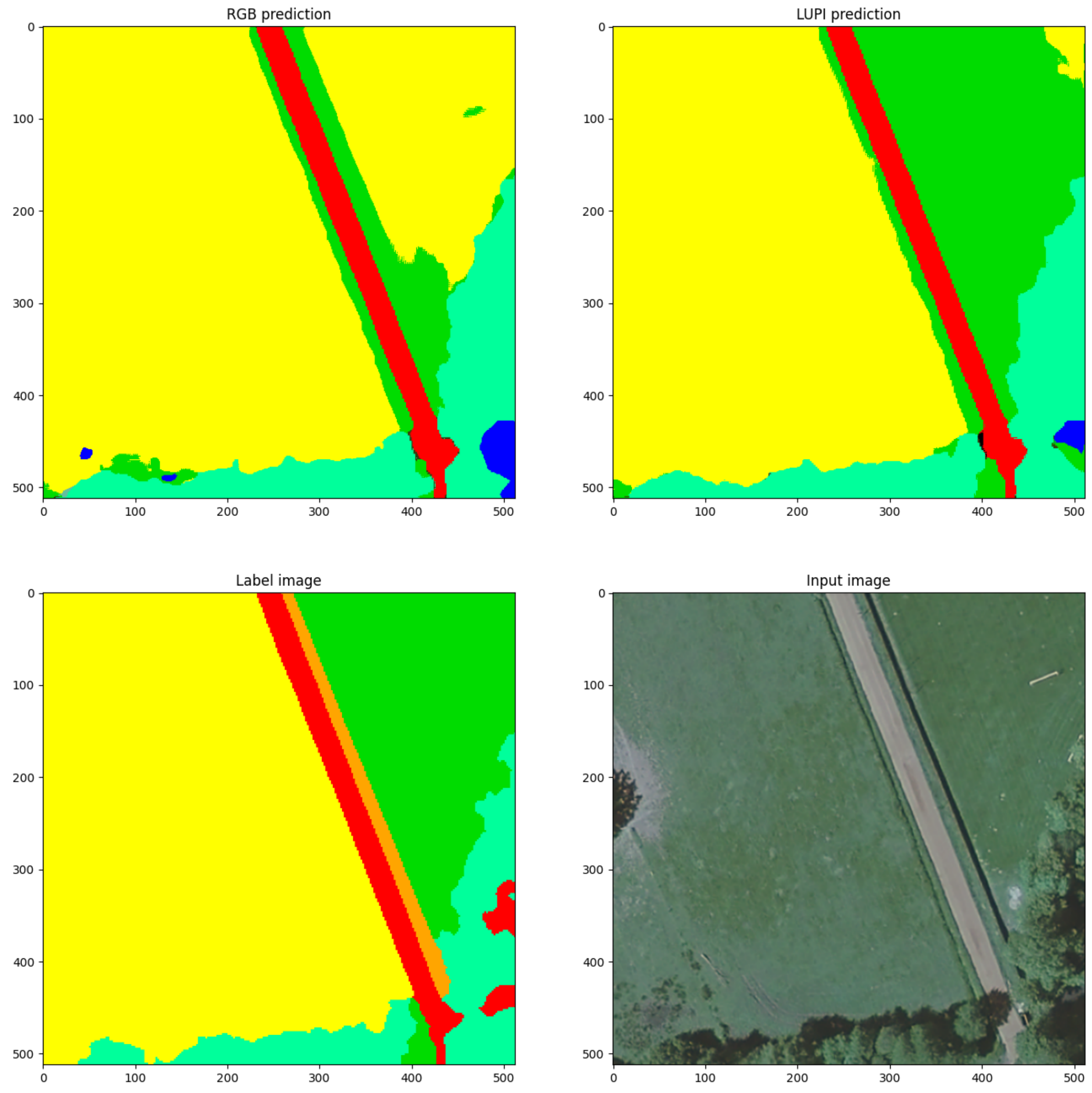


Figure 23: Example image of input, label and prediction using both the baseline model and the best LUPI model.

5 Discussion

In this section, we evaluate whether our objectives have been achieved, analyze the results, and consider future research directions. We will start by presenting general findings and then delve into architecture-specific discussions, focusing on the most exciting architecture; the teacher student models.

5.1 The Objectives

First of all, we succeeded to complete all our main objectives. We started by concluding that there was relevant information available in the privileged channels, as networks with access to NIR and elevation outperformed those trained and evaluated without. By doing that, we achieved the sub target of studying if privileged information was actually helpful, which it was deemed to be. However, the gap between the models with and without privileged information was not too big, which we will get back to shortly. Onwards, we studied noise architectures, teacher student architectures and predicting/generating privileged information architectures in the context of privileged learning for land cover classification. This was out main goal, during which we were even able to successfully identify a setup of the teacher student architecture which could transfer privileged information.

Our third objective regarded whether NIR or elevation was the most useful type of privileged information. From our studies, it was clear that NIR was more useful than elevation, even though we ended up concluding that both might still contain relevant features. However, as mentioned, the elevation data available in FLAIR 2 was constructed from the NIR and RGB information, why it is no surprise that the elevation information was less useful on its own. In a different dataset, with "real" elevation data, this conclusion might not hold. It should also be noted that which type of privileged information is useful, probably is heavily dependent on which classes are used in the land cover classification. That is to say, if all classes are "flat", e.g. different types of soil, elevation maps will be of no use. On the other hand, if "mountain" is a class elevation maps will likely be of more value. Likewise, NIR is known to be useful for vegetation classes, which also is suggested by our results in section 4.3.

Our last objective was to understand the potential in the available metadata and satellite imaging. The conclusion from those experiments was that the metadata could be useful on the overall performance, especially for 5% of the data. Also, we noticed that the metadata (which included date stamps) improved the IoU on the two classes that have a strong temporal dependency, namely agricultural and plowed land. In the end, the metadata was omitted because of simplicity reasons and the extent to which it improved performance. Another thing that is worth mentioning is that metadata and satellite images are often available during training and testing. In the case of land cover classification on aerial images, it would be a stretch to consider satellite images, with a lower resolution, privileged information. Even if the results had been better, it would arguably have been a bit out of scope for the main objective, which is about privileged information. The same is true for metadata such as latitude, longitude, time stamps and data stamps.

To summarize, we were able to successfully complete all our main objectives. In the following sections we will discuss the models's performances, analyze interesting results and behaviours and reason about future studies and fields of improvements.

5.2 Noise Architectures

In short, the noise architectures did not yield particularly compelling results. The only exception was the zero out model, as it demonstrated a near-successful application in our experiments. Nonetheless, it was not of primary interest, as we can see when comparing to the teacher student results and predicting/generating models. One thing that the noise models proved to be effective at was preventing overfitting. This outcome is not unexpected, as the application of noise can be likened to extreme data augmentation, while zero out may be viewed as a form of intense regularization similar to dropout techniques. Thus, there could be potential for further exploration within this topic. However, due to limited time and the disappointing results observed, we decided to focus on other studies. It should also be noted that one upside of these types of architectures is that it is easily implemented and does not require much to translate between different types of neural networks.

5.3 Teacher Student Architectures

The teacher student architectures gave many promising and interesting results. First of all, the architecture was successfully able to transfer privileged information from the teacher to the unprivileged student. Two other main observations are that

1. The student can learn a lot from the teacher even though the teacher and the student actually have access to the exact same input.
2. Some of the student's losses remained high, while still generating very good results.

If we start by thinking about the first statement: how come the model improves so much from getting pointers from a teacher with access to no more information than the student itself? This is not an easy question to answer, but there are a few possible reasons. As briefly mentioned, the teacher student setup resembles an ensemble model, as we get input from two different sources. In an ensemble model, two models with roughly the same score on some test can perform better together, because they are good at different things. Very much like humans, models become better when they collaborate. Also, this nudge from a peer or teacher, might decrease the risk of getting stuck in some local minima, and work like some sort of regularization. This also promoted the idea of looking into normal ensemble models. Furthermore, a known problem with many loss functions based on "hard" binary labels, i.e. comparing probabilities to either 0s (if the prediction was incorrect) or 1s (if the prediction was correct), suffer from the fact that the loss will decrease when the model increases its confidence, without improving its accuracy. Obviously, we are not that interested in rewarding our model for being confident, but instead for increased accuracy. How does this then relate to the teacher student experiments? When we introduce the loss between the teacher's prediction and the student's prediction, we introduce a part of the loss which gets less rewarded for being overly confident, as it is compared to the teacher's predictions, which will not be as "hard" as the ground truth, especially if we use a temperature T_T. As it happens, this might also answer why the teacher student models sometimes had a higher loss, while still rendering better results than the RGB baseline. That is to say, the loss might be high because the teacher student models were not as confident. To read more about the confidence of our models, see section A.2 in the appendix.

These observations give clues on how we can further improve both our baselines and LUPI models. One could, for example, alter the loss function to not reward overconfidence. It could also be interesting to look at a student with multiple teachers, even with the same input but different initiation, but also teachers trained on different information. Unfortunately, we will not have time to look at multiple teachers within the scope of this thesis.

Regarding the more technical details of the teacher student architecture it should be noted that this setup is very easy to implement for all different types of networks. The only tricky part is the loss function, but it can be built for any type of neural net. This also implies that the teacher and student do not even need to be of the same network type. It even implies that it is easily translated into other tasks than semantic segmentation, as all that is needed is to update the type of loss used between the teacher prediction and student prediction. One negative aspect of the teacher student architecture is that it requires the training a teacher first. To clarify, the number of necessary trainings needed to get a LUPI model, an RGB baseline and a privileged baseline, increases from three to five. This is a disadvantage as computational resources are in general scarce.

5.4 Predicting and Generating Privileged Information

The last type of architecture we studied was predicting and generating privileged information models. In short, these initially showed some promise, but the results did not hold, neither to test nor the full dataset. During these runs, we saved samples of the predicted/generated privileged information. Throughout, it was clear that the predicted/generated information was always blurry, without sharp edges, in comparison with the real privileged data. An example of this can be seen below in figure 24, where the prediction of a NIR image is shown together with the true image. The prediction captures all main traits of the true image, but have the said unsharp edged. We did not succeed in resolving this, even when trying losses that would penalize blurry edges more. Furthermore, as the results remained poor, it seemed like the architecture was of no use. Again, if the available privileged information had been more useful, this architecture would might have proved more helpful, as the reconstruction of the privileged information worked fairly well. Also, we hoped that our models would reproduce privileged information that was even better than the real one, but the results from these experiments did not show such promise. As always, however, it is not impossible that it could have been done with more thorough investigation. It should also be noted that all predicting and generating architectures are custom made for U-Net, and while the generating architectures are slightly easier to generalize than the predicting architecture, there is still no clear way of exactly how to transfer these setups to other network types. This is a disadvantage, as U-Net is mostly suitable for semantic segmentation and since the reconstruction of the networks is more complicated than changing the loss function.

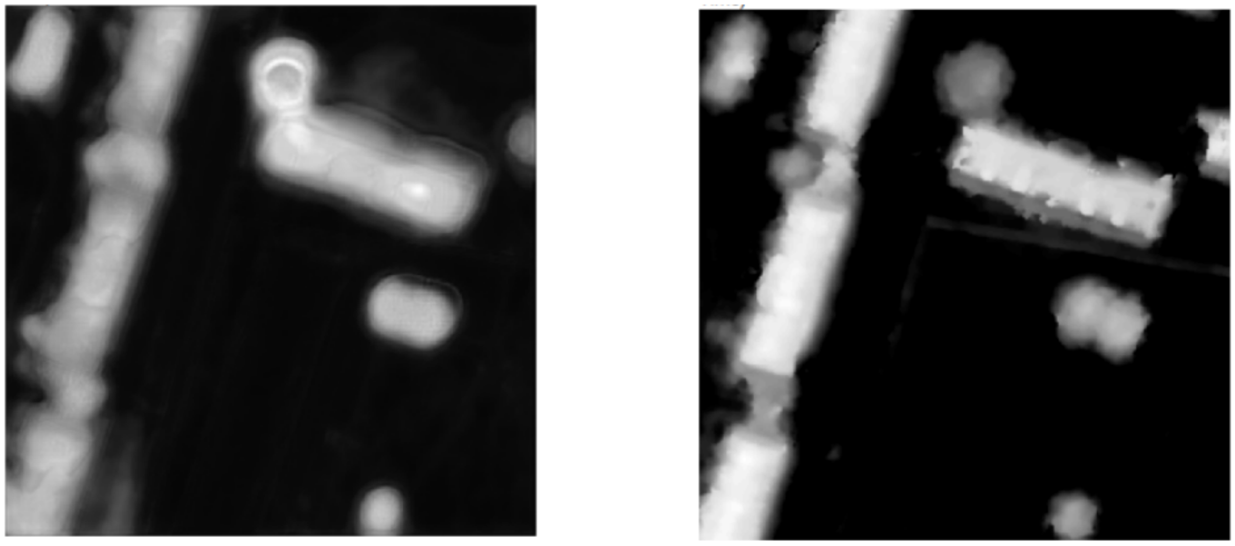

(a) Prediction of a NIR image. (b) Corresponding true NIR image.

Figure 24: Example of predicted privileged information compared to true privileged information.

5.5 General Observations

We were successfully able to transfer the privileged information available during training to improve performance during inference, primarily through the teacher student architecture. It should also be noted that our results are comparable with the performance of the state of the art models for this dataset, and clearly outperforms the baselines presented with the release of the FLAIR dataset [6]. Our thesis was not primarily about obtaining the best results on this dataset, but the fact that the achieved results are in the ballpark of the best, makes it more likely that our proposed methods actually hold for real world applications.

In terms of dataset size, we have seen that privileged information (both directly and in LUPI) seems equally useful when the network has access to 5% and 100% of the data, especially when focusing on the test set results. This is surprising, since when data is limited the marginal effect of more data should be greater than in the extreme case of having unlimited training data. Furthermore, it should also be noted that we have mostly optimized for 5% of the dataset. All models were first trained and fine-tuned on 5%, and then the most promising architectures and configurations were attempted on 100%. On the other hand, the results have generalized fairly well, and training on 100% takes such a long time that we due to time consumption would not have been able to fine-tune much if we had done it on the full dataset. Hence, it is not too unlikely that we achieved better results on the 100% models with this strategy than we would have by always training on 100% since we were able to attempt way more different things. Clearly, it is in itself a fascinating topic to understand if results on subsets of the data generalize well to the entire dataset as computational resources are commonly a limiting factor. We will not have time to study this further, but it is an interesting road for future research. It should also be noted that the results for the 5 % models compared to the results of the 100 % models differ in only a few percentage units. This is fascinating, as additional data is generally highly beneficial for a model. Unfortunately, we do not have time to consider this more closely.

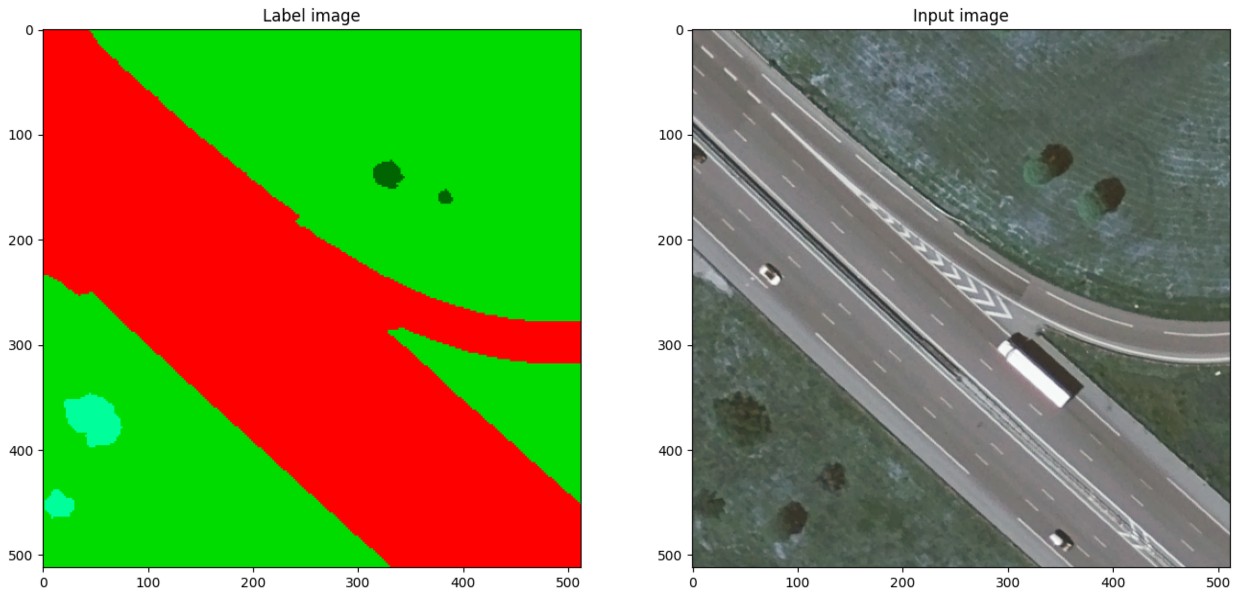
As seen throughout almost all experiments, the privileged channels in this dataset consistently help the model when available. While this improvement is significant and important, it is not enormous. To clarify, this is referring to gap between the baselines that have privileged information as input also during evaluation. Due to this, it is difficult to train a LUPI model which outperforms the non-privileged baseline since the gap is quite small. Imagining a situation where one has more relevant privileged information than we do now, it is not unreasonable to assume that all our proposed methods, and the teacher student architecture in particular, would be even more versatile.

Another thing which should be noted is that our dataset has a highly varying class frequency between the training, validation and test set. A common assumption for machine learning is that the training, validation and test set are drawn from the same distribution, which is clearly not the case in FLAIR 2. This is not something we have focused too much on, apart from using Dice loss which weights all classes equally regardless of their occurrence. Once again, we considered it to lie outside the scope of this thesis, even though it could be relevant future research. Also, this dataset is somewhat special since it contains seven tiny classes, which at the time of the release and the corresponding competition were considered irrelevant. This fact has not been a priority and apart for some experiments regarding how these affect the loss, the performance of these seven classes have mostly been ignored, while they still have affected the loss during training. Surely, there is a better way of handling these classes, especially since we do not care about the classification of them. However, since initial experiments of grouping them together and ignoring them in the loss did not improve any performance, this topic has not been prioritized. How to handle small classes, and/or the occurrence of irrelevant pixels could also be a relevant topic for future study.

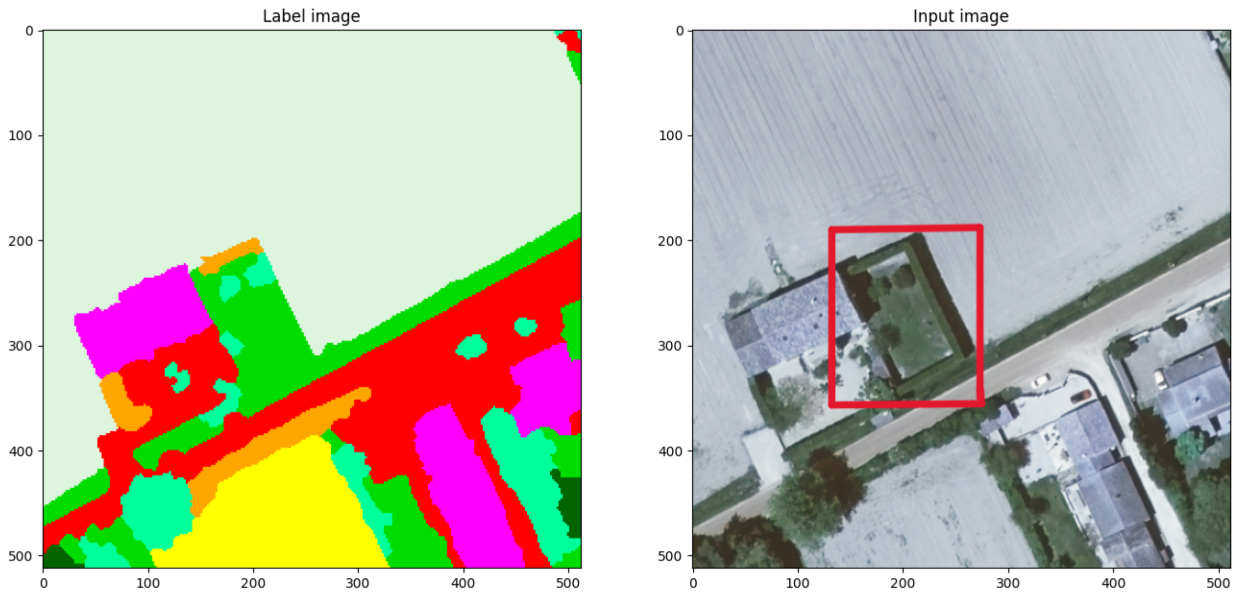
Finally, while training with Dice loss, we also logged the CE loss during validation. For most types of networks the CE loss actually increased, while the mIoU still increased. This fact is further discussed in Section A.3 in the appendix. We also noted that most of our models were very overconfident, meaning our models, in general, assigned a way higher probability to the class it guessed a pixel to be, than the probability it had of being right (the accuracy). This is was partly discussed in the above section and more in depth in Section A.2 on overconfidence, in the appendix.

5.6 Label Noise

When looking for illustrative examples of model outputs to include in our report, we realized that some of the labelling appeared inconsistent. Figure 25a shows an example of an input image with labels. Notice the four trees in the bottom left corner, and how two of them are labelled as deciduous (light green), and how the two other trees are labelled as herbaceous vegetation (green), the same class as the surrounding area. This labelling seems inconsistent. It is hard to understand what could possibly distinguish the two trees from the others. Notice also the two tress in the top right corner of the image. These two tree trees are labelled as coniferous (dark green), unlike the other four trees in the image. To the naked eye, these to trees do not look like they are coniferous, even though they might be, but it is once again hard to understand what motivated this labelling. Figure 25b shows another example of where, what seems like hedges, have been partly labelled as brushwood (orange), partly labelled as deciduous (turquoise), and partly labelled as herbaceous vegetation (green). The relevant area is marked in the image with a red rectangle. Once again, it is hard to say what the correct label is, but it should likely be one, and not three classes.



(a) Example one of inconsistent labeling.



(b) Example two of inconsistent labelling. The relevant area is marked with a red box.

Figure 25: Two examples of an input image with corresponding labels.

We have found other examples where a part of an object has been labelled as one class, and another part as a second class. All these inconsistencies likely lead to problems for the training of our model. If the labelling is not consistent, it is very probable that the model will struggle to correctly classify classes where this occurs. Labelling is a hard task, so it is natural that some errors occur in the dataset. To mitigate the negative implications, one could maybe bundle similar classes together to avoid the problem of having to distinguish between coniferous and deciduous trees, as an example. Without a specific task at hand, it is hard to say what the best solution would be. We can, nevertheless, conclude that the label noise probably makes our model worse.

6 Generalisation of Results

We have concluded that the teacher student architecture is an efficient way of incorporating privileged information into our model. One thing to consider, however, is that we have only shown this using a U-Net with EfficientNet-B4 as encoder. In order to conclude that these results hold in general, we want to try other networks too. Therefore, we will in this section reproduce the experiments on some architectures which are quite different from U-Net. The first one is FCN-8s, which is not an encoder-decoder net, but a CNN adapted for semantic segmentation. The second architecture is called UNetFormer, which is an encoder-decoder architecture but with a transformer-like decoder instead of a CNN decoder. We also evaluated the teacher student setup using ResNet50, which is a regular CNN. We have not performed any fine-tuning for the new nets. Furthermore, we have discussed that the usefulness of the currently available privileged information seems limited. Therefore, we also want to evaluate the teacher student setup in a scenario with stronger privileged information.

6.1 FCN-8s

In this section we study the teacher student model with the FCN-8s network, on both 5% and 100% of the data. Table 28 below present the test set results for these attempts when training on 5 % of the dataset. In terms of loss function, FCN-8s was trained with CE instead of Dice, which in turn caused an update of the reweighting R to 100%. As mentioned before, the temperatures did not matter much for the non-LUPI models why all such temperatures have been set to 1. Apart from that, no updates have been made to suit the new network architecture. As we can see in table 28 , the privileged TS baseline performs better than the LUPI model, which in turn outperforms the RGB TS baseline in terms of mIoU. Thus, the teacher student architecture seems to work for FCN8-s when using 5 % of the data.

	TS Baselines		**LUPI**
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S, T_T}$	(1,1)	(1,1)	(2,7)
R	1/100000	1/100000	1/65000
Loss	0.203	0.186	0.221
mIoU	0.405	0.452	**0.417**
Precision	0.578	0.595	**0.595**
Recall	0.572	0.630	**0.575**

Table 28: Test set results for FCN-8s trained on 5 % of data. The LUPI model, seen in the last column, is marked in bold where it beats the RGB baseline.

The same experiments were done for 100 % of the dataset. These results are presented in table 29. We can see that the LUPI model did not outperform the RGB TS baseine.

	TS Baselines		LUPI
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/30000	1/30000	1/20000
Loss	0.194	0.183	0.239
mIoU	0.507	0.521	0.488
Precision	0.650	0.683	**0.661**
Recall	0.681	0.673	0.641

Table 29: Table over test results for FCN-8s trained on 100 % of data. The LUPI model, seen in the last column, is marked in bold where it beats the RGB baseline, shown in the first column.

Summarizing, we are able to improve results with the teacher student setup when training on 5 %, but not on 100 % of the data. It should also be noted that the general performance of this network is worse than that of U-Net. This seems reasonable as the FCN8-s net is not a state-of-the-art model. However, it is not unlikely that the general results could have been improved by fine-tuning. It is also possible that there exists some set of hyperparameters for the LUPI model that would outperform the RGB model on 100 % of the data.

6.2 UNetFormer

Another neural network that we evaluated was the UNetFormer architecture. In table 30, we present the test results for the models trained on 5 % of the dataset. UNetFormer was trained with CE instead of Dice, why R is adjusted to match the scale, and a small weight decay of 0.0001 is added to increase stability. Furthermore, it has a U-Net teacher instead of a UNetFormer teacher. The main reason for this is that we faced instabilities when training a UnetFormer with all five channels as input. Having a different teacher is also efficient as we then do not have to train another teacher network. Finally, it is also interesting to understand if any teacher helps, or if it is better with the same type of network as the student. Apart from that, no updates have been made to suit the new network architecture. In the table below, we see that the LUPI model outperforms the RGB TS baseline when training on 5 % of the dataset.

	TS Baselines		LUPI
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/30000	1/30000	1/15000
Loss	0.191	0.180	0.195
mIoU	0.478	0.506	**0.490**
Precision	0.632	0.641	**0.636**
Recall	0.635	0.686	**0.659**

Table 30: Test set results for UNetFormer trained on 5 % of data. The LUPI model is marked in bold where it beats the RGB TS baseline, shown in the first column.

We continued with the models trained on 100 % of the data. These results are shown in table 31. Again, we see that the LUPI model outperforms the RGB TS baseline.

	TS Baselines		LUPI
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/30000	1/30000	1/15000
Loss	0.169	0.159	**0.157**
mIoU	0.555	0.575	**0.568**
Precision	0.695	0.709	**0.700**
Recall	0.712	0.726	**0.729**

Table 31: Test set results for UNetFormer trained on 100 % of the data. The LUPI model is marked in bold where it beats the RGB baseline, shown in the first column.

In general, we note that the metrics are not as good as when using U-Net as neural network. More importantly however, the results seem to generalize well to using UNetFormer in terms of transferring the privileged information to the student. We see that the student can learn when the teacher is of a different network type.

6.3 ResNet50

The final network we used to asses our methods was ResNet50, the same that was used in the early experiments. ResNet50 was trained using CE loss, which motivated an update of R. Apart from that, no changes were made compared to the U-Net runs. In table 32 we can see the 5 % results, where the LUPI model clearly outperforms the RGB TS baseline.

	TS Baselines		LUPI
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/30000	1/30000	1/20000
Loss	0.177	0.166	0.202
mIoU	0.496	0.531	**0.515**
Precision	0.657	0.667	**0.661**
Recall	0.646	0.671	**0.695**

Table 32: Test set results for ResNet50 trained on 5 % of data. The LUPI model is marked in bold where it beats the RGB baseline, shown in the first column.

In table 33, we see the results for the model trained on 100 % of the data. Note that the privileged TS baseline actually has a lower mIoU than the RGB TS baseline, unlike the case for 5 %. This result shows that ResNet50 cannot make use of the privileged information at all in our teacher student architecture when using the entire dataset.

	TS Baselines		LUPI
Teacher Input	RGB	RGB, NIR, elevation	RGB, NIR, elevation
Student Input	RGB	RGB, NIR, elevation	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/30000	1/30000	1/20000
Loss	0.174	0.164	0.207
mIoU	0.592	0.588	0.584
Precision	0.722	0.709	**0.723**
Recall	0.740	0.747	0.726

Table 33: Test set results for ResNet50 trained on 100 % of data. The LUPI model is marked in bold where it beats the RGB baseline, shown in the first column.

6.4 Extreme Privileged Information

As mentioned, we also wanted to attempt our finest methods in a scenario where the privileged information is more useful. We do this by using NIR as the non-privileged input, and RGB as privileged information. In other words, we "flip" what we consider to be privileged information. This flip is primarily done to show the usage of LUPI, rather than to argue for the usefulness of RGB images as privileged information when having access to NIR images. However, such a situation is still not unimaginable. The results for the 5 % models can be seen in table 34 below. The trainings are run using U-Net and Dice loss, just as usual. The LUPI model does not outperform the NIR TS baseline, and it seems like our methods do not hold for this attempt. Therefore, we have not attempted this on the full dataset.

	TS Baselines		LUPI
Teacher Input	NIR	RGB, NIR	RGB, NIR
Student Input	NIR	RGB, NIR	RGB
$\mathbf{T_S}, \mathbf{T_T}$	(1,1)	(1,1)	(2,7)
R	1/100000	1/100000	1/15000
Loss	0.184	0.160	0.186
mIoU	0.484	0.543	**0.485**
Precision	0.618	0.692	0.617
Recall	0.655	0.694	**0.657**

Table 34: Test set results for U-Net using RGB as privileged information, trained on 5 % of the data. The LUPI model is marked in bold where it beats the NIR TS baseline, shown in the first column.

6.5 Summary

All in all, it seems like the teacher student architecture works quite well for several neural networks when it comes to transferring privileged information. It did, however, not work on 100 % of the data when using the FCN-8s network. Given that FCN-8s does not yield good results overall, this is not too serious. One will always work with the best network at hand, and in this case, FCN-8s is not that network. The TS architecture also did not work too well when flipping the regular and privileged information, and for the 100 % ResNet50 models. A perfect architecture would need no hyperparameter tuning, but it is likely that some tuning had been necessary to make it work this time. In terms of parameters, we especially believe that some tuning of the temperature constants would improve the performance when trying different models or inputs. We have used U-Net when tuning all our parameters, and it should be mentioned that the probabilities that this network outputs are usually very high for the estimated class. The network is, so to say, very confident. By using temperatures, this effect is mitigated, and the student network compares its prediction to a smoothed version of the teacher's prediction. When using other networks, however, such as the FCN-8s, the confidence of the model have been a lot lower. In turn, this means that the teacher predictions might be too smoothed out when they are passed to the student, and could be what causes the teacher student architecture to generalize poorly in some cases. We have not had the time to look into this in much detail, but some more results and discussion about this is mentioned in section A.2 in the appendix.

When it comes to the extreme privileged information, we reduce the student input to only one channel, compared to three when using RGB as input. We wanted to see if the information could be transferred to the student, even when the teacher was relatively different from the student. A potential issue with this is that the student loses too much input and that the teacher is too different from the student. Remember that the student only receives information from the teacher imitating its smoothed predictions. It could be that the RGB channels contain so much spatial information that the network struggles to find shapes, edges and other objects. This does not explain why the transfer of privileged information does not work, but is a reminder maybe, that the teacher and the student cannot be too different. We also noted that the teacher student architectures improved the baselines for all above architectures, implying that the ensemble effect, of purely using a teacher student setup, is persistent.

7 Future Research

Clearly, there are many fields related to this work which need future research. We especially propose further studies of the temperatures T_T and T_S within the teacher student architecture. Some relevant questions are: how does T_T and T_S impact the performance of the student? How should T_T and T_S be set, depending on which network architectures are utilised, and how should T_T and T_S be set when the teacher and student input is very different? In general, how can the student be supported when the teacher input and student input is very different? Furthermore, the predicting and generating models showed some promise, and as they have many hyperparameters and could be rebuilt in many different ways, it could be worth studying these further. One idea would be to combine the prediction/generating of the NIR and elevation into one head, instead of using two separate heads, as we did in all experiments. Notably, it would be interesting to see if the proposed LUPI model holds in a setting with more useful privileged information, and how the hyperparameters should be set to help this. It would also be interesting to study the usage of multiple teachers. Furthermore, it would be valuable to study our setup in a different domain than land cover classification. Apart from the mentioned LUPI studies, the results have occasioned further studies on how well results generalise from a subset to the full dataset and the consequences (and solutions) of label noise.

8 Conclusion

In this thesis, we successfully explored all our main objectives related to utilizing privileged information for land cover classification. We demonstrated that networks with access to privileged information, outperformed those without it, confirming the value of privileged channels. Notably, we showed that the LUPI paradigm was successful in exploiting privileged information during training to improve performance during inference. Among the various LUPI architectures studied, the teacher student model proved particularly effective in transferring privileged information. Furthermore, the initially achieved results also generalised well to other network types.

Despite some challenges, such as the small performance gap between the RGB models and the privileged models, our proposed methods demonstrated significant potential for improving land cover classification. Additionally, while there may be a need for fine-tuning when generalising the proposed teacher student setup, it is easily implemented. Future research should focus on optimizing the teacher student model and the transferability of the architecture to other tasks and datasets.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Maximilian Bernhard, Tanveer Hannan, Niklas Strauß, and Matthias Schubert. Context matters: Leveraging spatiotemporal metadata for semi-supervised learning on remote sensing images. *arXiv preprint arXiv:2404.18583*, 2024.

[3] Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza. A survey and taxonomy of loss functions in machine learning. *arXiv preprint arXiv:2301.05579*, 2023.

[4] Kingma Da. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[6] Anatol Garioud, Nicolas Gonthier, Loic Landrieu, Apolline De Wit, Marion Valette, Marc Poupée, Sébastien Giordano, et al. Flair: a country-scale land cover semantic segmentation dataset from multi-source optical imagery. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science and Business Media, 2009.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Chengming Hu, Xuan Li, Dan Liu, Xi Chen, Ju Wang, and Xue Liu. Teacher-student architecture for knowledge learning: A survey. *arXiv preprint arXiv:2210.17332*, 2022.

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

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

[13] Rico Jonschkowski, Sebastian Höfer, and Oliver Brock. Patterns for learning with side information. *arXiv preprint arXiv:1511.06429*, 2015.

[14] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

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

[16] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015.

[17] Hongyu Pan, Hu Han, Shiguang Shan, and Xilin Chen. Mean-variance loss for deep age estimation from a face. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5285–5294, 2018.

[18] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

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

[20] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[23] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2022.

[24] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[25] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[26] Ioannis N Tzortzis, Konstantinos Makantasis, Ioannis Rallis, Nikolaos Bakalos, Anastasios Doulamis, and Nikolaos Doulamis. Learning using privileged information for segmenting tumors on digital mammograms. *arXiv preprint arXiv:2402.06379*, 2024.

[27] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural networks*, 22(5-6):544–557, 2009.

[28] Libo Wang, Rui Li, Ce Zhang, Shenghui Fang, Chenxi Duan, Xiaoliang Meng, and Peter M. Atkinson. Unetformer: A unet-like transformer for efficient semantic segmentation of remote sensing urban scene imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 190:196–214, August 2022.

A Appendix

In addition to the experiments reported thus far, we have trained other models, and analysed some results in more detail. In the end, these have not been essential to reaching our objectives, but some are still interesting and worth mentioning. This appendix contains the most important findings, and is mainly focused on overconfidence, loss functions, and ensemble models. Furthermore, we present a table containing the mean and standard deviation of all input channels.

A.1 Normalisation Table

Table 35 presents the mean and standard deviation for all five input channels. These were used to normalise the input, such that the input had a 0 mean and standard deviation of 1.

	B	G	R	IR	Height
Mean	113.7693	118.0902	109.2753	102.3808	16.7343
Standard deviation	52.3925	46.0153	45.2657	39.4576	29.5914

Table 35: Mean and standard deviation of the input channels.

A.2 Overconfidence

As a first step, we were curious about the phenomena of overconfidence, and how to reduce it. It should be noted that in our case, we are not particularly worried about the overconfidence itself. One can imagine cases where it is important that the model's confidence level reflects the true accuracy. E.g. for self driving cars it is highly relevant to know how confident the model is of not hitting a wall in the next ten meters. If the model's confidence is aligned with its accuracy, the car can be stopped when the model is insecure which coincide with when the model might be incorrect. However, in the case of Earth observation, the alignment between confidence and accuracy itself is not very important as there is no need to stop the model if it is overconfident (even though samples for which the model's accuracy is low could be sorted out to humans, if getting everything right is more important than time consumption). Rather, the loss function rewards overconfidence, which we are not very impressed by when actually evaluating the model. I.e. we want to avoid overconfidence as we want the loss to award improved performance, rather than overconfidence. Hopefully, it is now clear to the reader why we think understanding and decreasing the level of overconfidence might have an impact on the model's performance.

A.2.1 Confidence of Our Models

The first question we wanted to answer was if our models actually were overconfident. As explained, our models output some softmax probability for each class, and the segmentation is done by assigning each pixel to the class with highest probability. We used these probabilities to calculate an average probability for all correctly labelled pixels, and the same for incorrectly labelled pixels. Table 36 shows the result from using a regular U-Net and a TS architecture based on that U-Nets, for 5 % and 100 % of the data.

Model	RGB, NIR, elevation	RGB, NIR, elevation	TS LUPI	TS LUPI
Dataset Size	5 %	100 %	5 %	100 %
Dice loss	0.167	0.148	0.181	0.151
Accuracy	0.747	0.779	0.767	0.785
Avg. prob. correct	0.992	0.998	0.879	0.970
Avg. prob. incorrect	0.939	0.981	0.711	0.862

Table 36: Table of the average probabilities for predicted label, along with accuracy.

We see that both our baseline models, on average, predict with probabilities that are higher than the accuracy, which indicates overconfidence. The teacher student architectures are a bit different. For both the 5% and the 100% models, correct classes are predicted with probabilities that are also higher than the accuracy, but not by as much. The loss is also higher for both teacher student models, compared to the baseline, even though their overall performance is better. Note also that there is a larger difference between correctly and incorrectly labelled pixels when comparing our baseline to our teacher student setup. It seems like the teacher-student are not as prone to only minimizing the loss function without increasing performance, and thus avoiding overconfidence. Besides that, they seem to be not as certain when they output wrong labels compared to correct labels, which is a good thing.

A.2.2 Reducing Overconfidence

We have concluded that our baseline models are overconfident, and that the teacher-student architecture reduces this effect, although not completely. The question is what else we can do about it.

Label Smoothing
We started with introducing general label smoothing. This can be done for any type of model by smoothing the labels in the normal loss function. The smoothing can be done either uniformly over all labels, or possibly by re-weighting the labels differently between classes, as some classes are more similar to each other than others. In our case, the label smoothing trainings were ran with CE loss and a smoothing of 0.2, which means each label y_i is 0.8 and then distributed over all other classes, instead of 1 and then 0 for all other classes. It turned out this label smoothing indeed suppressed the overconfidence initially, but as the training went on, the confidence increased, and the model became overconfident again. Furthermore, the performance of the model was not improved. We therefore discarded the usage of smoothing.

Focal Loss
We then tried to change our loss function into a focal loss. This loss focuses less on "easy" samples, which are considered to be those which the model are very confident of, while being correct. Focal loss is a common loss used in semantic segmentation tasks. We ran experiments both with focal loss and with a combination of Dice loss and focal loss. Just as for smoothing, it turned out that focal loss indeed suppressed the overconfidence initially, but as the training went on the confidence increased, and the model became overconfident again. Again, the performance of the model was not improved. Hence, we discarded the usage of focal loss too.

General
As mentioned, the teacher student architecture, especially using some label smoothing/temperature T_T and T_S, offer less award to overconfidence. It might be possible to *exploit* the confidence gaps. As seen, our teacher student models are way more confident when they are correct, than when they are not. As seen in the results section above, our teacher student models trained with (priv, priv), (RGB, RGB), (NIR+elevation,

RGB) perform differently on different classes. Hence, letting them vote by using some ensemble model, might be a way to exploit the fact that there is a gap in the confidence when the model is incorrect. This way, we could improve the results without having to train any new models.

A.3 Loss Functions

During our various trainings, we logged the CE loss, the focal loss and the Dice loss (which also was the one the network was using for training) on our validations. It was noticed that the CE and focal loss often got successively higher, even as the Dice loss went down. As we had also successfully trained nets with both CE and focal loss (however, with somewhat worse results than Dice loss), we know all the losses contains valuable information. Hence, we figured that including some level of CE and/or focal loss during (i.e. using a combined loss with Dice and CE/focal loss) training might be beneficial, as there seemed to be an orthogonal component in those losses, compared to the Dice loss. This was tried, and it indeed kept the respective losses down a lot better, but the performance of the model on the mIoU, was not improved. Hence, we dismissed the idea of using combined losses.

A.4 Ensemble Model

In the tables below we will shortly present all models used in the 5 % ensemble model, in which all models will have been trained on only 5 % of the data. The first table 37 shows all included architectures which are not teacher student models. All of these predictions are smoothed by $T = 2$, meaning their prediction is halved before inputting it into the softmax function. This is since all non teacher student architectures are very overconfident. The second table, 38, lists all teacher student models included in the 5% ensemble model. All teacher student losses are KL loss, T stands for teacher and S for student. The "type" column indicates if it is a privileged model, a LUPI model or an RGB model. In the teacher student table, priv will then imply that the teacher and student had access to RGB, NIR and elevation, while LUPI will imply that the teacher had access to RGB, NIR and elevation but the student only to RGB and finally RGB will imply that both the teacher and the student had access only to RGB. For the privileged ensemble model, all LUPI and RGB models are also used and for the LUPI model all RGB models are also used. It should also be noted that the amount of time spent on the ensemble model is low, and no optimization has been done in terms of which models to include. Furthermore, the ensemble model could possibly have been improved by using suitable temperatures for smoothing for more models than currently done, but this is unfortunately outside the scope of this thesis.

Type	**Architecture**	**Loss Function**	**Notes**
priv	U-Net	Dice	-
RGB	U-Net	Dice	-
priv	U-Net	Dice	only NIR, elevation
LUPI	Zero out	Dice	-
LUPI	Multi decoder	MSE, Dice	-
priv	U-Net metadata	metadata loss	-

Table 37: Table of models included in 5 % ensemble model, which are not teacher student models.

Type	**T Architecture**	**S Architecture**	**Student Loss**	**Notes**
priv	U-Net	U-Net	Dice	-
RGB	U-Net	U-Net	Dice	-
LUPI	U-Net	U-Net	Dice	-
LUPI	U-Net	U-Net	Dice	only NIR, elevation
LUPI	U-Net	Multi decoder	Dice+CE	-
LUPI	U-Net	U-Net metadata	Dice	-
priv	U-Net	U-Net metadata	Dice+CE+focal	-
RGB	U-Net	UNetFormer	CE	-
LUPI	U-Net	UNetFormer	CE	-
priv	U-Net	UNetFormer	CE	-

Table 38: Table of teacher student models included in 5 % ensemble model.

Now follows the same experiment for the 100 % models. That is to say, the same type of architectures are used, handled in the same way. The models are presented in table 39 and table 40 below. Due to computational limitations some of the teacher student models on 100 % are utilizing pretrained models, and are even fine-tuned only on 5 % of the data.

Type	**Architecture**	Loss **Function**	**Notes**
priv	U-Net	Dice	-
RGB	U-Net	Dice	-
priv	U-Net	Dice	only NIR, elevation
LUPI	Zero out	Dice	-
LUPI	Multi decoder	MSE, Dice	-
priv	U-Net metadata	metadata loss	-

Table 39: Table of models included in 100 % ensemble model, which are not teacher student models.

Type	**T Architecture**	**S Architecture**	**Student Loss**	**Notes**
priv	U-Net	U-Net	Dice	-
RGB	U-Net	U-Net	Dice	-
LUPI	U-Net	U-Net	Dice	-
LUPI	U-Net	U-Net	Dice	only NIR, elevation, tuned on 5 %
LUPI	U-Net	Multi decoder	Dice+CE	pretrained student
LUPI	U-Net	U-Net metadata	Dice	-
priv	U-Net	U-Net metadata	Dice+CE+focal	-
RGB	U-Net	UNetFormer	CE	-
LUPI	U-Net	UNetFormer	CE	-
priv	U-Net	UNetFormer	CE	-

Table 40: Table of teacher student models included in 100 % ensemble model.

Master's Theses in Mathematical Sciences 2024:E23
ISSN 1404-6342

LUTFMA-3535-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/