Master's in Data Science and Advanced Analytics

Deep Learning

2021/2022

Lightnet for Land Cover and Use Classification

Group 13

Diogo Papeleira (20180416)

Henrique Barreiros (20211380)

Miguel Bernardo (20210580)

Simone Covino (20211894)

Introduction

In our project we address two problems regarding computer vision. The first is the development of a convolutional neural network model able to accurately classify land use and land cover of satellite images. The second is the improvement in terms of robustness and predictive performance of the developed model when using small datasets. Therefore, the goal is to develop a model that can be trained quickly and continuously with publicly available images from the Sentinel satellite of the Copernicus program.

Regarding the former task, our main reference is the paper “EuroSAT” by Helber, Bischke, Dengel, and Borth. [1] The authors propose a thorough explanation on the dataset acquisition and creation and on the importance of land use and land cover classification, which automatically provides labels describing the represented physical land type and usage. Moreover, the paper provides the benchmarks for the dataset used in this project. In this framework, our goal is to build a “light” classification network able to get close to the provided benchmarks in terms of accuracy, and also to take into account its training time.

Considering the latter task, we follow the main idea of the paper “A Close Look at Deep Learning with Small Data'' by Brigato and Locchi [2] which underlines the problem of training convolutional neural network models when big datasets are not available. In this context, we aim to improve the previously built model and to obtain a simple convolutional neural network able to output good predictive accuracy even when it’s trained with a small number of images in a very short time.

The content of this report is organized as follows: in the data section we describe the used dataset, discuss its features, and how it has been preprocessed; in the benchmarking section, we reproduce the EuroSAT paper benchmarks and set a baseline for our models; in the modeling section, the main tasks are pursued - the model-building steps are analyzed and the obtained results are discussed; in the prediction section, we analyze the quality of the predictions of the best model; lastly, the main results are summed up in the conclusion section. Moreover, the reader can consult the reference list of the models in the appendices section or the source code to see the full specification of each cited model.

Data

One of the main contributions of the EuroSAT paper is the introduction of the first large-scale patch-based land use and land cover classification dataset based on Sentinel-2A satellite images, provided by the European’s Space Agency’s (ESA) Copernicus program. The authors underline how the classification of remotely sensed images is a challenging task and this dataset, in which every image is labeled and geo-referenced, represents a unicum in this particular framework. In fact, this dataset stands out in terms of the resolution of the images which opens to numerous real-world applications and the reasonable number of images per class it provides. Consequently, the authors provide a free satellite images dataset accessible to various earth observation applications. This dataset contains 10 divergent balanced classes with 2000-3000 images per class, containing a grand total of 27000 images, each measuring 64 per 64 pixels. The included land use classes include: Annual Crop, Permanent Crop, Pastures, Industrial Buildings, Residential

Buildings, River, SeaLake, Herbaceous Vegetation, Highway and Forest (fig. 1). It should be noted how different water bodies appear in the class River and in the class SeaLake.

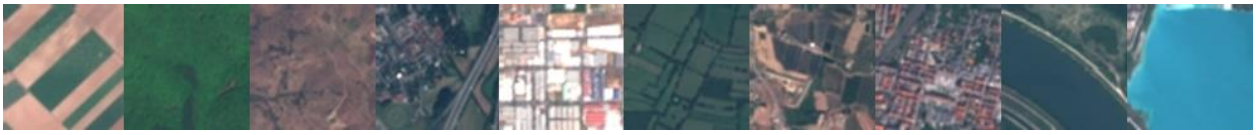

Figure 1 - ordered representation of each class of the EuroSAT paper dataset

To preprocess the images, we have created the data_gen function that takes the images of each class, each saved on their respective folders, and splits them into training (60%), validation (20%) and test (20%) sets respectively. Each image was converted into a tensor of dimension (64, 64, 3) and rescaled by multiplying each value by 1 / 255. For each class of the dataset, each defined proportion of images to be split were merged together into a 4D tensor of (*n*, 64, 64, 3) dimension, where *n* represents the number of samples of each split. Additionally, the respective labels of each image were converted into binary encoded vectors of size (10), merged together into a tensor of (*n*, 10) dimension, where *n* represents the number of samples of each split. The output tensors were saved in pickle format to be readily loaded and used throughout the modeling part of the project.

Benchmarking

Considering the EuroSat paper, the ResNet-50 pre-trained on the ILSVRC-2012 dataset was among the best networks in terms of accuracy to predict our dataset. For that reason, we have decided to use it to establish a baseline comparison model for our experiments. We replaced the output layer by a fully connected layer of 128 units followed by an output layer of 10 units, which uses a softmax activation function to predict the 10 classes of the problem, then the network was initialized with the ImageNet weights and retrained on our augmented dataset (0.2 of zoom and sheer range) for 120 epochs (500 steps per epoch), using stochastic gradient descent (SGD) optimizer. The hyperparameters controlling the learning process, such as the learning rate, the momentum, and the learning decay are set to the values reported in the paper. The network took 41 min and 44 seconds to train, scoring 0.9576 and 0.9556 of validation and test accuracy, respectively (table 1 - appendices).

We also needed a baseline shallow CNN model to be improved upon throughout the modeling phase of our project. In that sense, we have used the simple architecture reported in paper. The model consists on three consecutive pairs of a convolution layer with 64 filters (3 x 3 size) and a max pooling layer (4 x 4 size and 2 x 2 strides), followed by a fully connected layer of 128 units and an output layer of 10 units, which also uses a softmax activation function to predict the 10 classes of the problem. The model was trained with augmented data in 14 minutes and 20 seconds, scoring 0.9119 and 0.9033 of validation and test accuracy, respectively.

Furthermore, the plots showing the training and validation accuracy and loss at each epoch display that there's no evidence of overfitting for both models (fig. 2 and 3 - appendices). However, it's interesting to note that the shallow CNN model is able to achieve a good performance with such a simple architecture and that the ResNet-50 only outperforms the shallow network because it has a better initialization and can extract more high-level features due to its structure.

Modeling

As the benchmarks show (table 1 - appendices), even a shallow CNN with a relatively simple structure can achieve good results in terms of accuracy, performing closely the ResNet-50 but with around 1 / 3 of the training time. Our model building steps and choices take this aspect into account as well as our objective of creating a well-performing model simple enough to be retrained quickly with less input data.

Throughout the model building phase, our approach consists in comparing the baseline accuracy and training time of the shallow CNN with the considered model (reference list in the appendices). The reasoning consists in improving the models starting from a relatively simple architecture and adding complexity by introducing one modification at a time, while keeping track of overfitting by analyzing the loss and accuracy during training and validation of the model. Hence, we can correctly evaluate the effect that each modification has over the performance of the model and properly direct the following modification to achieve the desired outcome. [3]

Although the baseline model doesn't overfit the training data, two main reasons lead us to add a dropout layer to the model. The first one, as suggested by the literature [4], is related to the use of data augmentation, which often comes along with the regularizing effect of dropout. Dropout accelerates convergence and improves the network's generalization ability. The second one, is connected with the second task of our project, because there's a higher risk of overfitting when we start to use smaller datasets to train the model, consequently we aim to build a more robust model which doesn't overfit even in "extreme" cases. Therefore, defining the structure of Model 1, we added a dropout layer (0.5 rate) before the output layer of the baseline shallow CNN and reduced the number of filters to 32. The network was trained in 15 minutes and 54 seconds, scoring 0.8701 and 0.8764 of validation and test accuracy respectively.

Convolution layers don't need to be always followed by pooling layers, it's possible to stack convolution layers together to allow the network to extract high-level features from the input images (not just edges and corners). In that sense, Model 2 consists of two consecutive blocks of two stacked convolution layers of 32 and 64 filters respectively, followed by a max pooling layer. The remaining layers are the same as Model 1. As seen by the results (table 1 - appendices), Model 2 outperforms Model 1 both in terms of accuracy and training time, thus suggesting that higher-level features influence more the performance than modifications in the number of filters.

To improve further the results of Model 2, we added two dropout layers (0.25 rate) after each max pooling layer. Model 3 obtained a higher accuracy (table 1 - appendices) with only an increase of 13 seconds of training time. The result suggests that the dropout layers control some overfitting of the stacked convolution layers, hence making the model more robust to be used with small datasets.

Considering the results obtained by Model 4 and 5, it's clear that Model 3 is the best one for the first task that we defined previously since it has the best balance between accuracy and training time. In particular, Model 4 uses batch normalization layers instead of dropout layers in an effort to decrease training time. Even though the technique is effective to reduce internal covariate shift and speed up convergence in deeper networks, in simpler architectures like Model 4 is not a wise choice, as shown by the obtained results (table 1 - appendices). Furthermore, because dropout regularization is generally not advised to be used with batch

normalization [5], this network loses the desired robustness of Model 3. On the other hand, Model 5 has a fully connected layer with the number of units reduced to 64. Despite a minimal decrease of training time, around 10 seconds, the accuracy of the model drops 2% (table 1 - appendices).

Considering the second task of our project, we set out to develop a robust model trained with a smaller number of images. In that sense, we have created a function to generate a subset of the training set by randomly sampling without replacement a given proportion of images per each class, thereby maintaining within class variation. Moreover, by using the training set for resampling, we make sure there's no common observations between the training, validation and testing sets.

As a starting point, we selected 30% of the training observations for each class, thus obtaining a training set of 4860 observations (i.e. less than 500 observations for each class). Then, Model 6, which has the same structure as Model 3, was trained with the resampled training set, adjusting the number of steps per epoch accordingly. Following the mentioned approach to keep track of overfitting, we observed that Model 6 is robust enough as to not overfit this smaller dataset. However, the performance in terms of accuracy is not satisfying (table 1 - appendices). Even worse results are obtained with Model 7 when dropout rate is increased in all layers to 0.5 and 0.7 respectively.

When working with such small datasets, there's a trade-off between accuracy and training time. Therefore, with Model 8 and 9, we tried to find the optimal combination of the number of steps per epoch and batch size. Considering the obtained results, we can say that the goal of the second task is reached with Model 8. This model was trained in only 9 minutes (6 minutes less than Model 3), with less than 500 images per class, and it's able to classify the test set with almost 85% accuracy (table 1 - appendices). Moreover, as can be seen from estimation of Model 10, adding more complexity by increasing the number of filters, gives a minimal performance increase in terms of accuracy at the cost of greatly increasing the training time.

Lastely, we decided to analyze the robustness of Model 8 when the input dataset is reduced even more. Table 2 (appendices) shows the obtained results when the training set is reduced to 40, 30, 20 and 10% of the training observations of each class. We highlight that the robustness of the model is undoubtedly remarkable, even with around 150 samples per class, the model is able to classify the test with 77% accuracy.

Prediction

To test the quality of the predictions given by our best model, we removed the fully connected layer of Model 3 to maintain the spatial information contained in the output of the last convolution layer and added a global max pooling layer, to prevent the network from overfitting, connected to a dense layer with the same number of units as classes. [6] Thus, it was possible to generate class activation maps (CAM) to understand what are the portions of the images that most contribute to the output classification. As shown in the confusion matrix (fig. 4 - appendices), Model 3 struggles to predict the River class the most, confusing the class with mostly Pasture (3%) and Highway (8%). One CAM example (fig. 5) reveals that the areas surrounding the river are confusing the network, and therefore have bigger weight in the activation. Nevertheless, the comparision of the confusion matrices of Model 3 and the baseline shallow CNN (fig. 6 - appendices), show that Model 3 has the error more evenly distributed throughout all classes, thereby making this model more robust to error.

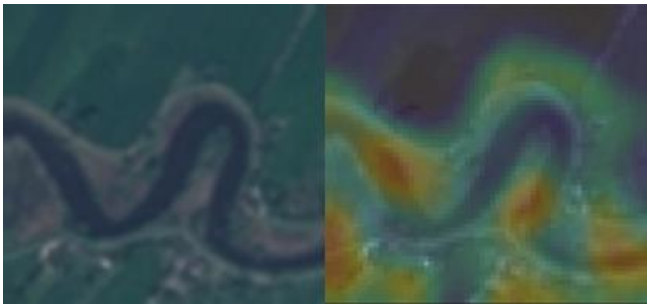

Figure 5 - River class picture (left) and its class activation map (right)

Conclusions

Considering the first task of our project, we successfully developed a convolutional neural network capable of accurately classifying land use and land cover of satellite images. The proposed Model 3 outperformed the baseline shallow CNN network with almost the same training time. Moreover, the use of stacked convolution layers was detrimental for the success of this network since the extraction of high-level features is more important than any other hyperparameter that we have tried to change, like the number of units or the number of layers. Dropout regularization allowed the network to avoid overfitting when extracting high-level features, thus making the network robust enough to be trained with smaller datasets.

As for the second task of the project, we were able to adapt Model 3 parameters to produce a model capable of training with a very reduced dataset. In that sense, Model 8 which takes the advantage of using more steps per epoch to offset the reduced number of examples, was trained in only 9 minutes and predicted the test set with 85% accuracy. The result opens the possibility of having models that can be quickly and continuously trained with updated data from the satellite of the Copernicus program for cartography and many other applications.

References

[1] Introducing EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification (Helber et al., 2019) Link: https://arxiv.org/abs/1709.00029

[2] A Close Look at Deep Learning with Small Data (Brigato & Iocchi, 2020) Link: https://arxiv.org/abs/2003.12843

[3] Practical Recommendations for Gradient-based Training of Deep Architectures (Bengio, 2012) Link: https://arxiv.org/abs/1206.5533

[4] Demystifying Dropout (Gao, Pei & Huang, 2019) Link: http://proceedings.mlr.press/v97/gao19d.html

[5] Demystifying Batch Normalization vs Dropout (Kim, 2021) Link: https://medium.com/mlearning-ai/demystifying-batch-normalization-vs-drop-out-1c8310d9b516

[6] Class Activation Mapping. A powerful method for weakly supervised object localization and debugging (Brus, 2020) Link: https://towardsdatascience.com/class-activation-mapping-using-transfer-learning-of-resnet50-e8ca7cfd657e

Appendices

Reference list of models

RestNet-50 - 1 x dense layer (units=128, activation=relu); 1 x dense layer (units=10, activation=softmax); epochs=120; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9; initialization=imagenet

Baseline shallow CNN - 3 x [convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2)]; 1 x dense layer (units=128, activation=relu); 1 x dense layer (units=10, activation=softmax); epochs=120; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 1 - 3 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=120; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 2 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 3 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 4 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x batch normalization layer]; 1 x dense layer (units=128, activation=relu); 1 x batch normalization layer; 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 5 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=64, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=500; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 6 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=50; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 7 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.5)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.7); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=50; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 8 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=200; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 9 - 2 x [convolution layer (units=32, size=(3,3), stride=1, activation=relu) + convolution layer (units=64, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=100; batch size=32; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Model 10 - 2 x [convolution layer (units=64, size=(3,3), stride=1, activation=relu) + convolution layer (units=128, size=(3,3), stride=1, activation=relu) + max pooling layer (size=(4,4), stride=2) + 1 x dropout layer (rate=0.25)]; 1 x dense layer (units=128, activation=relu); 1 x dropout layer (rate=0.5); 1 x dense layer (units=10, activation=softmax); epochs=100; steps per epoch=200; batch size=16; data augmentation (zoom range=0.2, shear range=0.2); optimizer=SGD; initial learning rate=0.001, decay rate=1e-6; decay steps=1e6; momentum (nesterov)=0.9

Figure 2 - ResNet-50 baseline

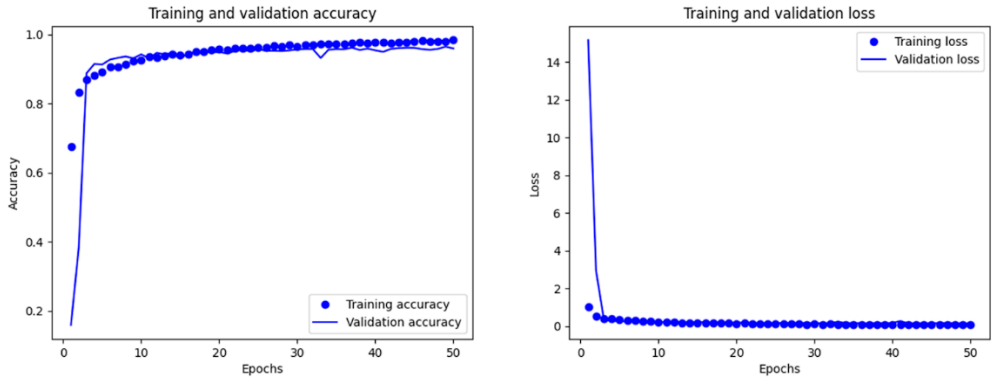

Figure 3 - shallow CNN baseline

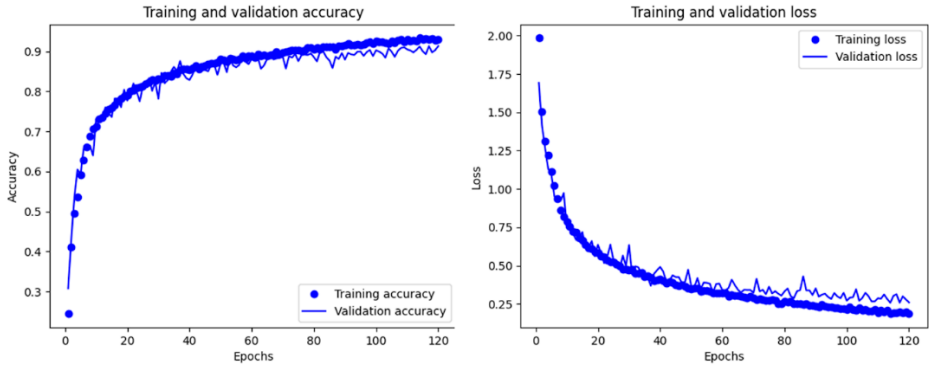

Table 1 - Experimental results

Model	Accuracy (val)	Accuracy (train)	Training time (seconds)
RestNet-50	0.9576	0.9556	2504.20
Shallow CNN	0.9119	0.9033	860.15
1	0.8761	0.8704	905.55
2	0.9165	0.9076	866.20
3	0.9274	0.9124	899.37
4	0.8796	0.8752	936.91
5	0.9107	0.8952	888.19
6	0.7228	0.7083	367.62
7	0.6678	0.6383	367.78
8	0.8380	0.8450	539.45
9	0.8309	0.8181	498.88
10	0.8604	0.8587	778.85

Table 2 - Model 8 breaking point results

Samples	Samples per class	Accuracy (val)	Accuracy (train)	Training time (seconds)
6480	600 - 700	0.8807	0.8696	537.94
4860	450 - 540	0.8380	0.8450	539.40
3240	300 - 360	0.8505	0.8474	536.83
1620	150 - 180	0.7928	0.7717	419.46

Figure 4 - Model 3 confusion matrix

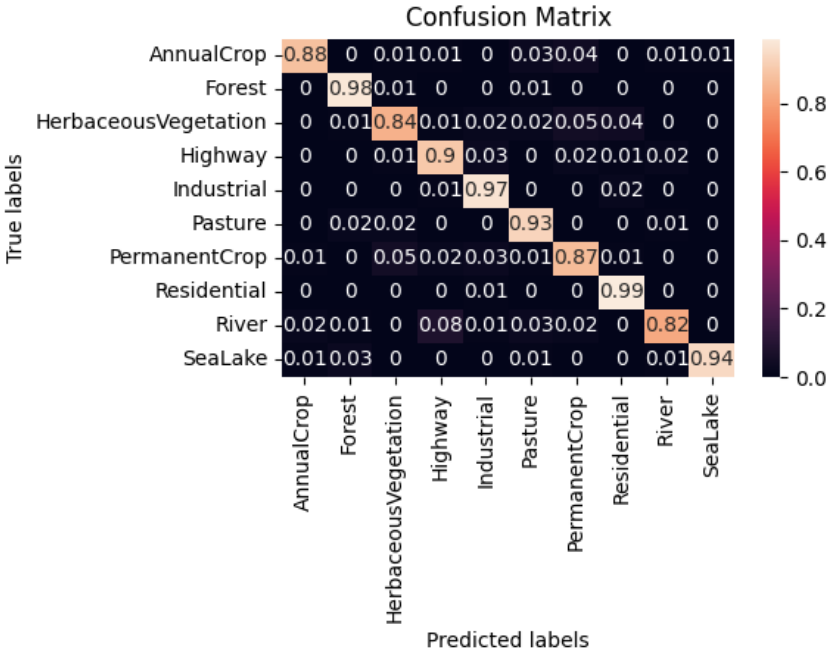

Figure 6 - Baseline shallow CNN confusion matrix