National Technical University of Athens

Interdisciplinary Master's Programme in
Data Science and Machine Learning

THIRD LAB IN THE COURSE OF GEOSPATIAL BIG DATA AND ANALYTICS

Classification and Segmentation of Hyperspectral Images

Written by: Christos Nikou
ID: 03400146

EMAIL:CHRISTOSNIKOU@MAIL.NTUA.GR

August 3, 2022

Contents

1 Introduction

1.1 The problem

In this third lab we begin our investigation into the Deep Learning framework. The problem at hand is the classification of hyperspectral images. Hyperspectral image classification has been a vibrant area of research in recent years. Given an image, usually obtained from a satellite, the goal is to assign each pixel of the image to a class from a predetermined set of classes. In the case of satellite images, one is usually interested in mapping each pixel to the corresponding land use; e.g. urban fabric, forest, water etc.

The main difference from the classical setting of the RGB image classification is that the hyperspectral images consist of hundreds of spectral bands. This difference has its advantages and its drawbacks. On the one hand, the more spectral bands correspond to more features, which in turn translates in a more informative dataset to train a model. However, these extra features pose some difficulties at the same time. For example, having high dimensional features might be really hard to handle; either when it comes to how to construct the dataset or on how to implement the training procedure. The computational cost and the required resources to train a model rises quickly as the dimensions of the features increase. Some other issues are the presence of redundant features and the imbalance among the limited number of available training samples.

To tackle these problems several algorithms and approaches have been developed. Among those approaches are support vector machines (SVMs), random forests, neural networks and deep learning architectures. For our purposes we make use of almost all (except from the SVMs) the aforementioned techniques in order to make pixel-wise predictions for three different satellite images. In addition, we use several techniques and approaches on how we pre-process the training images. For the random forest and the Multi layer perceptron (MLP) neural network we follow the pixel-based approach described in 2.1. For the CNN([6]) we use the patch-based approach where we generate squared images of size $M \times M$ from the initial image of size $H \times W$. In this case we consider two different approaches; the overlapping approach where we allow overlaps between the generated images and the non-overlapping approach. These approaches are described in detail in 3.1, 3.2. The last model is based on the UNet architecture as presented in [5]. Through the experiments the superiority of the deep architectures over the conventional ML models becomes evident. As we shall shortly see, the best models for this problem are the CNN and the UNet. The comparison of the models is presented thoroughly in 5.1. Finally, we use the best model according to our criteria in order to make predictions to three satellite images (Erato, Kirki and Nefeli) for which the ground truth labels are unknown. The experiments were developed using the programming language Python and PyTorch library.

1.2 The dataset

The dataset that we utilized to train and validate our models is the HyRANK dataset which was developed by a scientific initiative of the ISPRS, WG III/4. The main goal with the HyRANK dataset is to provide a dataset with hyperspectral images along correspondent ground truths, to enable the scientific community to validate new classification approaches against the state-of-the-art methods. The dataset is introduced in [3] and it is composed of five hyperspectral images gathered with the Hyperion sensor Earth Observing-1. After a preprocessing step, these images were provided with 176 surface reflectance bands. The ground truths from two (Loukia and Dioni) out of five images were provided. 14 Land Use and Land Cover (LULC) were annotated in the ground truth following the CORINE Land Cover principles. These classes are presented in Table 1.

Table 1: The 14 LULC classes of the HyRANK dataset.

Label	Class Name	Color Code	Color
0	Non Defined	#000000	
1	Dense Urban Fabric	#ff0000	
2	Mineral Extraction Sites	#a600cc	
3	Non Irrigated Arable Land	#ffffa8	
4	Fruit Trees	#f2a64d	
5	Olive Groves	#e6a600	
6	Broad-leaved forest	#80ff00	
7	Coniferous-Forest	#00a600	
8	Mixed Forest	#a600cc	
9	Dense Sclerophyllous Vegetation	#819c25	
10	Sparse Sclerophyllous Vegetation	#e6cc4d	
11	Sparcely Vegetated Areas	#e6e64d	
12	Rocks and Sand	#cccccc	
13	Water	#4d4dff	
14	Coastal Water	#80f2e6	

From the above Table we observe that there is a class with name "Non Define" which corresponds to the unknown land uses. To train our models properly we have to exclude the pixels that correspond to undefined areas. The approach for each individual model is different and in some cases is the most challenging part of the construction of the learning framework. The images of Dioni and Loukia and their corresponding ground truth labels are depicted in Figures 1, 2.

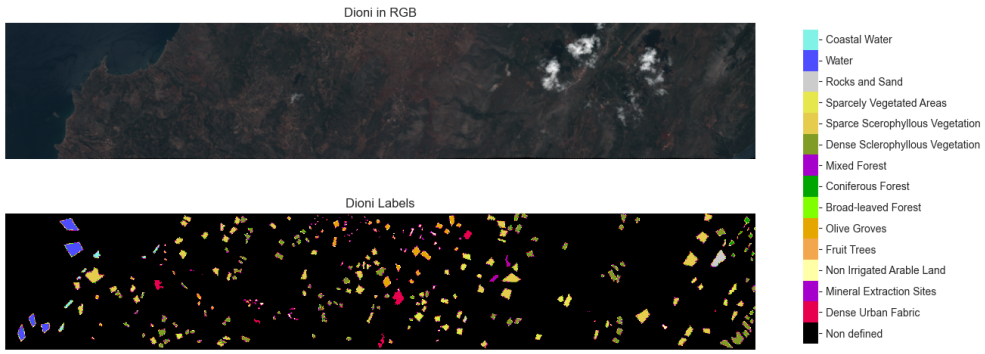

Figure 1: The image of Dioni in RGB composite and the corresponding ground truth labels. The size of the image is 250×1376 with 176 spectral bands and the spatial resolution is 30m. Pixels shown in black color correspond to undefined land uses.

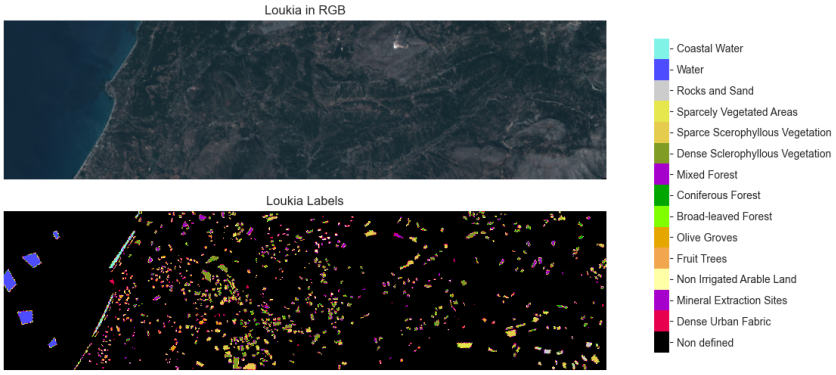

Figure 2: The image of Loukia in RGB composite and the corresponding ground truth labels. The size of the image is 249×945 with 176 spectral bands and the spatial resolution is 30m. Pixels shown in black color correspond to undefined land uses.

The ground truth labels are not provided for the other three images (Erato, Kirki and Nefeli) and the purpose of this lab is to make predictions on these three images using the most promising model with respect to its generalization power. In Figure you can see Erato, Kirki and Nefeli in RGB composite.

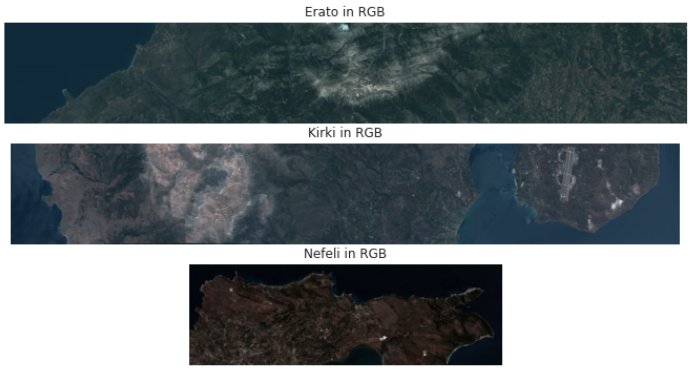

Figure 3: The validation images in RGB composite. Each image has spatial resolution equal to 30m and 176 spectral bands. On top is Erato with size 241×1632. In the middle is Kirki with size 245×1626 and in bottom is Nefeli with size 249×772. The ground truth labels are not given and the end goal is to make predictions for these images.

2 Training some classic Classifiers

Before we proceed with the more complex deep architectures we start by testing some basic classifiers on the problem. In this section we adopt the pixel-based approach to construct our dataset and we train a random forest and a MLP classifier. These two models serve the basis of our investigation and give us an idea on what to expect from the deep approaches adopted in Sections 3 and 4.

2.1 Pixel-based approach

In the pixel-based approach our features consist of all the available pixels from the images of Dioni and Loukia. Each pixel can be represented as a C dimensional vector with real entries. The letter C is the number of spectral bands which is 176 in our case. For example, suppose that our image is of size $C \times H \times W$ and $N \leq H \cdot W$ is the number of pixels for which the land coverage is defined. Then, the dataset obtained from the image can be compactly represented as a matrix $\mathbf{X}$ with dimensions $K \times C$ in the form of:

$$\mathbf{X} = \begin{pmatrix} x_{11} & \dots & x_{1C} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NC} \end{pmatrix}. \tag{2.1.1}$$

The labels are represented by a N dimension vector $\mathbf{y} \in \mathbb{R}^{N \times 1}$ with entries in $\{1, \dots, K\}$, with K being the number of individual classes. In this setting we can train any classic ML classifier. In our case, we trained a Random forest classifier using Scikit-learn library and a MLP classifier

using PyTorch to define the architecture and training procedure. One important issue that arises at this point, as well as in the other cases, is the high dependency between spatial neighboring pixels. In satellite images we expect neighboring pixels to be almost identical. Therefore, when we split the dataset into training and test test, we will have an information leakage onto the test set. This results in extremely high scores on the test set. However, we cannot rely on these scores to estimate the generalization performance since if we use a different image to make predictions we might obtain inaccurate results. In the following subsections we adopt two different approaches; first, we combine the datasets in the form of (2.1.1) obtained from the two images and then we split the dataset into training and test sets. In the second approach, we use only the largest image (Dioni) as the training set and validate on Loukia. As will shall shortly see, the results are very different in each case proving that we cannot rely on these classifiers to predict the three unknown images.

2.2 Random forest

For the Random forest classifier we used scikit-learn's implementation. To transform and bring Dioni and Loukia in the form of (2.1.1) we developed a custom function named *standard_dataset* located in *lib.py* file. The main functionalities are to determine the pixels for which the land use is undefined, to reduce and resize the images from $C \times H \times W$ to $(H \cdot W) \times C$ and keep only the pixels for which the label corresponds to non zero entry and finally to normalize the matrix X using min-max normalization column-wise; i.e. normalize all samples with respect to each spectral dimension.

2.2.1 Training on both images

The first approach was to combine both images and then use scikit-learn's method *train_test_split* with train-test ratio 0.67-0.33 to split the whole dataset. The training set consists of 22463 samples and the test set of 11064. As we already stated above, with this approach we expect to have extremely high scores on the test set since we train the model on a training set which is almost identical to the test set. The obtained model will only be useful if we were to use it on images similar to the images used for training or use it in order to completely annotate the training images. However, we cannot rely on such a model in order to make predictions to Erato, Kirki and Nefeli. In Table 2 you can see the results obtained from the classification report and in Figure 4 the corresponding confusion matrix for the random forest.

Table 2: Classification report of Random forest trained on Dioni & Loukia.

Category	Precision	Recall	F1-Score	Support
Dense Urban Fabric	0.91	0.72	0.80	513
Mineral Extraction Sites	0.96	0.88	0.92	103
Non Irrigated Arable Land	0.90	0.84	0.87	386
Fruit Trees	0.90	0.64	0.75	73
Olive Groves	0.92	0.84	0.88	1026
Broad-leaved Forest	0.93	0.60	0.73	72
Coniferous Forest	0.95	0.81	0.88	291
Mixed Forest	0.73	0.70	0.71	355
Dense Sclerophyllous Vegetation	0.87	0.90	0.88	2861
Sparce Scerophyllous Vegetation	0.87	0.93	0.90	3104
Sparcely Vegetated Areas	0.90	0.89	0.89	728
Rocks and Sand	0.96	0.95	0.96	311
Water	0.99	1.00	1.00	950
Coastal Water	1.00	0.98	0.99	291
Accuracy			0.89	11064
Macro Avg.	0.91	0.84	0.87	11064
Weighted Avg.	0.89	0.89	0.89	11064

As we can see, without any extravagant effort, the overall accuracy is 89% which is only 2% less than reported results on the same dataset, e.g. [1]. The fitting time was 19.0469 seconds on a local laptop machine with 2.8GHz CPU and 4 cores. From the above table we see that the model is able to distinguish the water from coastal water. By looking at the confusion matrix below we see that only 7 pixels from coastal water were wrongly classified as water. The biggest confusion seems to be between the dense sclerophyllous and sparce sclerophyllous vegetation classes. As we can see 179 out of 3104 pixels corresponding to sparce slecrophyllous vegetation were mistakenly classified as dense while 208 out of 2861 of dense sclerophyllous vegetation pixels were classified as sparce.

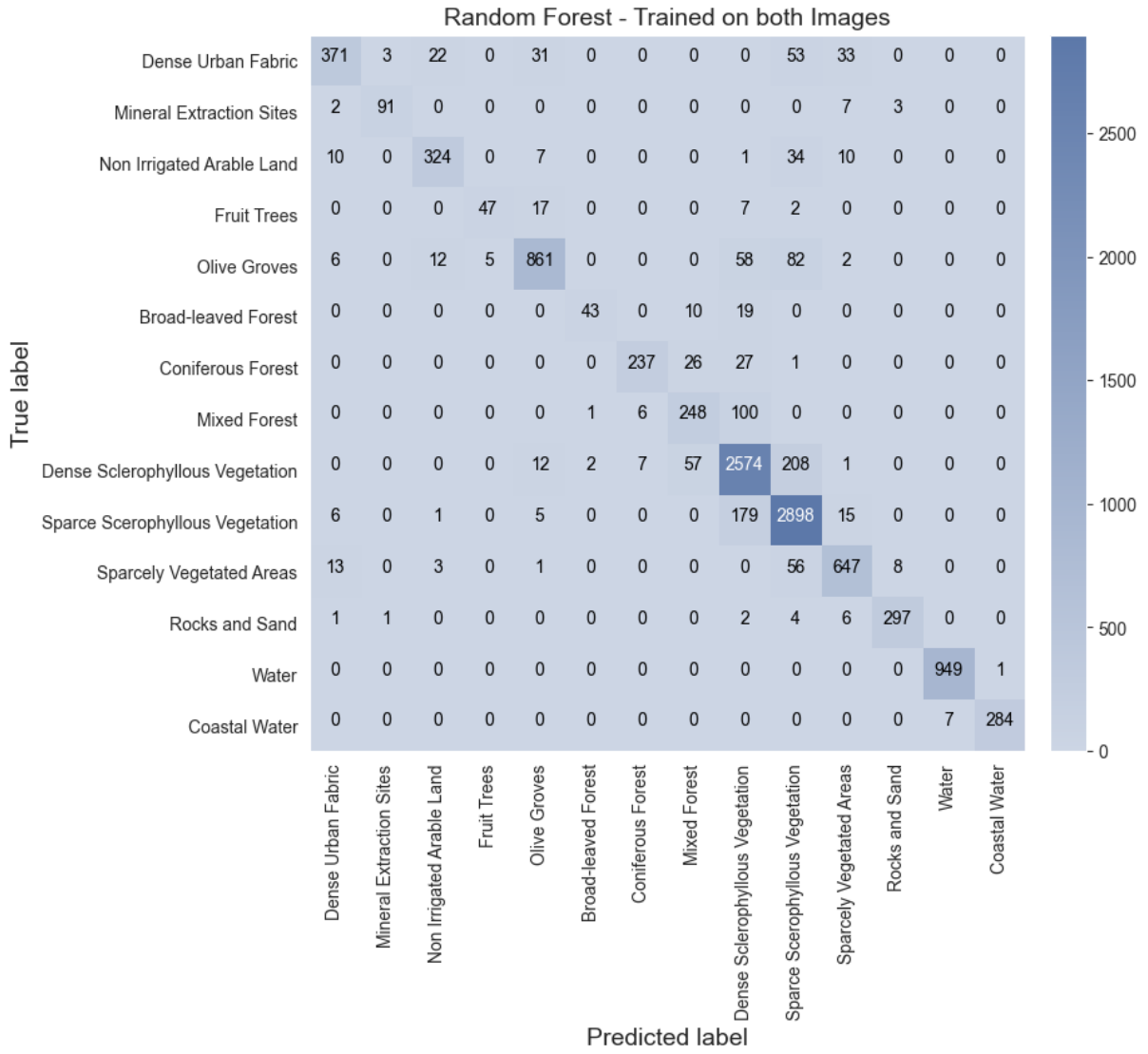

Figure 4: The Confusion matrix obtained from the random forest when trained on both images, Dioni and Loukia.

2.2.2 Training on Dioni validating on Loukia

Now, our training data consists only of the pixels from Dioni and the test dataset of the pixels that correspond to Loukia. The training data consists of 20024 samples and the test data of 13503 samples. The fitting time was 15.1250 sec. Below you can see the results for this case. This time, the scores have dramatically decreased as opposed to the previous case and this is because the model captured the noise from the data and not the useful information needed in order to make accurate predictions.

Table 3: Classification report of Random forest trained only on Dioni and validated only on Loukia.

Category	Precision	Recall	F1-Score	Support
Dense Urban Fabric	0.14	0.05	0.08	288
Mineral Extraction Sites	0.37	0.88	0.52	67
Non Irrigated Arable Land	0.91	0.16	0.27	542
Fruit Trees	0.50	0.05	0.09	79
Olive Groves	0.00	0.00	0.00	1401
Broad-leaved Forest	0.00	0.00	0.00	223
Coniferous Forest	0.53	0.32	0.40	500
Mixed Forest	0.00	0.00	0.00	1072
Dense Sclerophyllous Vegetation	0.52	0.92	0.66	3793
Sparce Scerophyllous Vegetation	0.58	0.79	0.67	2803
Sparcely Vegetated Areas	0.18	0.22	0.19	404
Rocks and Sand	0.00	0.00	0.00	487
Water	0.99	1.00	0.99	1393
Coastal Water	1.00	1.00	1.00	451
Accuracy			0.59	13503
Macro Avg.	0.41	0.38	0.35	13503
Weighted Avg.	0.47	0.59	0.50	13503

As we can see, the overall accuracy has dropped by 30% as opposed to the previous case. By looking at Table 3 we see that there are some classes that the model is unable to distinguish. For example, we see that in all metrics the score is 0 for pixels corresponding to rock and sand. Similar phenomenon occurs for the mixed forest, broad-leaved forest and olive groves areas. An interesting observation is that the model is still able to distinguish the water from coastal water by achieving the highest score in all metrics. One explanation for this might be that the water areas experience less variation throughout the year as compared to the areas of vegetation and forest. Therefore, if Dioni and Loukia are captured in different time seasons it is expected from the model to make inaccurate classifications for these classes if it has learned the noise instead of the underlying similar characteristics of these pixels. In Figure 5 we see the corresponding confusion matrix for the above classification.

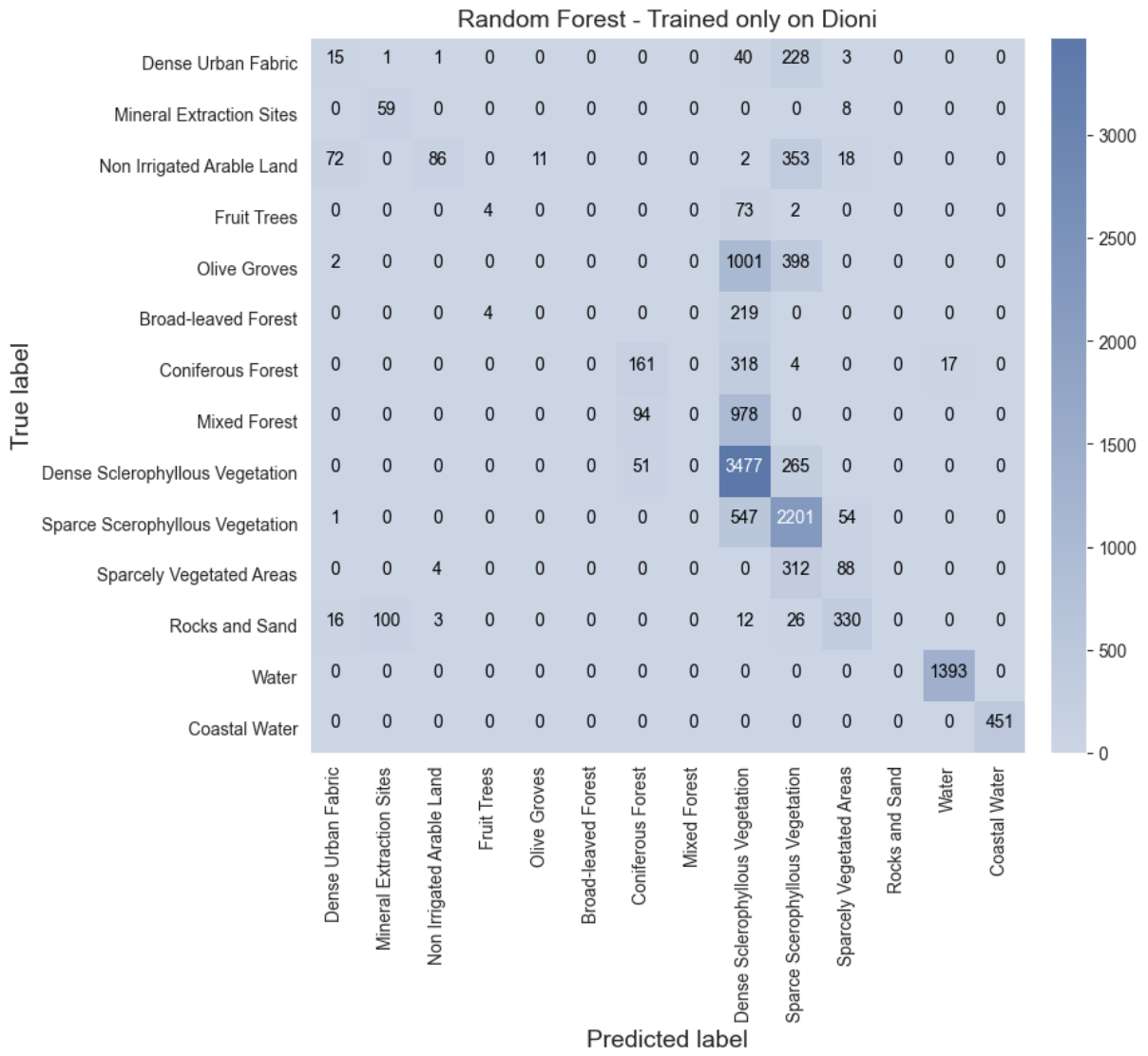

Figure 5: The Confusion matrix obtained from the random forest when trained only on Dioni and validated on Loukia.

2.3 MLP

2.3.1 The architecture and the training set up

Similarly, for the MLP we begin by first combining Dioni and Loukia and train the model on the whole dataset and on a second test with train only on Dioni and validate on Loukia. As in the case of the random forest we observe the same phenomenon. For the development of the MLP we used the PyTorch library to define the architecture of the model, to handle the data and to construct the training and validation methods. The model consists of 4 hidden layers with sizes 256, 512, 256 and 128, respectively. The activation function that we use is the ReLU

given by $g(x) = x \cdot \mathbb{1}_{x \geq 0}$.[1] After each linear transformation we use a 1D batch normalization and before the application of the next transformation we use a dropout with probability $p = 0.3$. The architecture of the model is handled by a custom class named *MLP_Net* which is located in *lib.py* file and inherits from the *nn.Module*. The class is designed in such a way that allows the user to define the number and the size of each hidden layer in an interactive manner. The transformation of the dataset in order to be compatible with the MLP is carried out by the custom class named *MLP_Dataset*. The class simply accepts the array $\mathbf{X}$ as described in (2.1.1) and the vector of labels $\mathbf{y}$ and converts them from numpy arrays to tensors. The training procedure is carried by a custom function named *training_loop* located also in *lib.py*. This function will be used also later on to train the more sophisticated deep learning models as well. The loss function that we use to train all networks is the cross-entropy. At this point we must be careful about how to handle the index 0 which corresponds to the undefined areas. In general, the MLP accepts as input a vector $\mathbf{x} \in \mathbb{R}^{C \times 1}$, where $C = 176$ in our case and through a series of operations outputs a vector $\mathbf{z} \in \mathbb{R}^{K \times 1}$, where K is the number of classes which is equal to $K = 14$ in our case. Then, the vector $\mathbf{z}$ is mapped through the soft-max function to a probability vector $\hat{\mathbf{y}} \in \mathbb{R}^{K \times 1}$ given by

$$\hat{\mathbf{y}} = \left(\frac{e^{z_1}}{\sum_{j=1}^{K} e^{z_j}}, \ldots, \frac{e^{z_K}}{\sum_{j=1}^{K} e^{z_j}} \right), \tag{2.3.1}$$

where $\mathbf{z} = (z_1, \ldots, z_K)^T$. Now, if $\mathbf{y} \in \mathbb{R}^{K \times 1}$ is the ground truth label vector with all entries being zero except one which is equal to 1, the cross-entropy is given by

$$\mathscr{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{K} y_i \cdot \log(\hat{y}_i). \tag{2.3.2}$$

In the case of mini-batch training with size N now the output of the neural network is a matrix of size $N \times K$ with each row $1 \leq j \leq N$ row being equal to the output vector $\mathbf{z}_j \in \mathbb{R}^{K \times 1}$. Then, the cross-entropy with mean reduction is given as the average

$$\mathscr{L} = \frac{1}{N} \sum_{j=1}^{N} \mathscr{L}(\hat{\mathbf{y}}_j, \mathbf{y}_j), \tag{2.3.3}$$

where $\hat{\mathbf{y}}_j$ is the probability vector corresponding to the output $\mathbf{z}_j$, $\mathbf{y}_j$ is the ground truth label and $\mathscr{L}(\hat{\mathbf{y}}_j, \mathbf{y}_j)$ is calculated as in (2.3.2). Now, the problem in our case is that we have a redundant class label which represents the undefined areas and therefore we need somehow to find a way to exclude this label from the calculation in (2.3.3). Luckily, PyTorch provides an extra argument to the cross-entropy loss function to handle this situation; the *ignore_index*. By setting *ignore_index* equal to 0 the calculation of (2.3.3) becomes

$$\mathscr{L} = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{K} y_i \cdot \log(\hat{y}_i) \cdot \mathbb{1}_{y_i \neq 0}. \tag{2.3.4}$$

[1] By $\mathbb{1}_A$ we denote the indicator function on the set A.

For technical reasons instead of working with the set $\{0,\dots,14\}$ for the class labels we work with the set $\{-1,0,\dots,13\}$ and we use *ignore_index*=-1. Another form of regularization that we make use of is the early stopping. The early stopping is handled by a class named *EarlyStopping* located in *lib.py*. The early stopping tracks the change of the validation loss in each training epoch and keeps a checkpoint for the model with the lowest validation loss up until that point. The hyperparameter *patience* sets an upper bound to the number of consecutive epochs that the validation loss did not decrease further from the minimum validation loss up until that point. The last regularization technique that we use is the ℓ_2-regularization. If we denote by W the set of all weights of the neural network, we add an additional penalty term to (2.3.4) to combat overfitting by reducing the capacity of the model. In the case of the ℓ_2-regularization (2.3.4) becomes

$$\widetilde{\mathscr{L}} = \mathscr{L} + \lambda \cdot \sum_{\mathbf{w}\in W} |w|^2, \tag{2.3.5}$$

where λ is the regularization penalty hyperparameter. High values of the penalty parameter $\lambda > 0$ correspond to adding greater penalty to the weight parameters of the network. In this way, we are deducing the capacity of the model in order to combat overfitting. In other words, we restrict our initial model space into a space that contains models that exhibit lower variance in the cost of higher bias. On the other hand, small values for the parameter λ correspond to a larger space of models. Therefore, in this case we expect to have models with low bias but with higher variance. The goal of ℓ_2-regularization is to achieve a compromise between bias and variance in order to obtain a model that fits the data well but without completely losing its predictive power. To train the MLP we use $\lambda = 10^{-5}$ and we set the batch-size equal to $N = 64$ in both cases; either we use the combined dataset or we use only the pixels obtained from Dioni.

2.3.2 Training on both images

Since we have constructed the architecture of the model, the training loop and the construction of the dataset the only part that's left is to define the hyperparameters. We use a batch-size equal to $N = 64$, the penalty parameter λ is set to 10^{-1}, the learning rate equal to 10^{-5} and the number of epochs is equal to 200. The patience is equal to 30. We first split the dataset into a training and test set with sizes 80% and 20% and then we further split the resulting training set into 80% for the training and 20% for the validation set. The MLP trained for 100 epochs out of the 200 since the validation loss did not decrease from its lowest value 0.4913 for more than 30 epochs. The training time was 4 mins and 12.4432 secs. During the training procedure we keep track of the mean (with respect to the number of batches) training and validation loss for each epoch and we plot a graph of the two losses for each epoch. In Figure 6 you can see this graph and in Table 4 you can see the classification report obtained from the testing procedure.

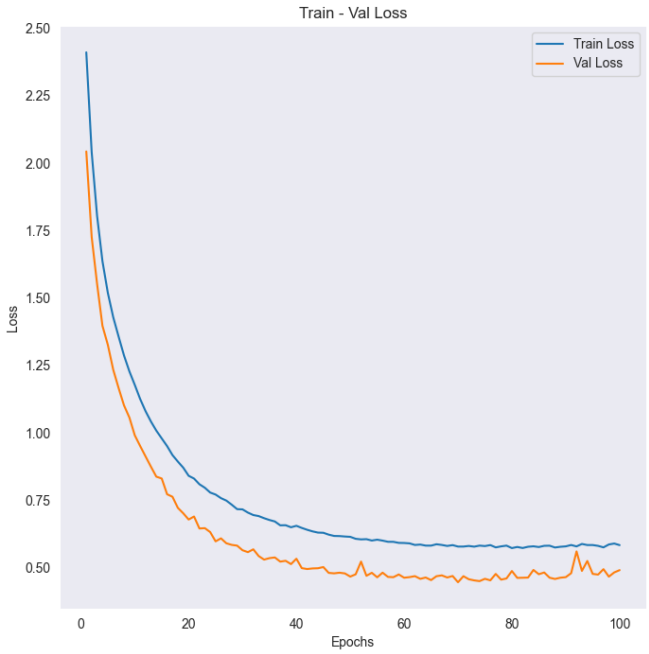

Figure 6: The training and validation loss of the MLP when trained on both images.

Table 4: Classification report of MLP trained on Dioni & Loukia.

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Dense Urban Fabric	0.89	0.85	0.87	318
Mineral Extraction Sites	1.00	0.67	0.80	54
Non Irrigated Arable Land	0.95	0.87	0.91	218
Fruit Trees	0.00	0.00	0.00	38
Olive Groves	0.90	0.96	0.92	635
Broad-leaved Forest	0.00	0.00	0.00	37
Coniferous Forest	0.99	0.61	0.76	145
Mixed Forest	0.61	0.48	0.53	233
Dense Sclerophyllous Vegetation	0.87	0.92	0.89	1827
Sparce Scerophyllous Vegetation	0.91	0.96	0.93	1852
Sparcely Vegetated Areas	0.95	0.91	0.93	385
Rocks and Sand	0.93	0.93	0.93	180
Water	1.00	1.00	1.00	591
Coastal Water	1.00	0.99	0.99	193
Accuracy			0.90	6706
Macro Avg.	0.79	0.72	0.75	6706
Weighted Avg.	0.89	0.90	0.90	6706

As we can see the overall accuracy is 90% which is 1% more than the random forest. Again the most distinguishable land uses are water and coastal water. There are two land uses, fruit trees and broad-leaved forest, that the model did not make any accurate prediction. By looking at the support of these two classes we see that in the former class we have 38 samples and in the latter we have only 37. Therefore, we cannot draw any conclusions for these two classes. Below we see the corresponding confusion matrix for this classification.

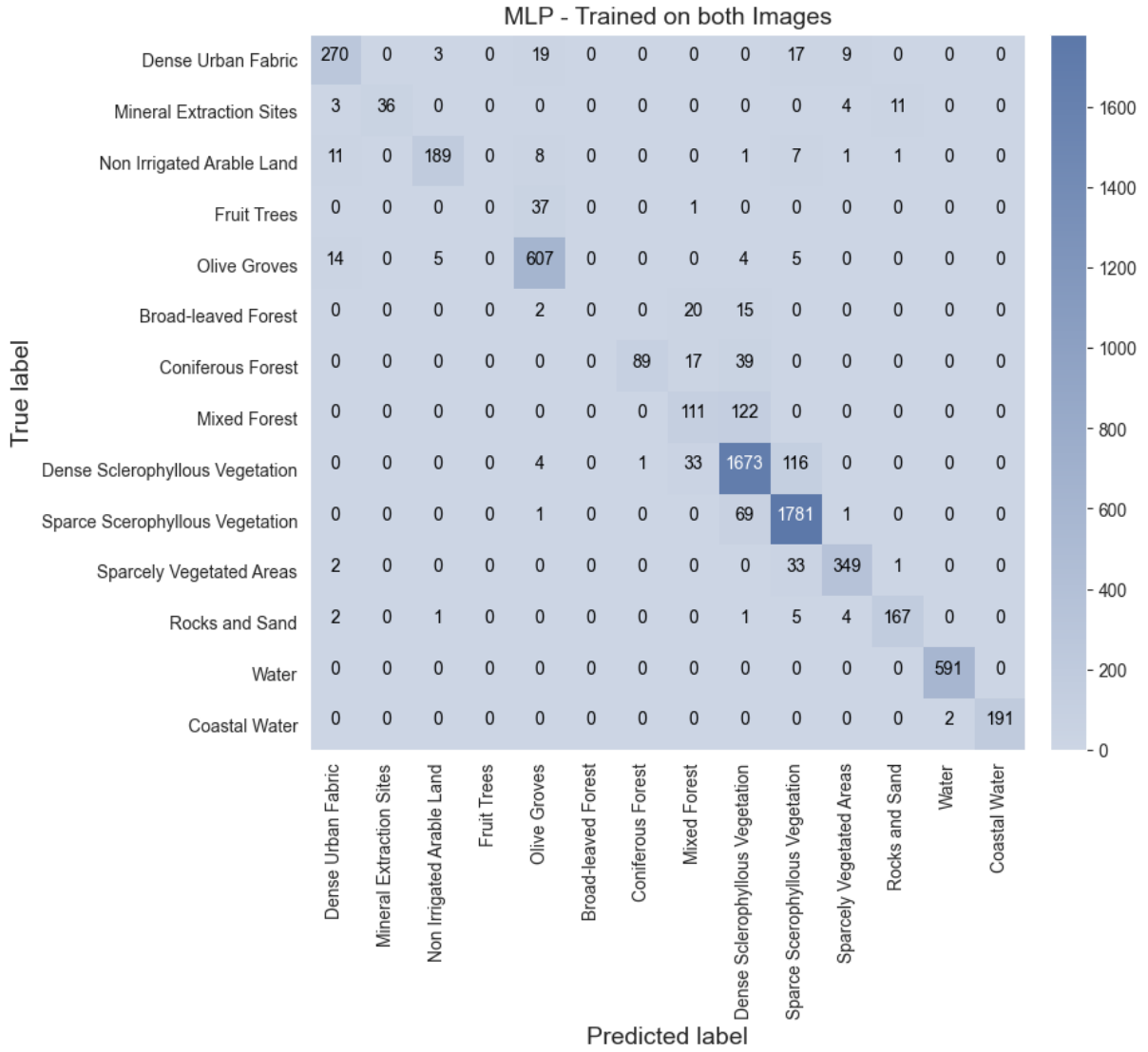

Figure 7: The Confusion matrix obtained from MLP when trained on both images.

By looking at the confusion matrix we see again that the model is having difficulties to distinguish the dense sclerophyllous vegetation from the sparce sclerophyllous vegetation and vice versa. Moreover, we see that 122 samples belonging to dense vegetation areas were mistakenly classified as mixed forest.

2.3.3 Training on Dioni validating on Loukia

As in the case of the random forest we perform the same experiment to train the MLP model described in 2.3.1 only on Dioni and validate on Loukia. To compare with the previous case we kept the training set up intact. The batch-size is again equal to $N = 64$, the learning rate is 10^{-5}, the ℓ_2-penalty parameter is 10^{-1} and we trained let the training to last at most 200 epochs if no early stopping is encountered. The training part lasted for 4 mins and 0.1774 secs. Below you can see the classification report.

Table 5: Classification report of MLP trained only on Dioni and validated on Loukia.

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Dense Urban Fabric	0.40	0.51	0.45	288
Mineral Extraction Sites	0.38	0.75	0.50	67
Non Irrigated Arable Land	0.84	0.38	0.52	542
Fruit Trees	0.00	0.00	0.00	79
Olive Groves	0.80	0.21	0.33	1401
Broad-leaved Forest	0.00	0.00	0.00	223
Coniferous Forest	0.00	0.00	0.00	500
Mixed Forest	0.00	0.00	0.00	1072
Dense Sclerophyllous Vegetation	0.50	0.88	0.64	3793
Sparce Scerophyllous Vegetation	0.69	0.81	0.75	2803
Sparcely Vegetated Areas	0.18	0.24	0.20	404
Rocks and Sand	0.00	0.00	0.00	487
Water	1.00	1.00	1.00	1393
Coastal Water	0.99	1.00	0.99	451
Accuracy			0.61	13503
Macro Avg.	0.41	0.41	0.38	13503
Weighted Avg.	0.55	0.61	0.54	13503

This time the overall accuracy dropped by 29% reaching 61%. Again it seems that the model learns almost perfectly the water and coastal water pixels and the most difficult pixels to decipher are those that correspond to fruit trees, broad-leaved forest, coniferous forest, mixed forest and rocks and sand. Below we see again the graph obtained from the training procedure which shows that the models were able to capture the information from Dioni and in Figure 9 the resulting confusion matrix.

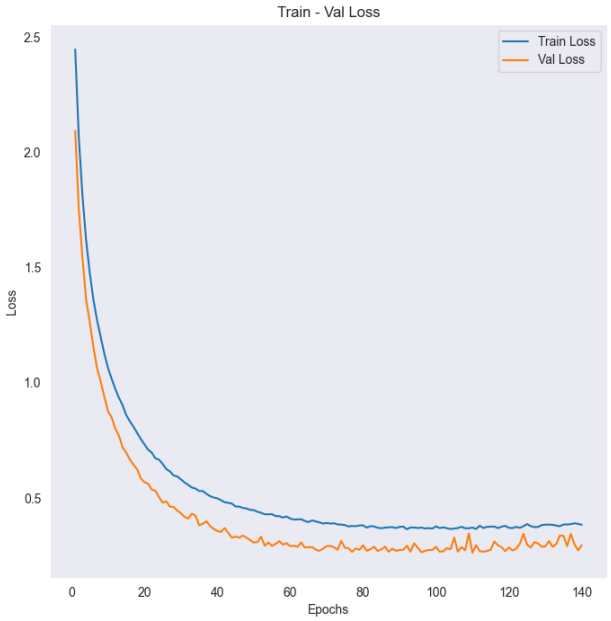

Figure 8: The training and validation loss of the MLP when trained only on Dioni.

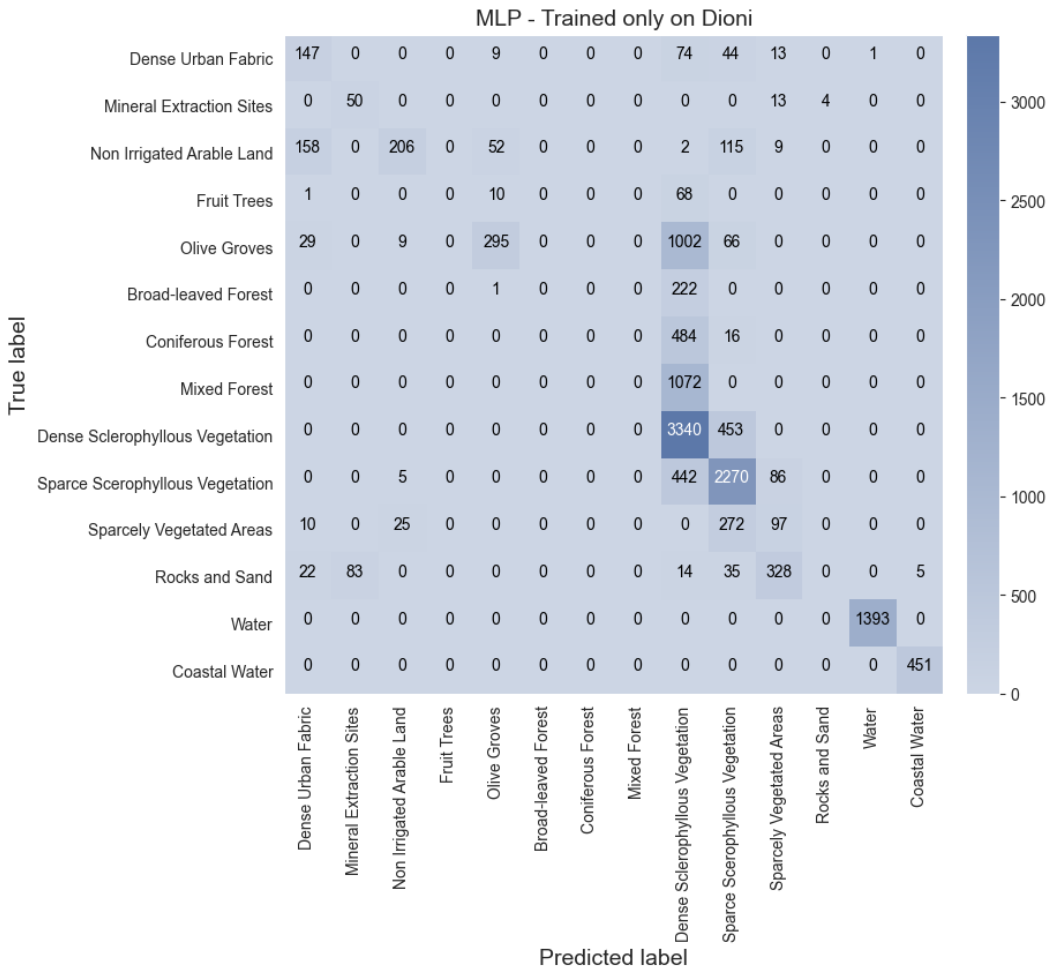

Figure 9: The Confusion matrix of MLP when trained on Dioni and validated on Loukia.

3 CNNs and Transfer learning

In this section we start developing more sophisticated architectures for the classification of the hyperspectral images, the so called Convolutional Neural Networks [6] (CNNs). CNNs are inspired by the structure of the visual system. In contrast to fully connected networks, CNNs make use of local connections to extract the contextual 2-D spatial features of images. In addition, network parameters can be significantly reduced via the weight-share mechanism [4]. In this section we construct a CNN model and developed two different approaches on how we preprocess the images. Furthermore, we use transfer learning to enhance the predictive power of our model. As we shall shortly see, the CNNs architecture provides a better solution to the classification problem of the hyperspectral images and a way to surpass the problem of having highly correlated train and test sets due to neighboring pixels.

3.1 The architecture of the model

The CNN consists of an initial double CONV-ReLU block. The first 2D-Convolutional operation accepts an input with 176 channels and outputs 32 channels. In all convolutional operations we use kernel size equal to 3, stride is 1 and padding is also 1. The series of operations of the first block are shown in Table 6.

Table 6: The series of operations of the first block.

Operations	**Stride**	**Padding**	**Dimensions**
CONV3-32	1	1	$H \times W \times 32$
ReLU	-	-	$H \times W \times 32$
CONV3-64	1	1	$H \times W \times 64$
ReLU	-	-	$H \times W \times 64$

The first CONV-ReLU block is followed by three CONV-ReLU blocks as described below.

Table 7: The series of operations for the three CONV-ReLU blocks.

Operations	**Stride**	**Padding**	**Dimensions**
CONV3-64	1	1	$H \times W \times 64$
ReLU	-	-	$H \times W \times 64$
CONV3-64	1	1	$H \times W \times 64$
ReLU	-	-	$H \times W \times 64$

Since our task at hand is a classification problem, at the end of the final Conv-ReLU block we place a small Feed forward neural network (FFNN). To connect the last convolutional operation with the FFNN we perform an adaptive average pooling to the output of the last Conv-ReLU

block to produce an image of size $64 \times 4 \times 4$. Then, we reshape the 3-dimensional matrix of size $64 \times 4 \times 4$ to obtain a vector of $64 \times 4 \times 4$ dimensions. In the following table we see the operations of the FFNN which consists of the classifier part.

Table 8: The series of operations of the FFNN part.

Operations	**Input dimensions**	**Output dimensions**
Linear	1024	128
ReLU	128	128
Linear	128	14

During the forward pass we also use skip connections to pass the information from the previous layers to deeper layers. After each double Conv-ReLU block we perform a max pooling with kernel size 2 and stride 2. In the following table we summarize the forward pass of an input image of size $C \times H \times W$ through the network.

Table 9: The forward pass in the CNN architecture.

Operations	**Input dimensions**	**Output dimensions**
Double Conv-ReLU	$C \times H \times W$	$32 \times H \times W$
Skip connection	$32 \times H \times W$	$32 \times H \times W$
Max pooling	$32 \times H \times W$	$32 \times H/2 \times W/2$
Double Conv-ReLU	$32 \times H/2 \times W/2$	$64 \times H/2 \times W/2$
Skip connection	$64 \times H/2 \times W/2$	$64 \times H/2 \times W/2$
Max pooling	$64 \times H/2 \times W/2$	$64 \times H/2^2 \times W/2^2$
Double Conv-ReLU	$64 \times H/2^2 \times W/2^2$	$64 \times H/2^2 \times W/2^2$
Adaptive max pool	$64 \times H/2^2 \times W/2^2$	$64 \times 4 \times 4$
Reshape	$64 \times 4 \times 4$	$64 \cdot 4 \cdot 4 = 1024$
Linear	1024	128
ReLU	128	128
Linear	128	14

The architecture is contained in *lib.py* in a class named *CNN*.

3.2 Overlapping patches

There are several ways to deal with the hyperspectral images in order to serve as input to the CNN model. In the overlapping patches approach, assuming our image has size $C \times H \times W$, given an odd number N, we produce from our initial image smaller images of size $C \times N \times N$. The features consists of the set containing all of the generated sub-images and the label for

each sub-image corresponds to the label of the pixel located at the center of the $N \times N$ square. By looking at Figures 1, 2 we see that there are pixels corresponding to undefined land areas. Therefore, to find the pixels with label number non zero we need to scan the whole image and select only the pixels for which the land area belongs to one of the fourteen categories. The aforementioned technique is called the patched based approach. In this section we allow overlapping patches, meaning that for two neighboring pixels in positions $(i, j), (i, j+1)$ we allow both squares

$$([i-(N-1)/2, i+(N-1)/2] \times [j-(N-1)/2, j+(N-1)/2]) \cap \mathbb{Z} \times \mathbb{Z}$$

and

$$([i-(N-1)/2, i+(N-1)/2] \times [j+1-(N-1)/2, j+1+(N-1)/2]) \cap \mathbb{Z} \times \mathbb{Z},$$

to be at the feature set. Observe that these two squares have

$$\frac{(N^2-N)}{N^2} \cdot 100\% = \left(1 - \frac{1}{N}\right) \cdot 100\%,$$

pixels in common. For e.g. for $N = 7$, these squares share the 85% of the pixels, for $N = 15$ the 93% and so on. The problem that arises at this point is the same as the one discussed in 2.1. For even smaller values of the length size N, the dependency between neighboring pixels is really high. This means that if we want to train the model by allowing overlapping images and predict on other images then we will probably get similar results as in the situation of 2.2.2, 2.3.3. Therefore, this approach is only useful when we want to annotate the training images, e.g. Dioni and Loukia in our case.

Therefore, in this subsection we construct as many as possible overlapping patches with length size $N = 15$ for both images of Dioni and Loukia. The challenging part at this point is to construct the class to handle the dataset. The class that inherits from PyTorch's Dataset class is located in *lib.py* under the name *patching_dataset*. It accepts the *path* where the *tif* file to be read is located, the name of the image, e.g. Dioni, Loukia, etc, the length size N of the patches (variable *patch_size*) and the variable *stride* corresponding to the minimum distance between the center pixels of each patch. For example, if *stride* = 1 we allow maximum overlapping and if *stride* = N we have no overlapping at all. Once an object of the class *patching_dataset* is initialized for a given image, e.g. Dioni, the function *find_centers* starts automatically to determine the pixels corresponding to nonzero labels that will serve as the centers for the patches of length size N. In the maximum overlapping case, where *stride* = 1 every pixel of the image will be examined. In the case where *stride* > 1 then only the pixels of the form $(k \cdot stride, m \cdot stride)$ for $k, m \in \mathbb{Z}_{\geq 0}$ will be examined. For the preprocessing part of the dataset, we use a normalization by dividing each channel-image by 10000. This operation is handled by the class *patching_dataset* which accepts a list of transformations to be applied as callable functions. The only transformation that we use at this point is the aforementioned normalization handled

by the function *normalize_patch* located in *lib.py*. A custom class named *prepare_datasets* takes as input the path of the two training images; Dioni and Loukia. For each image it creates the patches for a predefined length size N and combines them using the *ConcatDataset* from *torch.ultils.data* to form a single dataset of patches accompanied with their centered pixel labels. Furthermore, it creates the train/test and validation sets and returns them as output.

For the training part we use a length size $N = 15$ and *stride* = 1 allowing maximum overlapping between the images. The total number of patches obtained from the two images with length size $N = 15$ and *stride* = 1 are 32264. 30% serves as the test set and from the rest the 30% forms the validation set. The batch-size is equal to 32 and the learning rate is 10^{-4}. The maximum training epochs is 200 if no early stopping is encountered. The loss function is the cross entropy plus an ℓ_2-regularization term with $\lambda = 10^{-3}$. Below we see the classification report obtained from this approach.

Table 10: Classification report for the CNN using patches with maximum overlapping.

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Dense Urban Fabric	0.96	0.98	0.97	441
Mineral Extraction Sites	1.00	1.00	1.00	68
Non Irrigated Arable Land	0.99	0.97	0.98	324
Fruit Trees	0.91	0.87	0.89	47
Olive Groves	0.99	0.96	0.97	959
Broad-leaved Forest	0.92	0.81	0.86	70
Coniferous Forest	0.97	0.94	0.95	245
Mixed Forest	0.94	0.96	0.95	291
Dense Sclerophyllous Vegetation	0.98	0.99	0.98	2521
Sparce Scerophyllous Vegetation	0.98	0.99	0.98	2677
Sparcely Vegetated Areas	0.99	0.99	0.99	649
Rocks and Sand	1.00	0.99	1.00	277
Water	1.00	1.00	1.00	872
Coastal Water	1.00	1.00	1.00	239
Accuracy			0.98	9680
Macro Avg.	0.97	0.96	0.97	9680
Weighted Avg.	0.98	0.98	0.98	9680

As we can see from Table 10 the CNN achieves really high scores in all metrics and in all categories when it comes to the pixels originating from Loukia and Dioni. However, as we have already mentioned this approach is not the optimal one for making predictions on other images.

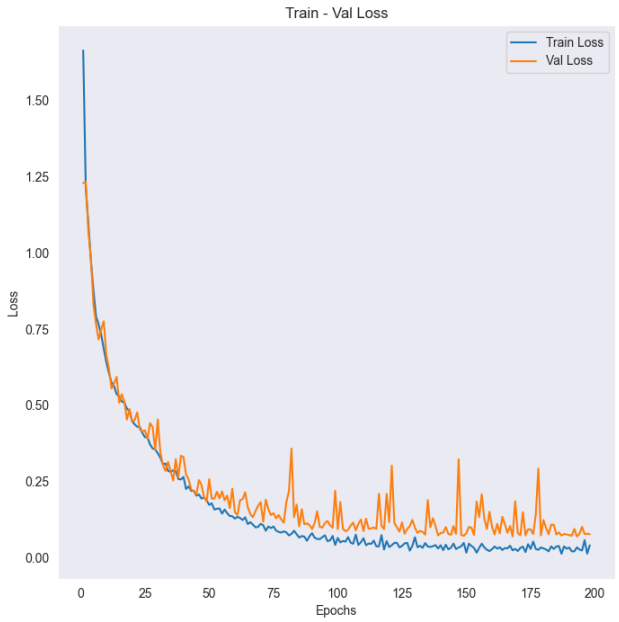

Figure 10: The training and validation loss of the CNN with maximum overlapping patches.

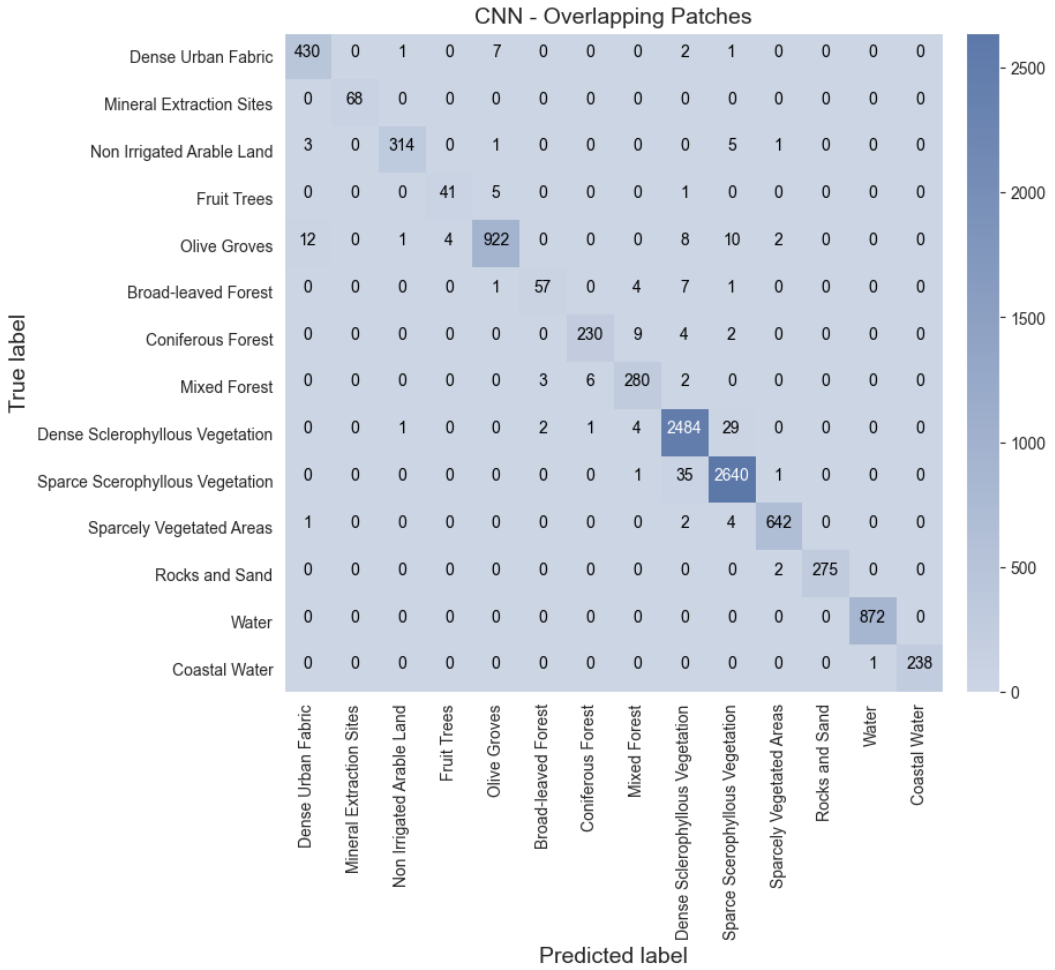

Figure 11: The Confusion matrix of CNN trained on maximum overlapping patches.

Since this approach achieves really high accuracy on the test set it can be used to annotate Dioni

and Loukia. In the figure below we see the predictions made by the CNN on the whole image of Dioni.

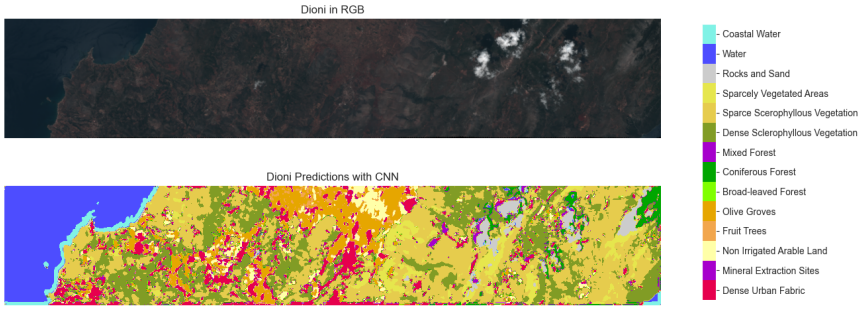

Figure 12: Predictions made by the CNN with maximum overlapping on Dioni.

3.3 The non-overlapping approach

In the non-overlapping approach we do not allow intersections between the patches. To achieve this, we simply choose the length size N of the patches and we set the *stride* argument in the *patching_dataset* class to be equal to $N-1$. One drawback of this approach is that we obtain a dataset with significantly reduced size as opposed to the case of the overlapping patches. Therefore, despite the fact this approach offers a logically correct solution to the task at hand, on the other hand, we cannot be sure if the model will be able to generalize in upcoming images with disproportionately larger samples than the training set. One way to tackle this problem is to use data augmentation techniques. For the data augmentation we use the *albumentations* library. For this case, and all the upcoming training procedures we use vertical and random rotate flips with probability 0.5 and one of elastic, grid distortion, optical distortion transformations with probabilities 0.5, 0.5 and 0.8, respectively. The list of transformations can be found in *main.py* in a variable named *transforms*. To have a larger dataset, we use $N = 7$ for the length size of the patches. To obtain non overlapping patches we set *stride* = 6. The resulting training, validation and test sets consist of 736, 316 and 452 images, respectively. For the training part, we use a batch-size equal to 32, and a learning rate equal to 10^{-4}. The loss function is the cross entropy plus an ℓ_2 regularization term with $\lambda = 10^{-4}$. We train the model for a maximum of 300 epochs if no early stopping is encountered. Since we keep a checkpoint for the model with the lowest validation error we do not need to worry about setting a low patience. Therefore, in this training we use a patience number equal to 120. The training of the CNN with no overlapping lasted for 1 min and 39.32 seconds. In Table 11 you can see the confusion matrix obtained from this technique on the test set and in Figure 13 the graph of the training/validation error for each epoch.

Table 11: Classification report for the CNN using patches with no overlapping.

Category	Precision	Recall	F1-Score	Support
Dense Urban Fabric	0.50	0.56	0.53	16
Mineral Extraction Sites	1.00	0.67	0.80	3
Non Irrigated Arable Land	0.75	0.52	0.62	23
Fruit Trees	0.33	0.14	0.20	7
Olive Groves	0.55	0.69	0.61	45
Broad-leaved Forest	0.60	0.33	0.43	9
Coniferous Forest	0.71	0.62	0.67	16
Mixed Forest	1.00	0.31	0.47	13
Dense Sclerophyllous Vegetation	0.75	0.81	0.78	114
Sparce Scerophyllous Vegetation	0.80	0.85	0.82	115
Sparcely Vegetated Areas	0.64	0.66	0.65	32
Rocks and Sand	0.86	0.71	0.77	17
Water	0.97	0.97	0.97	29
Coastal Water	0.92	0.92	0.92	13
Accuracy			0.74	452
Macro Avg.	0.74	0.63	0.66	452
Weighted Avg.	0.75	0.74	0.73	452

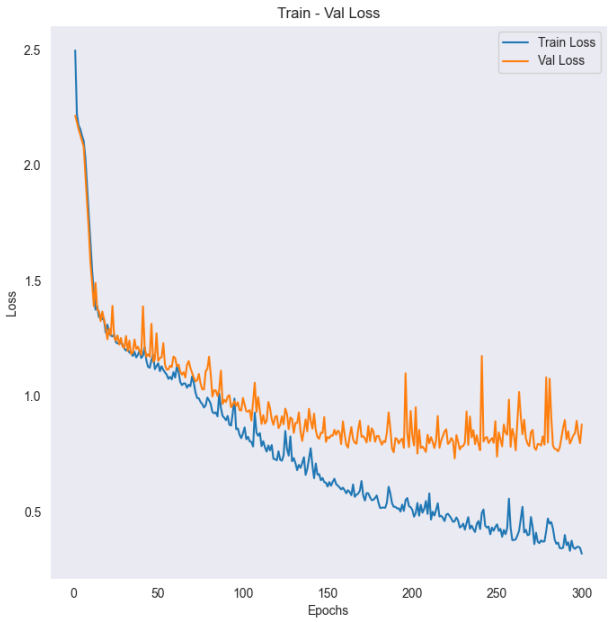

Figure 13: The training and validation loss of the CNN with non overlapping patches.

As we can see from the above results the overall accuracy of the model is 74%, which is significantly lower than the case of the overlapping patches. However, since we eliminated the dependency between the patches, we expect this model to behave in a similar manner to unseen images. The only problem with this approach that we need to worry about is low support of the test set in Table 11. As we can see the test set consists only of 452 images and as such we cannot be sure whether the model learnt certain classes or not. For example, we see that the land coverage by mineral extraction sites, urban fabric, non irrigated arable land, fruit trees, broad-leaved forest, coniferous forest, mixed forest, rocks and sand and water areas have support less than 30 samples. This means that we cannot be sure about the generalization power of the model on these classes. Of course, the data augmentation was utilized to enhance the training set and helped the model in order to distinguish these classes but still the size of the test set is relatively small to be certain about the generalization of the model. In Section 4 we will develop a model that deals with the high dependency of the neighboring pixels and results in a more reliable training set. We conclude this subsection with the confusion matrix obtained from the CNN on the test set.

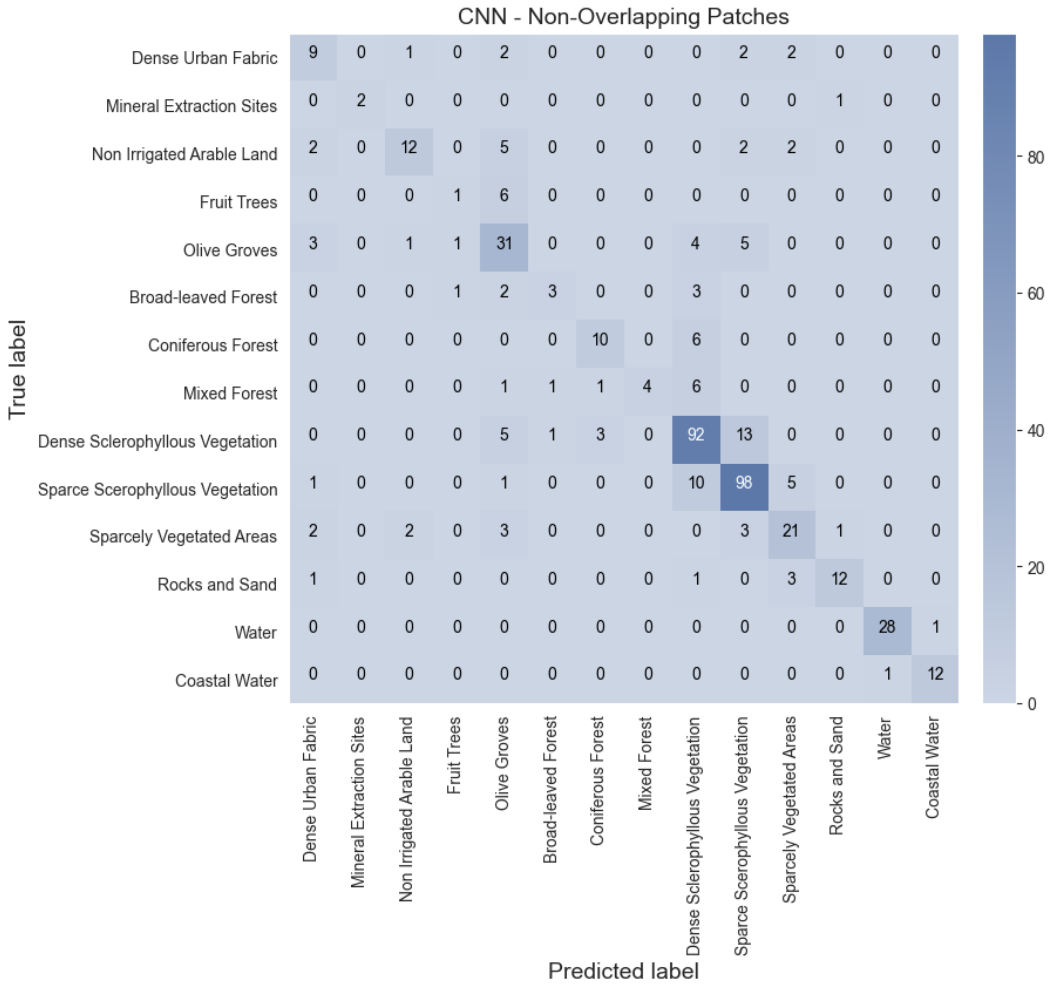

Figure 14: The Confusion matrix of CNN trained on non-overlapping patches.

3.4 Non-overlapping and transfer learning

In this subsection we try to enhance the predictive power of the model by using transfer learning. We use ResNet18 for the transfer learning with pretrained weights. We freeze the part that corresponds to ResNet18 and we add a small FFNN at the end consisting of linear mapping $\mathbb{R}^{256} \to \mathbb{R}^{128}$ followed by a ReLU activation function and another linear mapping $\mathbb{R}^{128} \to \mathbb{R}^{14}$. Since the architecture of ResNet18 is designed for 3-channeled images we keep only the RGB channels of the hyperspectral images. The architecture of the model is located in *lib.py* under the name *TransferResNet*. The training procedure is exactly the same as in 3.3. Below we see the resulting classification report, the confusion matrix and the graph of training/validation error for each epoch.

Table 12: Classification report for the CNN using patches with no overlapping.

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Dense Urban Fabric	0.59	0.50	0.54	20
Mineral Extraction Sites	1.00	1.00	1.00	6
Non Irrigated Arable Land	0.58	0.50	0.54	22
Fruit Trees	0.00	0.00	0.00	8
Olive Groves	0.65	0.63	0.64	62
Broad-leaved Forest	0.00	0.00	0.00	6
Coniferous Forest	0.59	0.48	0.53	21
Mixed Forest	0.67	0.47	0.55	17
Dense Sclerophyllous Vegetation	0.59	0.80	0.68	107
Sparce Scerophyllous Vegetation	0.71	0.70	0.71	100
Sparcely Vegetated Areas	0.68	0.57	0.62	23
Rocks and Sand	1.00	0.82	0.90	17
Water	1.00	1.00	1.00	27
Coastal Water	1.00	1.00	1.00	16
Accuracy			0.69	452
Macro Avg.	0.65	0.60	0.62	452
Weighted Avg.	0.68	0.69	0.67	452

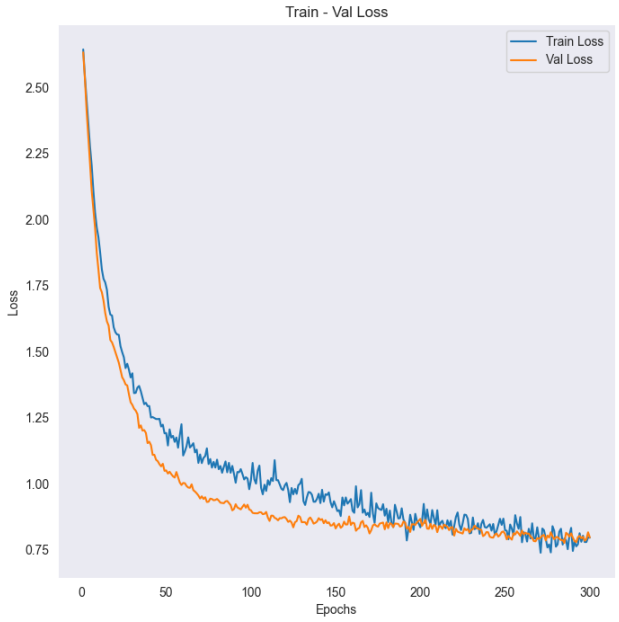

Figure 15: The training and validation loss of CNN using transfer learning and non overlapping patches.

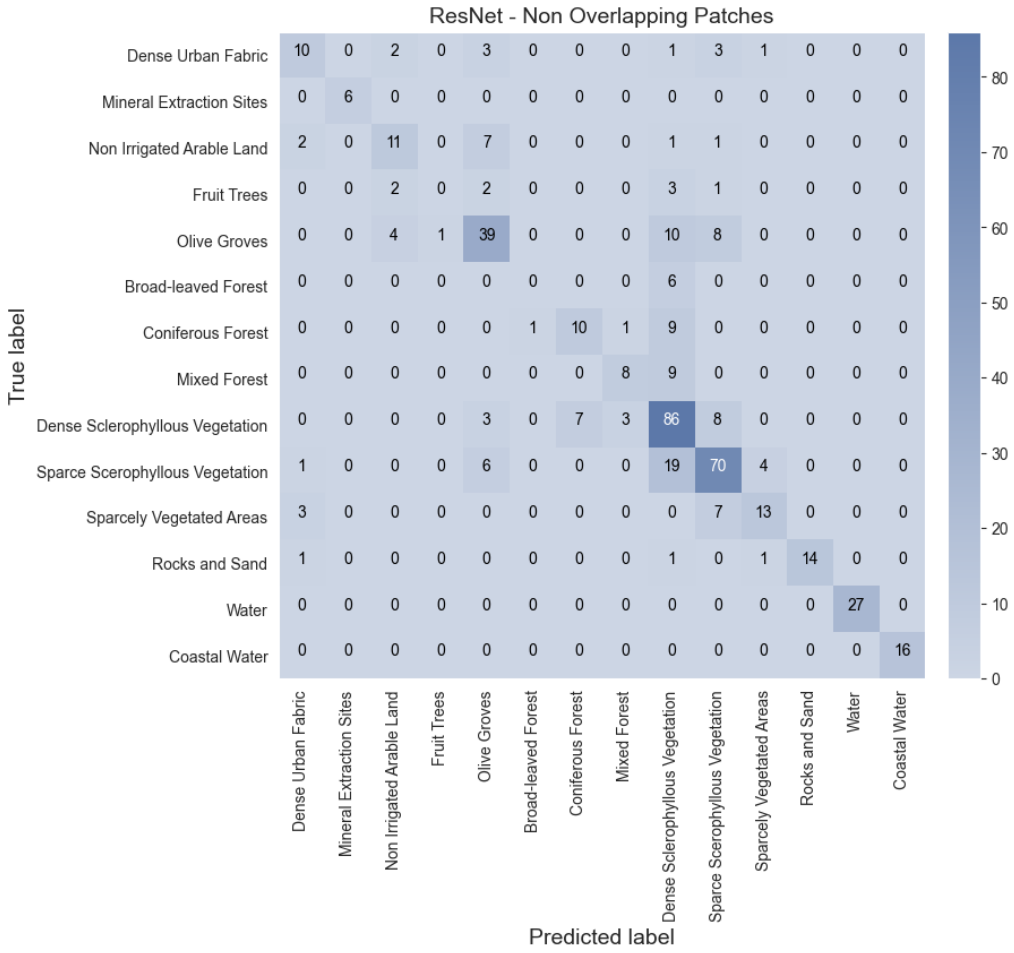

Figure 16: The Confusion matrix of CNN using transfer learning and non-overlapping patches.

4 The UNet

In this section we adopt a different approach to the problem. As opposed to previous sections where we used the pixel and patched based approaches in this section we classify the whole image at once. In particular, we partition each training image (Dioni and Loukia) into squared images of size $N \times N$. Therefore, the features consists of a finite sequence $(X_i)_{i=1}^M$ of hyper-spectral images with size $179 \times N \times N$. The benefit of this approach is two fold; on the one hand since the sequence $(X_i)_{i=1}^M$ is a partition of the two training images we avoid the high-dependency between the samples and on the other hand we do not need to worry about the undefined pixels of the images X_i. This is because we can use a mask to exclude these pixels from the calculation of the loss function during the training procedure. In the next two subsections we analyze in detail the architecture of the model and the tools that we developed in order to create the sequence of feature images $(X_i)_{i=1}^M$.

4.1 The Architecture of the model

The architecture of the model that we use in this section is essentially the same as the UNet model which was first presented in [7]. In Figure 17 we see the architecture of the model used in [7].

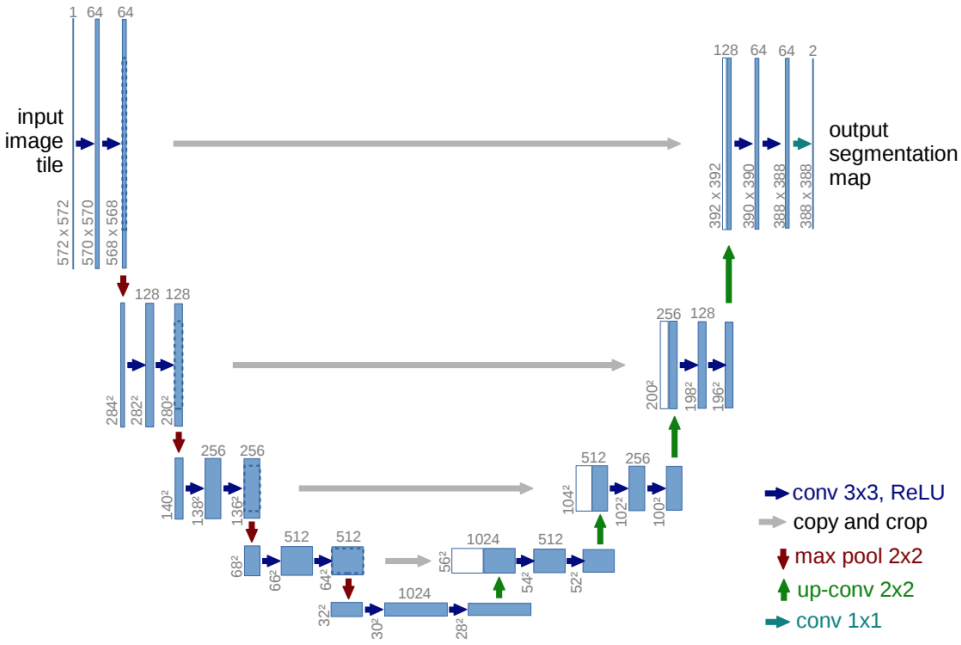

Figure 17: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

In our implementation we slightly modify the above architecture in order to adjust it to our task at hand. To begin with, in each conv-ReLU (blue arrows in Fig. 17) operation we keep the height and width of the images intact. In particular, the first double conv-ReLU operation for an input image of size $179 \times 32 \times 32$ can be described by the following series of operations shown in Table 13.

Table 13: The series of operations of double conv-ReLU part.

Operation	**kernel size**	**(stride, padding)**	**Input dim.**	**Output dim.**
Conv2D	3	$(1,1)$	$179 \times 32 \times 32$	$128 \times 32 \times 32$
BatchNorm2D	-	-	$179 \times 32 \times 32$	$128 \times 32 \times 32$
ReLU	-	-	$179 \times 32 \times 32$	$128 \times 32 \times 32$
Conv2D	3	(1,1)	$128 \times 32 \times 32$	$128 \times 32 \times 32$
BatchNorm2D	-	-	$128 \times 32 \times 32$	$128 \times 32 \times 32$
ReLU	-	-	$128 \times 32 \times 32$	$128 \times 32 \times 32$

As you can see from the above table there are several differences in our implementation as opposed to the architecture shown in Figure 17. Since our images have 179 images we do not want to drop from 179 channels to 64 and hence in the first double conv-ReLU operation we start by 179 channels and drop to 128 instead. Furthermore, in between each conv-ReLU operation we use a 2D batch normalization with respect to the channel dimension. In addition, instead of 4 max pooling and up-conv operations we use 3 and as such we have 6 double conv-ReLU operations in total. This is because we do not want to shrink our images a lot in order to maintain as much "local" information as possible. For example, if we were to use 4 max pooling operations then the height and width of our input images would drop from 32 to 2 resulting in 2×2 images. By using 3 max pooling operations we drop our images from 32×32 to 4×4 images. In our case, since the spatial resolution of our images is $30m$ then 4×4 images corresponds to $120m^2$ of area surface. Summarizing all the above, the number of channels in each conv-ReLU operation can be described by the following sequence:

$$179 \to 128 \to 256 \to 512 \to 1024 \to 512 \to 256 \to 128 \to 14,$$

where the number 14 corresponds to the total number of classes (land uses). The full architecture of the model can be found in *lib.py* file under the name *UNet*. In the following table we summarize the forward pass for an input image X of size $179 \times 32 \times 32$. By $X \oplus Y$ we denote the concatenation of the images X, Y of size $C \times H \times W$ with respect to the first dimension. Therefore, the resulting image $X \oplus Y$ has size $2C \times H \times W$.

Table 14: The forward pass of UNet for an input image X with size $179 \times 32 \times 32$.

Operation	**Input dim.**	**Output dim.**	**Output Img.**
DoubleConvReLU	$179 \times 32 \times 32$	$128 \times 32 \times 32$	X_1
MaxPool2D	$128 \times 32 \times 32$	$128 \times 16 \times 16$	X_2
DoubleConvReLU	$128 \times 16 \times 16$	$256 \times 16 \times 16$	X_3
MaxPool2D	$256 \times 16 \times 16$	$256 \times 8 \times 8$	X_4
DoubleConvReLU	$256 \times 8 \times 8$	$512 \times 8 \times 8$	X_5
MaxPool2D	$512 \times 8 \times 8$	$512 \times 4 \times 4$	X_6
DoubleConvReLU	$512 \times 4 \times 4$	$1024 \times 4 \times 4$	X_7
ConvTranspose2D	$1024 \times 4 \times 4$	$512 \times 8 \times 8$	X_8
Skip connection	$512 \times 8 \times 8$	$1024 \times 8 \times 8$	$X_9 = X_8 \oplus X_5$
DoubleConvReLU	$1024 \times 8 \times 8$	$512 \times 8 \times 8$	X_{10}
ConvTranspose2D	$512 \times 8 \times 8$	$256 \times 16 \times 16$	X_{11}
Skip connection	$256 \times 16 \times 16$	$512 \times 16 \times 16$	$X_{12} = X_{11} \oplus X_3$
DoubleConvReLU	$512 \times 16 \times 16$	$256 \times 16 \times 16$	X_{13}
ConvTranspose2D	$256 \times 16 \times 16$	$128 \times 32 \times 32$	X_{14}
Skip connection	$128 \times 32 \times 32$	$256 \times 32 \times 32$	$X_{15} = X_{14} \oplus X_1$
DoubleConvReLU	$256 \times 32 \times 32$	$128 \times 32 \times 32$	X_{16}
OutputConv	$128 \times 32 \times 32$	$14 \times 32 \times 32$	X_{17}

4.2 Transforming the images

As soon as we have constructed the architecture of the model the next step is to partition Dioni and Loukia into the finite sequence of images $(X_i)_{i=1}^M$ with sizes $179 \times N \times N$. To achieve this, we create a custom class named *cropped_dataset* located in *lib.py*. The class inherits from the *Dataset* class of PyTorch. The arguments are the path where the images are located, the name of the image to be read and the length size N which corresponds to the size of the square images. Upon initializing an instance of a *cropped_dataset* object the built-in function *find_up_left_corners* finds the coordinates of the up-left pixels of the square images $(X_i)_{i=1}^M$. The function proceeds as follows: assuming the input image has size $C \times H \times W$, at first, an empty list named *up_left_corners* is initialized. Then, in a double for-loop the function *find_up_left_corners* appends all coordinates of the form $(k \cdot N, m \cdot N)$, where $k, m \in \mathbb{Z}_{\geq 0}$ satisfying $0 \leq k \leq \left[\frac{H-1}{N}\right], 0 \leq m \leq \left[\frac{W-1}{N}\right]$. The *len* method returns the length of the list *up_left_corners* and the *getitem* method for a given index i finds the ith element of the list *up_left_corners* which corresponds to the ith up-left corner with coordinates (k,l) and returns a slice of the input image X with size $C \times H \times W$ described by $X[:, k : k+N, l : l+N]$. This

approach allows to obtain the sequence of square images $(X_i)_{i=1}^{M}$ "on the fly" during the training and testing procedures without constructing another dataset from scratch. Finally, the cropped image $X[:, k:k+N, l:l+N]$ is normalized by dividing each entry by 10000.

4.3 Training and results

For the training part we use the same *training_loop* that we used in the previous sections as well. The only difference at this point is that we need to exclude the pixels that correspond to undefined areas from the calculation of the loss function. We simply do this by using the *ignore_index* argument in PyTorch's cross entropy loss function. For technical reasons we shift the classes by subtracting 1 from the label image. Therefore, the pixels corresponding to undefined areas have value -1, the urban areas 0, the mineral extraction sites 1 and so on. Apart from aforementioned normalization on the cropped images we use data augmentation. The transformations are the same with the ones used in 3.3. We first partition Dioni and Loukia separately to obtain two finite sequences $(X_i)_{i=1}^{M}$, $(X_i')_{i=1}^{K}$ of cropped images with sizes $179 \times 32 \times 32$ and then combine these two sequences to obtain the training set. The train set consists of 12300 cropped images. After splitting this set to train/validation and test sets we obtain mpla mpla samples for each set respectively. The batch-size is equal to 16, the learning rate is set to 10^{-5}. An additional ℓ_2 regularization term is added to the cross entropy loss function with $\lambda = 10^{-1}$ to combat overfitting. We train for a maximum of 500 epochs if no early stopping sign is encountered. The number of patience is equal to 120. The training lasted for 27 mins and 41.14 seconds, in the figure below you can see the graph of the training/validation error for each epoch.

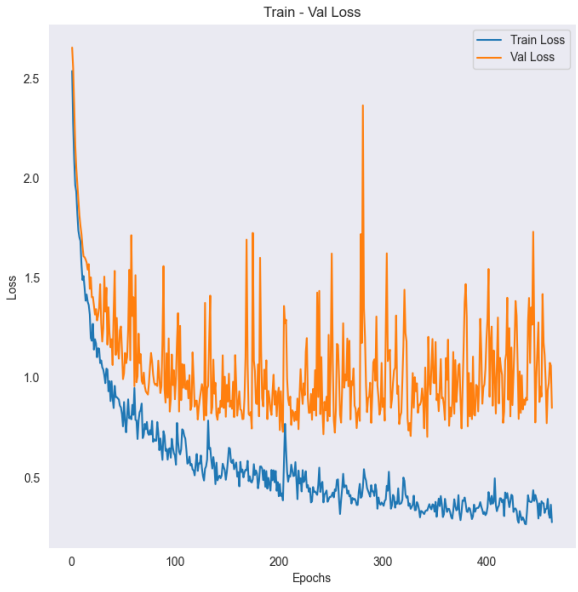

Figure 18: The training and validation loss of UNet on 32×32 cropped-sized images.

Below we see the classification report obtained from the test set.

Table 15: Classification report for the UNet with 32×32 cropped-sized images.

Category	**Precision**	**Recall**	**F1-Score**	**Support**
Dense Urban Fabric	0.68	0.71	0.69	271
Mineral Extraction Sites	1.00	0.12	0.22	140
Non Irrigated Arable Land	0.87	0.69	0.77	450
Fruit Trees	0.00	0.00	0.00	27
Olive Groves	0.89	0.88	0.89	589
Broad-leaved Forest	0.14	0.83	0.24	18
Coniferous Forest	0.90	0.69	0.78	144
Mixed Forest	0.87	0.50	0.63	303
Dense Sclerophyllous Vegetation	0.83	0.88	0.86	1370
Sparce Scerophyllous Vegetation	0.85	0.89	0.87	2449
Sparcely Vegetated Areas	0.54	0.75	0.62	406
Rocks and Sand	0.84	0.74	0.78	511
Water	1.00	1.00	1.00	421
Coastal Water	1.00	1.00	1.00	354
Accuracy			0.82	7453
Macro Avg.	0.74	0.69	0.67	7453
Weighted Avg.	0.84	0.82	0.82	7453

From the above table we see that the overall accuracy of the model is 82% which is 8% higher than the CNN when trained on non-overlapping patches (see Table 11). Moreover, as opposed to the CNN the weighted average of the UNet is significantly higher in all three metrics (precision, recall, f1-score) and almost the same with respect to the macro average. Another advantage of this approach over the CNN with the non-overlapping patches is that this model is tested on a significantly larger test set as opposed to the former. Therefore, we conclude that this approach finds a balance between the two major problems: 1) the high dependency of the overlapping patches and 2) the very few training/test samples. In Figure 19 we can see the corresponding confusion matrix.

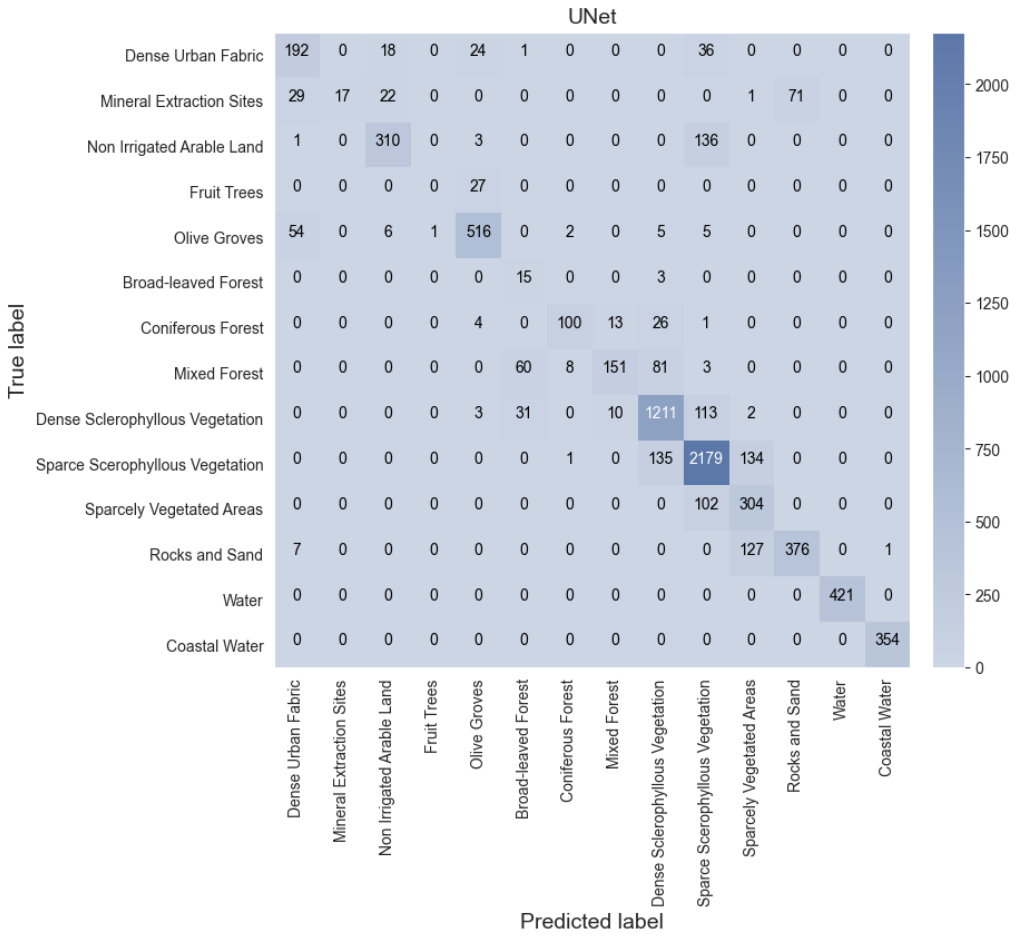

Figure 19: The Confusion matrix of UNet with 32×32 cropped-sized images.

5 Results and predictions

In this section we summarize the results of the developed models and the different approaches adopted for the classification of the hyperspectral images. Our end goal is to compare these models and find the most suitable one in order to make predictions to the three satellite images of Erato, Kirki and Nefeli. In addition, we thoroughly elaborate on our final choice and we present the annotated images of Erato, Kirki and Nefeli based on our final choice.

5.1 Comparing and choosing the final model

Since during the training of the MLP and the Random forest we allowed neighboring pixels to be in the training and the test set as well we can assume that these two models follow a similar approach to the CNN when trained with overlapping patches. From these three different approaches the CNN by far achieves the best scores in all metrics (see Table 11). Similarly, from the models that complied to the non-overlapping approach (UNet and CNN) the UNet is a better candidate from these according to the results obtained in 3.3, 4.3. Therefore, the biggest dilemma about which model to choose comes down to which approach is better: the overlapping

approach or the non-overlapping approach. As we have already stated above, the high scores achieved by the CNN with maximum overlapping patches are misleading since there is leakage of information on the test set. Therefore, we cannot be sure about the behavior of this model on Erato, Kirki and Nefeli. For example, when we trained the MLP and the Random forest only on Dioni and validated on Loukia the score on Loukia was approximately 60% while the score on Dioni was 90%. Choosing the CNN model with the maximum overlapping approach might lead to a similar outcome and hence for the final model we choose the UNet. For completion we provide the predictions made by the CNN with the maximum overlapping approach on Erato, Kirki and Nefeli in the following images.

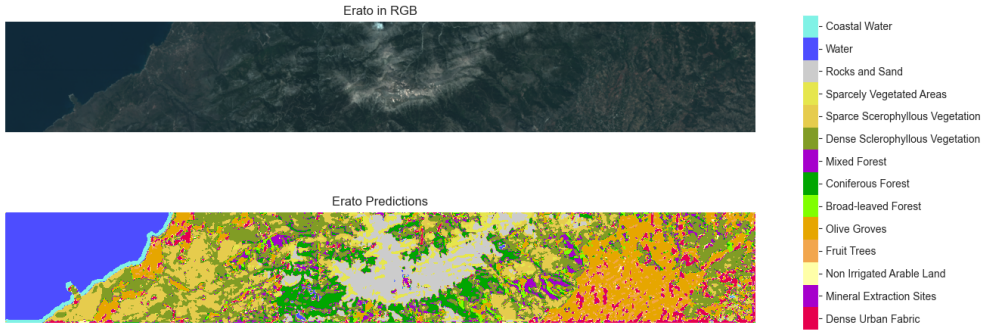

Figure 20: Predictions made by the CNN with maximum overlapping on Erato.

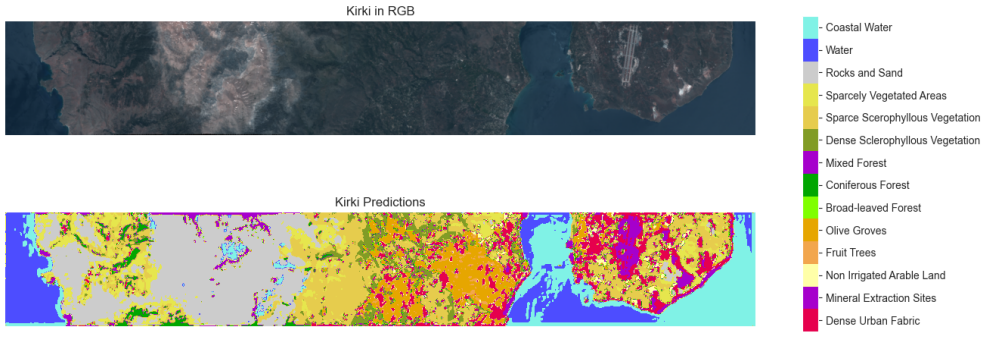

Figure 21: Predictions made by the CNN with maximum overlapping on Kirki.

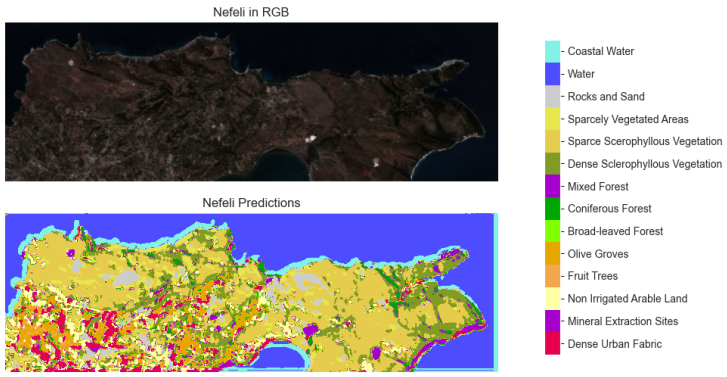

Figure 22: Predictions made by the CNN with maximum overlapping on Nefeli.

5.2 Predictions on Erato, Kirki and Nefeli

In this subsection we present the prediction by the UNet on Erato, Kirki and Nefeli. The last and quite challenging part is to construct the annotated images for the three aforementioned satellite images. Since the model has trained on $179 \times 32 \times 32$ images we must first partition the validation images into a sequence of sub-images of size $179 \times 32 \times 32$. The problem that arises at this point is that the height and width of the images its not divisible by 32. For example, Erato has size 241×1632 and 241 is not divisible by 32. Our approach at this point is to pad with zeros the remaining parts of the images in order to obtain the cropped 32×32 images. To handle this procedure we construct a custom class named *validation_dataset*. This class reads any of the three validation images and partitions the image into a sequence $(X_i)_{i=1}^{M}$ of $179 \times 32 \times 32$ sub-images by using zero-padding when needed. Having created the dataset class we create a function named *predict_image* to handle the prediction-loop. This function creates an array of zeros with height and width equal to the validation image. For example, if the validation image is Erato then this function will initialize an array of zeros with size 241×1632. Then, passing one by one the sub-images through the UNet we obtain the predictions and fill the corresponding pixels into the zero array to obtain the final prediction. We conclude this section by presenting the predictions obtained by UNet.

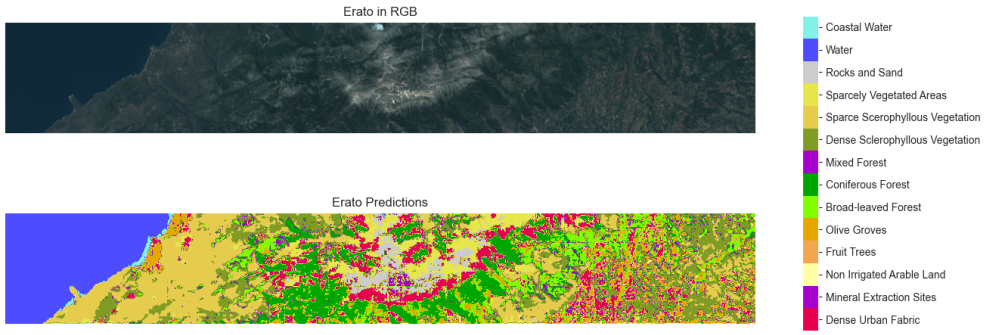

Figure 23: Predictions made by UNet on Erato.

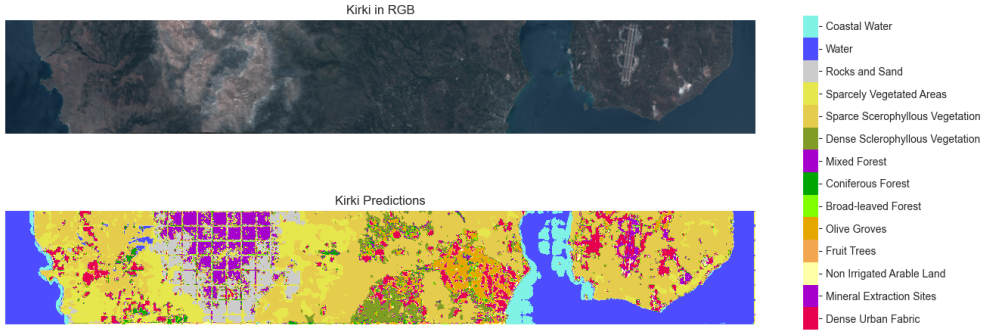

Figure 24: Predictions made by UNet on Kirki.

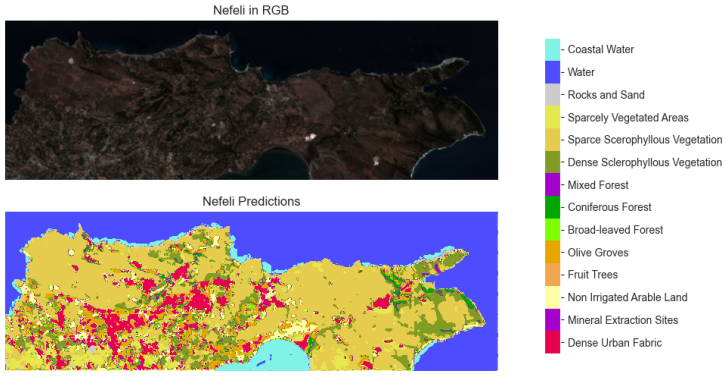

Figure 25: Predictions made by UNet on Nefeli.

References

[1] LE Christovam et al. “Land use and land cover classification using hyperspectral imagery: evaluating the performance of spectral angle mapper, support vector machine and random forest.” In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* (2019).

[2] Pedram Ghamisi et al. “Advanced spectral classifiers for hyperspectral images: A review”. In: *IEEE Geoscience and Remote Sensing Magazine* 5.1 (2017), pp. 8–32.

[3] K Karantzalos et al. “HyRANK hyperspectral satellite dataset I”. In: *Version v001). doi: https://doi. org/10.5281/zenodo* 1222202 (2018).

[4] Shutao Li et al. “Deep learning for hyperspectral image classification: An overview”. In: *IEEE Transactions on Geoscience and Remote Sensing* 57.9 (2019), pp. 6690–6709.

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

[6] Keiron O’Shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.