Adapting Occlusion-Based Latent Representation to work with Geospatial Data.

Mick Tijdeman, s1954636

Bachelor Project Creative Technology

Supervisors: Dr. Andreas Kamilaris, Dr. Le Viet Duc.

University of Twente, Enschede

31/07/2021

Abstract

Occlusion-Based Latent Representation is a method within image recognition developed by Karatsiolis and Kamilaris in 2021. The method can turn binary multi-labelled datasets into meaningful representations and can find correlations within the labels of such a dataset. It does this by pre-training a classifier network to generate distributions for the labels and uses this classifier to train a Siamese network. While originally developed to work with facial data, in this thesis Occlusion-Based Latent Representation is adapted to work with geospatial data. Two satellite datasets are used to train the classifier network, the PlanetLab dataset and the Orthoimages dataset. Both are combined with the Copernicus Corine European Land Cover dataset to provide labels to use. As geospatial data is a challenging type of data to classify with a general-purpose classifier, Occlusion Based Latent Representation could not be adapted to work with geospatial data. This is presumed to be because of the high level of imbalance within the dataset and the broadness of the labels within the dataset, as well as the performance of the classifier network itself. Several methods are tried to improve the classifier performance, namely Weighted Classes, Pruning Labels, Randomized Batches, Combining Labels, Focal Loss and Lowering the Binary Accuracy Threshold. Of these methods, Combining labels is found to increase the performance of the networks the most, although not enough to work properly with Occlusion-Based Latent Representation. To make Occlusion-Based Latent Representation work with geospatial data in the future, it is advised to use a pre-existing classifier made specifically for the recognition of multi-labelled geospatial data and combine this with a Siamese network.

Acknowledgements

This project was guided and supervised by Dr. Andreas Kamilaris, who I would like to thank for his apt feedback, quick responses to any inquiries I had and great dedication to the project. His expertise on the subject was well appreciated and helped greatly in the ideation of what steps to take and what methods to employ during the project. Even when things looked south at times, his attitude helped in keeping me motivated to keep going as well.

I would also like to thank Dr. Savvas Karatsiolis, whose technical insights within the field of Deep Learning has been a huge help in developing the project. I have learned much from discussing the models with him, and like think my abilities within the field of Deep Learning have greatly improved with his guidance.

I would like to thank my critical observer Dr. Le Viet Duc, whose sharp feedback on my report has been of great help in improving it to the state in which it is now.

Finally, I would like to thank MS. Chirag Padubidri, who has been of great help in preparing the datasets used in this research. His quick response times were greatly appreciated, as well as his willingness to take my comments on the data into account.

Table of Contents

Chapter 1 – Introduction

Machine learning methods have seen a vast increase in usage over the past decade in their applications within a range of technologies and sectors [1]. Through years of research, these networks are now a key technology in classifying various types of data. Convolutional Neural Networks are a subtype of machine learning which are specialized in the classification of multi-modal data, such as audio, video, and image data. Their applicability includes computer vision, speech recognition, translation, and even healthcare [2]. Where most traditional image classification methods specialize in the classification of objects within images, multi-labelled data classification (also called Multi-Label Learning or MLL) forms an interesting area in which a multitude of labels are utilized to describe the semantics of data. Classifiers for this usually binary labelled data have also shown promising results, utilizing machine learning techniques to classify said labels [3]–[6]. However, representing data with binary labels results in a representation that forms a high-level abstraction of the input, condensing the data to a degree that is impossible to reconstruct and which does not account for the more subtle semantics found within the data. In contrast, labels that are described by a distribution instead of a binary representation form a more meaningful representation of the data, mitigating the problem of context unawareness and including semantics of the data. Therefore, research into these distributive forms of labelled data classification has seen an increase over the past years. This research has already seen widespread success within the field of natural language processing for the development of the word2vec model, developed by Mikolov et al [7], which exemplifies the ability to use distributive label representations as a means of finding meaningful labels to describe the semantics of words. Similarly, in the field of image recognition, Karatsiolis and Kamilaris developed Occlusion Based Latent Representation (OLR) [4], which has the ability to classify 38 distinct binary labels within the facial image data and generate a distribution for the labels within said data, capturing meaningful semantics in the process. Because this distributive representation maintains semantics, the technique allows for the storage of the data more efficiently as well as the approximate recreation of the original image from just distributions. On top of this, the technique can be used to find correlations between the labels associated with the data, which can be used to find new insights into data interrelations.

Currently, OLR has only been applied to facial data. The architecture however allows other forms of labelled image data to be used as input as well. This raises the question of whether such a method may be successful in a wider range of labelled image data. Studying such data interrelations may reveal properties about data that previously went undetected by human subjects. In this thesis, OLR will be applied to a different type of image data, namely geospatial data. Geospatial data is a type of data that has been documented well over the past decades. As a result, labelled data of land cover is readily available and can thus be used in combination with OLR. The goal of this thesis is to discover whether this method is effective in classifying labelled geospatial data and generating a distributive label representation of said data. If this proves successful, it is a further indication that OLR can be a beneficial method within the broader deep learning community and industry. On top of this, the fully trained network can be developed to find data intercorrelations within land data, which can prove useful in many applications. Within the remote sensing area, the ability to estimate specific land categories based on just satellite imagery or surrounding (correlated) land can be beneficial. Besides, a deeper understanding of the semantics of land data may help in the generation of artificial terrain, which has applications within areas like simulations and (educational) video game development. The study of the data interrelations for different areas of land may also further research in the way types of land interact with each other, and the way human involvement in nature influences this. Also, representing land data in a labelled fashion can mitigate its storage requirements greatly, as satellite data often takes much space to store.

While still experimental, this research should prove beneficial for the deep learning and remote sensing communities, providing a tool to use in future research and applications regarding geospatial data and further building upon the broader labelled data research area.

Chapter 2 – Related Work

2.1 Multi-Label Learning models

There have been multiple studies dedicated to labelled data classification and its applications. The original MLL models used logical labels, where every label is represented in a binary fashion with -1 indicating absence and +1 indicating presence [8]. To improve upon this and save more of the data semantics, Geng et al. [3] introduced a method called Label Distributed Learning (LDL) which introduces a distribution to MLL to differentiate in the importance of district labels. However, to train the LDL, the label distribution must be available beforehand, which is often not the case as assigning label distributions to data manually is highly labour intensive and prone to human bias. Shao et al. [5] improve on LDL by introducing Label Enhanced Multi-Label Learning (LEMML). This method removes the need for label distributions to be present in the dataset beforehand by incorporating regression of the numerical labels and label enhancement in the method. Consequently, the method is applicable for many more types of data, as long as it has binary labels present.

An often-used method to recognize correlations in data is the Siamese network, introduced by Chopra et al [9]. In a Siamese network, two or more distinct pieces of data are run through an identical network, and their outputs are assessed for correlation. This method has seen success in image recognition applications, most notably facial recognition [10]. Karatsiolis' OLR combines a siamese network for correlation detection with a distributive MLL model to find correlations in facial features [4]. This method shows promising results, reducing the image data to a collection of feature distributions from which the original image can be recreated somewhat accurately. This is achieved by utilizing a normal convolutional neural network trained on the CelebA dataset, a dataset containing over 200000 labelled facial images [11], and in turn using this CNN to train the siamese network to classify facial images and find correlations between the labels contained within the CelebA dataset.

2.2 Geospatial Data research

Since the launch of several synthetic radar satellites in the period between 2013 and 2016, the interest in geospatial data research has seen a steady increase [12]. The European Copernicus program is one of the results of these satellite launches, a free dataset labelling continental Europe into 48 labels, which will also be used in this research [13]. As a result of these launches, there is now a bigger body of research on the topic of geospatial data and satellite image recognition [14]. Khatmani et al. [15] found that the best performing methods of classifying land-cover data were Support Vector Machine (SVM) and Neural Networks (NN), with a similar performance of about 85% accuracy for both. However, for big applications SVM appeared to use up too many resources, making NN the preferred choice. Within the NN domain, Deep Learning, specifically Convolutional Neural Networks (CNN), performed best [16], [17]. This makes sense, as CNNs were specifically designed to identify spatial data such as satellite data. Most research so far has however looked at classifying rather specific types of land area as opposed to broad categories [14], [18], [19]. This research in the recognition of specific types of land data within satellite images has so far been successful. Kassul et al. [14] recognized crops types within Ukraine, reaching about 85% accuracy for major crop types using a 2-dimensional CNN architecture.

Shendryk et al. [20] made an attempt similar to our research at classifying general labels for satellite data. High-resolution (<10 m) image data of the Amazon Rain Forest was analyzed using 17 distinct labels. While not as broad as the Corine dataset (48 labels) this was already a step towards more general-purpose land image classification. The amazon landscape provided a good base for this, having a large amount of similar land available for training the network. However, not all satellite data has this luxury of providing this many similar samples, and thus the research may not be fully representative for other cases of land data classification. The method achieved about 90% accuracy using binary cross entropy on the labels as the metric to evaluate the performance.

Chapter 3 - Background

3.1 Convolutional Neural Networks

To recognize satellite data, and most other spatial data in general, Convolutional Neural Networks (CNN) are used. At its core, a CNN is a series of layers that attempts to correctly classify spatial data. A layer consists of several nodes which are connected to nodes in the next layer. The exact relations between nodes are determined by weights and biases, which are values that can be updated by the CNN by training. The weights and biases are updated so that patterns can be recognized by layers. A low-level layer may pick up on lines and edges, where a certain combination of these lines and edges may trigger a node in a higher layer corresponding to a shape. A certain combination of shapes may then be used to recognize objects. This process goes on until the network has classified the data. Also see figures 2 and 3 for examples for low- and high-level layers, respectively.

To determine these weights and biases and train the network, a process called backpropagation is used. During this process, training datasets are fed into the network to be predicted. After all data has been evaluated, the network uses a loss function to calculate how far off the predictions were. The loss function must be optimized to ensure the network correctly classifies the data as often as possible. To optimize the loss function, its local minimum must be found. This can be done by taking its derivative and shifting all the function's variables (the earlier discussed weights and biases) "down-slope". The steeper the slope, the more the variable is in- or decreased, until a local minimum is found (See figure 1). This training process aims to find the optimal settings for the weights and biases for each node in the network. By doing this, the network can correctly classify the input data as often as possible.

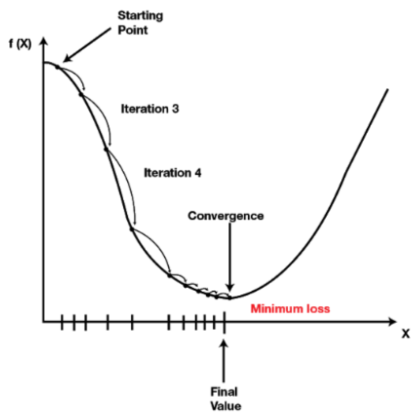


Figure 1: Finding the local minimum of a function [21].

The training process as described above is common practice in Deep Learning, and thus not unique for CNNs. CNNs are optimized for spatial data by use of the convolutional layer. This layer takes a combination of data points in close proximity and checks whether this combination appears in the next layer to form a more complicated combination of data points. Figure 2 shows such a low-level convolutional layer once properly trained, having multiple edges and lines it can recognize. Figure 3 shows higher-level convolutional layers, which can recognize complex structures, for example faces.

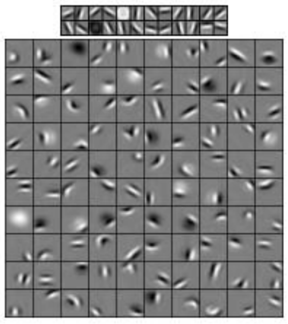


Figure 2: Example of low-level convolutional Layers [22]

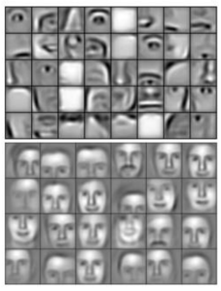


Figure 3: Example of high-level convolutional Layers [22]

After these convolutional layers, a fully connected layer is traditionally used to converge the network into the desired number of possible outputs within the network. These are however only the basics of CNNs, many methods have been developed to improve or change the performance of these networks to make them work for specialized purposes. These methods often include special layers, various loss and activation functions, data manipulation and more.

3.2 Occlusion Based Latent Representation

The OLR model will be adapted to work with the Copernicus Corine 2018 dataset [13], [23]. The OLR model consists of two important parts, the classifier network and the Siamese network. The classifier serves as a reference point for the Siamese network to utilize in its training phase. To calculate the loss function for the Siamese network, the output of the classifier is compared with the output of the Siamese network, over which the Mean Squared Error is taken, to calculate its loss function. Therefore, before the classifier can be used in combination with the Siamese network, it has to be pre-trained on the dataset. After this, the Siamese network can be used on the same dataset.

3.2.1 Classifier

As mentioned, the Siamese network must be trained with an existing classifier to properly classify the labelled data. The classifier that is used must be able to classify visual data into several distinct binary labels. For this purpose, a Multi-Label Learning Convolutional Neural Network is used. The network architecture as described in the original paper [4] is relatively straightforward and consists of 2 convolutional layers followed by a maxpooling layer. This structure is repeated five times in total, with the last maxpooling layer being replaced with a fully connected layer with *N* outputs, where *N* is the number of distinct label categories for the image data. The output consists of N distributions representing the likeliness of a label being present in the image and is a value between 0 and 1 represented by a 32-bit float. Also see figure 4 for the architecture.

The images which are analyzed by the classifier have a certain width and height (w & h), and three colour channels: red, green and blue, each with a pixel value between 0 and 255. These pixel values are normalized to be between 0 and 1. Therefore, the total input of the classifier is a matrix of $w \times h \times 3$ values between 0 and 1. The classifier converts this input via the CNN structure to a classification of N distinct labels, which represent the probability of the label category being present on the image.

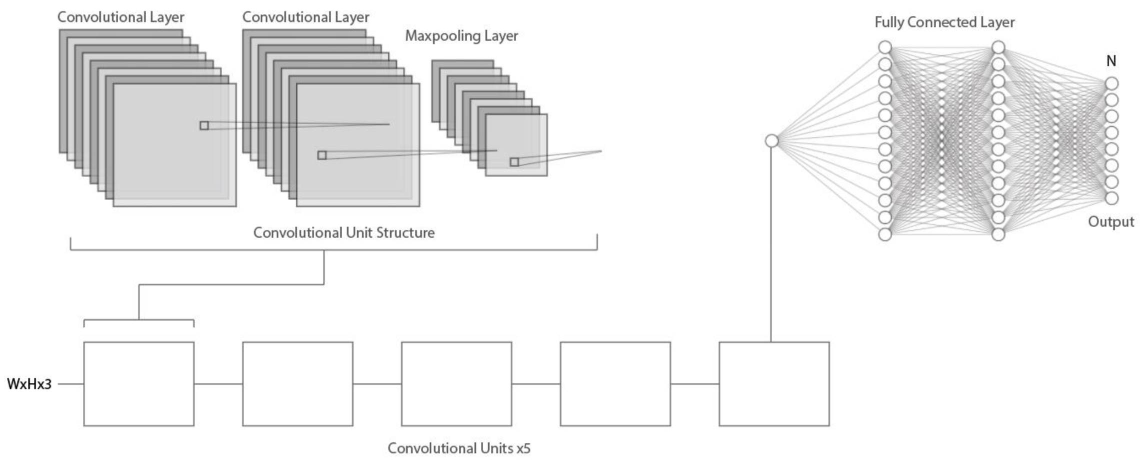


Figure 4: Schematic representation of the classifier network architecture. Note that hyperparameters are variable and not accurately represented in the image to maintain legibility.

This structure of classifier was chosen as a basis as it has already been shown to work in combination with OLR and the code was readily available for usage. It must however be noted that the exact classifier used in OLR can also have a different architecture, as long as it results in an output of N probability distributions that successfully classify the N labels in the dataset. Therefore, it is an option to change the network architecture should this prove necessary in the research.

3.2.2 Siamese network

The Siamese network is used to classify images and find correlations within the fed dataset. The models' architecture relies on having two images inputted, which are both evaluated by the same Convolutional Neural Network, consisting of a structure similar to the classifier. However, where the classifier ends with a fully connected layer, the Siamese network has a more thorough architecture which allows it to compute meaningful label embeddings for image pairs. After the convolutional and pooling layers, the model consists of *k* feature maps of a size $f \times f$ (where f is a tunable hyperparameter) for every label $x \in R^N$. Each of these groups of feature maps is responsible for classifying one labels' probability vector. The number of feature maps *k* per group is equal to the dimensionality of each embedding's vector size. So, if a vector were to be saved in a 32-bit size format, this would mean *k* = 32. In total, there are $N \times k$ feature maps in this layer. As the feature maps are of a size $f \times f$, the total output size of this layer is $N \times k \times (f \times f)$. For every feature map, an average pooling layer is used to compute the average of the map, decreasing the size from $f \times f$ to only 1 scalar. Thus, the output of this average pooling layer is $N \times k \times 1$. This output can be reshaped to *N* layer vectors of size *k*, thus representing the *N* label classifications for the input image. Also see figure 5:

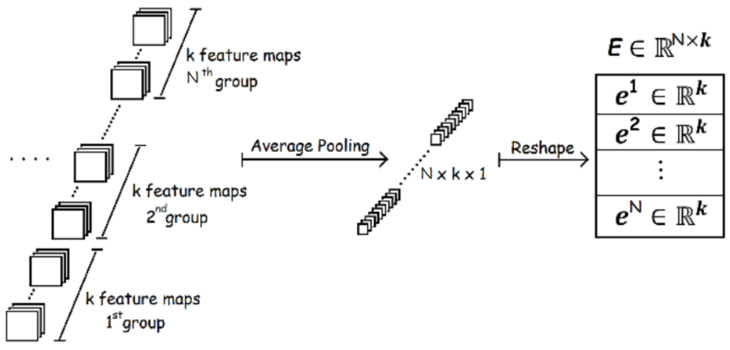


Figure 5: The final layers of the Siamese Network [4].

Once this process has been done with two input images, the dot product between the computed probability vectors is calculated and serves as the final output of the Siamese network. This means that when images share a label, the outputted dot product should be high. To train the network, a loss function is required to evaluate the performance of the network after each iteration. The loss function chosen is the Mean Squared Error (MSE) between the Siamese network's dot product vector and the product of both image probabilities as assessed by the classifier network. To minimize the MSE, the Siamese dot product and the classifier probability vector have to be as close as possible. This way, the network is trained to recognize the probabilities as assessed by the classifier. The MSE is calculated as $\frac{1}{N}\sum_{i=1}^{N}(c_i - s_i)^2$ which computes the average error over the *N* datapoints *i*, which are evaluated for the output of classifier *c* as a "correct" reference and the output of the Siamese Network *s*.

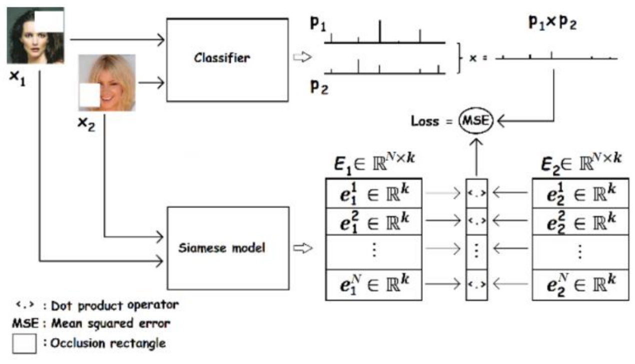


Figure 6: Schematic representation of the training phase of the Siamese model [4].

The classifier tends to output probability vectors that are close to 0 or 1, resulting in joint probability vectors that may closely resemble binary classifications as opposed to distributions. When these overconfident vectors are used in the loss function to assess the Siamese model, the Siamese model is trained to also become overconfident in its predictions, which may result in overfitting. To combat this, the Siamese model makes use of random occlusion. In other words, the images which are used in the Siamese network get a randomly placed square imposed over them, which hides some of the features in the image. This prevents the Siamese network from becoming overconfident and encourages it to find meaningful distributions more closely representing the actual distribution a certain feature may have.

3.2.3 Dataset

The dataset to be used in the analysis is the Copernicus European Land Cover 2018 dataset. This dataset consists of 48 distinct labels indicating the presence of various land types in continental Europe. The labels are distributed over a land map of Europe utilizing a vector map, see figure 7 for an example of the dataset, and figure 8 for the labels.

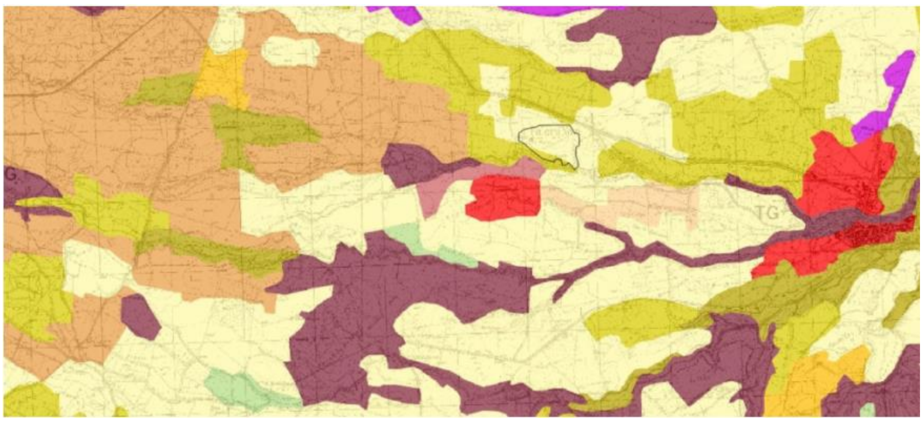
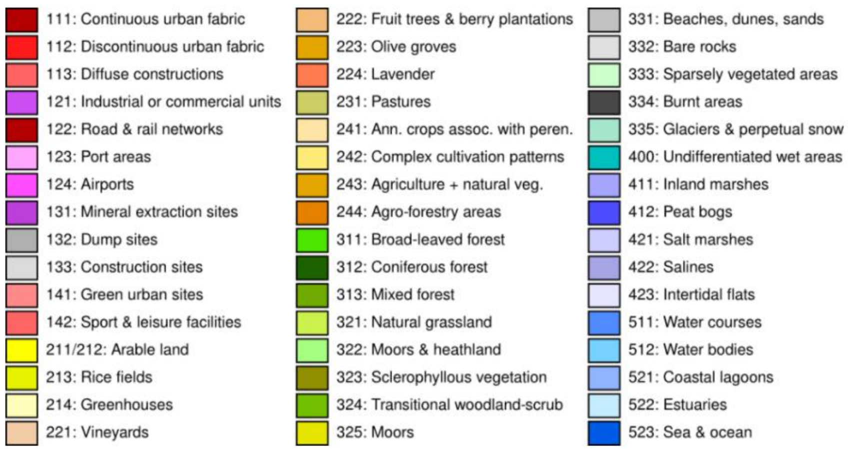


Figure 7: Several vector labels of the Corine land cover dataset. Each colour indicates a different label representing a type of land use [24].



🟥 111: Continuous urban fabric	🟧 222: Fruit trees & berry plantations	⬜ 331: Beaches, dunes, sands
🟥 112: Discontinuous urban fabric	🟧 223: Olive groves	⬜ 332: Bare rocks
🟥 113: Diffuse constructions	🟧 224: Lavender	🟩 333: Sparsely vegetated areas
🟪 121: Industrial or commercial units	🟨 231: Pastures	⬛ 334: Burnt areas
🟥 122: Road & rail networks	🟨 241: Ann. crops assoc. with peren.	🟩 335: Glaciers & perpetual snow
🟪 123: Port areas	🟨 242: Complex cultivation patterns	🟦 400: Undifferentiated wet areas
🟪 124: Airports	🟧 243: Agriculture + natural veg.	🟦 411: Inland marshes
🟪 131: Mineral extraction sites	🟧 244: Agro-forestry areas	🟦 412: Peat bogs
⬜ 132: Dump sites	🟩 311: Broad-leaved forest	🟦 421: Salt marshes
⬜ 133: Construction sites	🟩 312: Coniferous forest	🟦 422: Salines
🟥 141: Green urban sites	🟩 313: Mixed forest	⬜ 423: Intertidal flats
🟥 142: Sport & leisure facilities	🟨 321: Natural grassland	🟦 511: Water courses
🟨 211/212: Arable land	🟩 322: Moors & heathland	🟦 512: Water bodies
🟨 213: Rice fields	🟩 323: Sclerophyllous vegetation	🟦 521: Coastal lagoons
🟨 214: Greenhouses	🟩 324: Transitional woodland-scrub	🟦 522: Estuaries
🟧 221: Vineyards	🟨 325: Moors	🟦 523: Sea & ocean

Figure 8: The categories of land data as defined by the Corine dataset [24].

The Corine dataset only includes the labels. The classifier network however must learn to associate image data with these labels. Therefore, the dataset has to be coupled to actual image (satellite) data. The PlanetScope satellite dataset is used for this purpose. For the network to work, this image data must be separated into numerous images consisting of patches of land which together make up a bigger land area. Each patch must then have its own file with associated labels for the network to learn and recognize. There are a few considerations to take into account when preparing this dataset. First of all, the resolution of the satellite images chosen. When the resolution is higher, it is easier for the network to correctly classify the land, as the data is more clear. However, this also causes the land to be more densely represented as it takes more pixels to visualize the same amount of land. Thus making the network slower and more computationally intensive. On top of this, an image patch of the same pixel size would be able to display less label information if it were of a higher resolution as it would display less land area. A balance must be found between image clarity and computational size.

Secondly, the size of the image patches has to be considered. The bigger a patch, the more land can be displayed and thus the more labels can be included in one image. As the goal is to discover correlations, having multiple labels per analyzed patch is crucial. However, when the patches are too big they may include labels in one image which are not related, causing the network to learn correlations which in reality do not exist. Once again, the right balance must be found in the patch size for optimal results.

To find which works best, two different satellite datasets are considered in this research, which vary largely in resolution. The first dataset is the PlanetLab dataset [25], which has been collected on the 30th of April 2021 and has been downloaded on the 24th of May 2021. It has a resolution of 3m per pixel, which is relatively small. The advantage of this dataset is that reasonably big patches of land area can be used while maintaining images that are computationally feasible for the classifier to work with. Also, since the resolution is small, more land can be covered in total for the same data size (1 GB of data can hold more land in this dataset as opposed to one with a higher resolution). Therefore, the dataset allows for more land and as a result more labels to be covered. This means the more precise label interrelations could be collected, assuming the classifier performs well on the dataset. The second satellite dataset is the High-Resolution Orthoimages Dataset [26], whose data has been collected on the 31st of March 2012 and has been downloaded on the 29th of June 2021. It has a considerably higher resolution of 0.5m per pixel. The images are more detailed, as the land area can hold relatively more information in this dataset. Thus, the classifier should be able to classify distinct labels relatively easier, at the cost of heavier computational operation.

3.3 Hardware

Training a complex network like this one requires a high amount of computational power. When training a Neural Network, it is preferable to make use of a high-end GPU that has the capacity to make the required calculations to successfully train the network quickly enough. For this purpose, the University of Twente (UT) server cluster and Google Colab are considered for training.

3.3.1 UT server cluster

The UT server cluster has several powerful GPUs available to use and can run programs indefinitely when required. For Deep Learning operations, the NVIDIA TITAN X and GeForce GTX 1080 Ti are utilized. The server however requires some overhead in setting up before programs can be run properly and has the tendency to randomly fail on some arbitrary iteration of the program, in which case no output log is returned. The cluster also does not allow for reading and writing to and from .txt files, which means that keeping track of results between sessions must be done manually. Also, no intermediate results are shown during execution, the output log is only visible after a complete execution. Finally, the Cluster consists of a waiting queue before programs can be run, which varies highly in waiting time.

3.3.2 Google Colab

Google Colab also has several powerful GPUs which can be used with relative ease, as all required dependencies are installed by default. The GPU utilized is the Tesla K80. It makes use of a Jupyter notebook for its interface, which allows for quick testing and editing of individual blocks of code while keeping the rest of the program intact, including their variables and outputs. However, Colab is made for interactive use only, and does not allow for a program to run for over 90 minutes without interaction and will terminate user sessions when no input is detected, for which it will check using a captcha.

Considering the advantages and disadvantages of both systems, the UT server cluster has been chosen to run training applications on, as these often need long-running times to properly train the network. For evaluation and debugging related purposes, Google Colab is used for its relative ease in execution and immediate feedback when used, as well as its lack of waiting queues to use.

Chapter 4 - Methodology

4.1 Procedure

The OLR network will be trained utilizing the two aforementioned datasets, which are preprocessed with the Corine dataset to a format compatible to be used by the classifier and Siamese network. To evaluate its performance, Binary Accuracy is used as a metric for the classifier’s label probability outputs. When a probability for a label is above a certain threshold, it is considered as being evaluated as present. When it is below said threshold, the label is considered absent. This threshold is 0.5 by default but can be changed to improve performance. For every label, there are four possible evaluation results for the classifier: True Positive, True Negative, False Positive and False Negative, where positive correlates with classifying a label as being present and negative correlates with classifying a label as being absent. The Binary Accuracy is found by calculating the percentage of True label evaluations. A manual evaluation of the predictions the classifier makes on the test set will however also be necessary to evaluate its performance. As present (positive) labels are sparse within the dataset, it is deemed more important for the dataset to get these labels correct. When the dataset accomplishes a high accuracy, yet has very few or no True Positives, the classifier is not deemed as being successfully trained despite high accuracy.

For the Siamese network, the loss function (MSE) is used as a performance metric during training for the network to access itself. When the network has finished training, a manual inspection of the results is done as an evaluation. Specifically, an NxN matrix is created which displays the correlations found between labels. This matrix is compared with the NxN co-occurrence matrix of the labels, which shows how often labels are found together on images. If the Siamese network is trained well, it is expected that these two matrices resemble each other in their division of high and low values. (E.g. when two labels have a very low co-occurrence, they should also have a strong negative correlation and vice versa). If the Siamese network can find these correlations itself, this is deemed a success. Once trained, this would yield a Siamese classifier capable of finding correlations within new land data as well, without the help of the classifier network.

In the case that the networks are not evaluated to perform well as according to the previously mentioned metrics, the networks will be adapted to increase performance. There are several methods available to be tried in this case, the ones which to use will be decided according to an evaluation of the network to try and understand what causes the misperformance. This way, the proper method in response to the misperformance can be applied. These methods will be discussed in more detail when they are brought into use.

4.2 Testing on facial data

Before being able to adapt the OLR network to be usable with land data, the network is tested with facial data, as the original network is made for the recognition and labelling of said facial data [4]. An attempt is made to replicate the findings of this original paper to confirm that the correct methodology can be followed and applied with the available resources.

The network is a python-based application, utilizing the commonly used TensorFlow and Keras libraries for their deep learning methods which are used to realize the network. The dataset used to test the model is the CelebA dataset [11], a dataset consisting of over 200,000 images of faces which are divided over 100 image batches of size 2000. The images are annotated with 40 different binary labels, indicating the presence or absence of said label. The images are cropped to be 178 x 178 pixels in dimension, which often cuts off the neck of said images. Therefore, the labels “wearing necklace” and “wearing necktie” are not considered, making for a total of 38 labels. The images are divided into 100 batches of 2026 images each as a Numpy array with a dimensionality of $2026 \times 178 \times 178 \times 3$. This represents the 2026 images of 178x178 pixels, where each pixel has 3 colour channels with a value between 0 and 255, representing the amount of red, green or blue in the pixel.

First, the classifier is trained on this dataset. During one iteration of training, 98 of the 100 batches are used to train the network, and the remaining 2 are used to test the performance of the network. To assess the performance of the network, binary accuracy is chosen as the metric to be used. As the network produces a probability vector between 0 and 1 for each binary label, this metric assesses whether the prediction for each label is closest to the actual label. So, if a label for a certain image were to be predicted with a probability of 0.6, this would return a positive assessment if the label were present on the image and negative if the label were not present. The metric makes no distinction based on the confidence of the predictions, meaning that a probability of 0.05 (very confident) is assessed the same as a prediction of 0.45 (very uncertain). The metric returns the percentage of labels correctly classified by the network.

Second, the Siamese network is trained on the same dataset, utilizing the trained classifier to assess itself after each training iteration. As discussed previously, the MSE is used as the metric to evaluate the model. A low MSE indicates little difference between the prediction made by the classifier and the Siamese network and is thus deemed desirable. The network attempt to minimize this MSE. After the model has been trained, the correlations found between labels are assessed using a heatmap. If this were to return realistic correlations in line with the original paper, the classifier and Siamese network have both performed well on the dataset.

The classifier was trained until an accuracy of 91% was reached on the test set, at which point it did not appear to improve anymore. The Siamese network was then trained with the classifier up until an MSE of 0.0135 was reached, after which the MSE results of the network plateaued. To test the network, 2 training batches were classified by the Siamese network and analyzed for the discovered correlations between labels. This resulted in the following correlation heatmap:

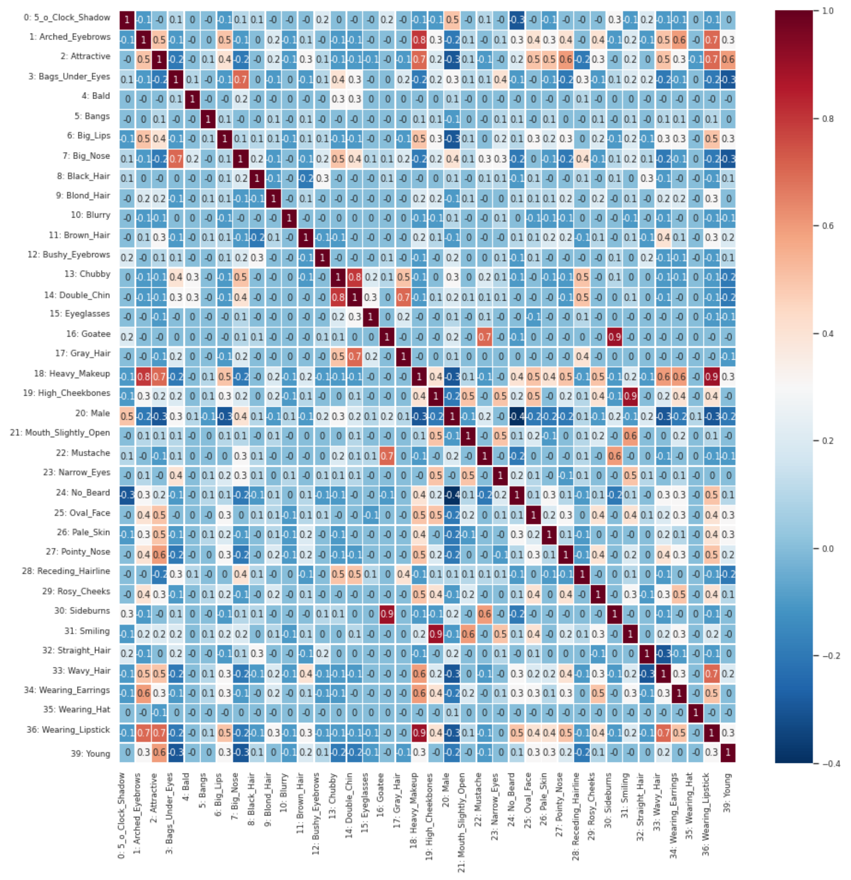


Figure 9: Heatmap of correlations detected by the Siamese network in the CelebA dataset.

These correlations appear logical at a first glance (“Chubby” and “Double Chin” for example have a very high correlation, while “wearing lipstick” and “male” have a very low correlation, etc.). The correlations resemble those found in the original paper to a decent degree. There are however a high number of correlations that are simply set to zero. This suggests that the network may have not picked up on more subtle correlations within the dataset. Also, the “weaing_hat” label has an exceptionally high amount of neural correlations, which may suggest that the classifier did not recognize the label at all. This is possible if the label is underrepresented in the dataset, in which case it can get the label “correct” by simply guessing it to be zero the majority of the time.

4.3 Preparing the datasets.

As it is confirmed the networks (Classifier and Siamese network) can classify and correlate facial image data correctly, the network can be prepared to be used with geospatial data. For this purpose, both satellite image datasets are prepared with the Corine dataset for the network to classify. Since the entire Corine dataset is far too big to analyze, as it consists of the entirety of Europe, a subset of the dataset is used for this research. For both satellite datasets, the data used consists of a part of Cyprus around the cities of Nicosia and Larnaca. This area was chosen somewhat arbitrarily, primarily because the research institute for which the project is done is based in Cyprus. See figure 10 for the land which is used in the analysis.

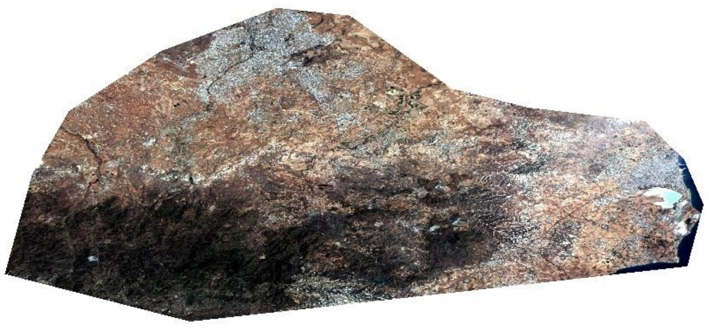


Figure 10: PlanetLab satellite data used for analysis. In the north, the city of Nicosia can be seen and in the east the city of Larnaca. Apart from this, a big portion of the dataset consists of mountain ranges and forests, providing all types of nature labels.

Both datasets must be prepared by preprocessing them from their original format to a NumPy array structure which can be used as input for the classifier and Siamese networks. The network expects each image as an array in the format (w,h,b) where *w* is the width of an image, *h* is the height of an image and *b* is the number of bands. To make analyzing the image arrays more efficient, the network utilizes the Tensorflow *train_on_batch()* method. This method expects an input of (n,w,h,b) where *n* represents the number of images in a batch. So, all images are divided over batches of size *n.* If there are *I* images, this means that there are a total of $B = I/n$ batches. The number of batches and the size of the batches are chosen such that the network can train on big batches without experiencing issues while training, as batches too big may cause the program to experience memory issues. This also means that if an image (w,h,b) is of a smaller total size $w \times h \times b$, the size of the images batches *n* can be made larger and vice versa as to have the total size of batch $w \times h \times b \times n$ of a size manageable by the program. The exact size which works or does not work must be decided experimentally for the network, as there is no simple way to calculate this.

Apart from the arrays representing the images, the networks also require the present labels per image. The labels are gathered from the Corine dataset based on the associated images' geolocation coordinates. The batches are expected by the network in an (n,l) format, where *n* represents the number of label arrays in this batch and is equal to the size *n* of the image batches. The *l* represents the number of possible labels associated with the image. The labels are formatted as a binary NumPy array of size *l* indicating the presence of a label with a one, and absence with a zero. Thus, every image in a batch has a corresponding label array which can be found on the same location in a batch. For example, an image that has the labels 2, 3 and 7 present of size 10 will have a label array as such:

[0,1,1,0,0,0,1,0,0,0]

Once a dataset has been properly prepared, it consists of *B* image batches named *0.npy, 1.npy, … B.npy* and *B* label batches named *labels0.npy, labels1.npy, … labelsB.npy*. With these batches, the network can be trained. The details getting each dataset in this format differ somewhat, however, as the type in which they are acquired differs.

4.3.1 PlanetLab Dataset

The first dataset which is prepared is the PlanetLab dataset. This dataset is in TIF format, a datatype specifically used for geolocation data. As mentioned, these must be converted to NumPy arrays. Additionally, the satellite images must be divided into patches of a certain width and height. A size of 2 kilometres by 1 kilometre is chosen as the size. This way, the patches will contain around 3 labels per patch. This results in 14100 patches of 327 x 132 pixels. While this may seem odd, as the ratio of the pixels is not 2:1 as might be expected, this is because of the satellite images not being taken on a flat surface due to the curvature of the earth, thus skewing the axes somewhat. The pixel images in turn are converted to 4 NumPy arrays per image. Every array represents a colour band within the image, which are bands for red, green, blue and near-infrared. See figure 11 for examples of the 4 bands and figure 12 for an example of a full image.

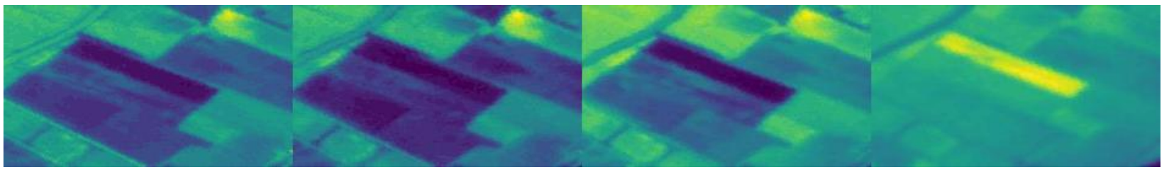


Figure 11: Example of the 4 bands representing a patch. From left to right: Red, Green, Blue, Near-Infrared. Note that this is a representation of the intensity of each band within part of an image using the Viridis colour mapping. The actual bands are 1 dimensional

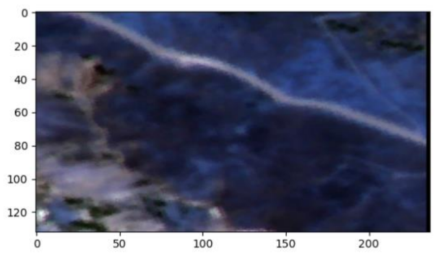


Figure 12: Example of a full image of the PlanetLab dataset.

The network however expects an array in the format (w,h,b) where *w* is the width of an image, *h* the height of an image and *b* the number of bands. For this dataset, this is an input of (327,132,4). The four bands are currently separated into arrays of (327,132) each. This means that the bands must be combined into a bigger array. This is done through the Numpy method *stack()*, which creates a bigger array out of the individual ones. However, when transferring from geolocation data to pixels, several band arrays are formatted to a dimensionality of either (327,133) or (326,132). To be able to use the stack method and evaluate the images correctly with the network, these arrays have to be converted to the (327,132) dimensionality. To do this, the (327,133) arrays are stripped of the last row-vector, thus making the image slightly smaller. This should not influence the performance of the network much, as it is unlikely one horizontal stripe of pixels will hold decisive information on the recognition of a label. For the (326,132) arrays, they were extended with a column vector that copies the column next to it. This way, the extension should be as close to reality as possible. It could also be extended with a simple zero-value column-vector, however, this adds the risk of the network recognizing wrong labels because of the odd data. Now that the band arrays all have a consistent dimensionality, they can be stacked to a (327,132,4) array, which the networks can use. As mentioned, the images must be combined into batches. The *stack*() method is used once more to create 141 batches of size 100 each. This way, there are 141 batches of images.

The dataset has a total of 32 different labels present throughout the various patches on the map of Cyprus. This is less than the total labels in the Corine dataset, as some labels do not apply to the chosen area. Since the image arrays are divided into batches of size 100, the label arrays must be of a dimensionality (100,32) where each image has a correlated label array. This makes for a total of 141 batches of label arrays.

When converted from the Corine dataset, the labels are represented in a CSV file which has 1 label per row, combined with the appropriate coordinates of the patch as such:

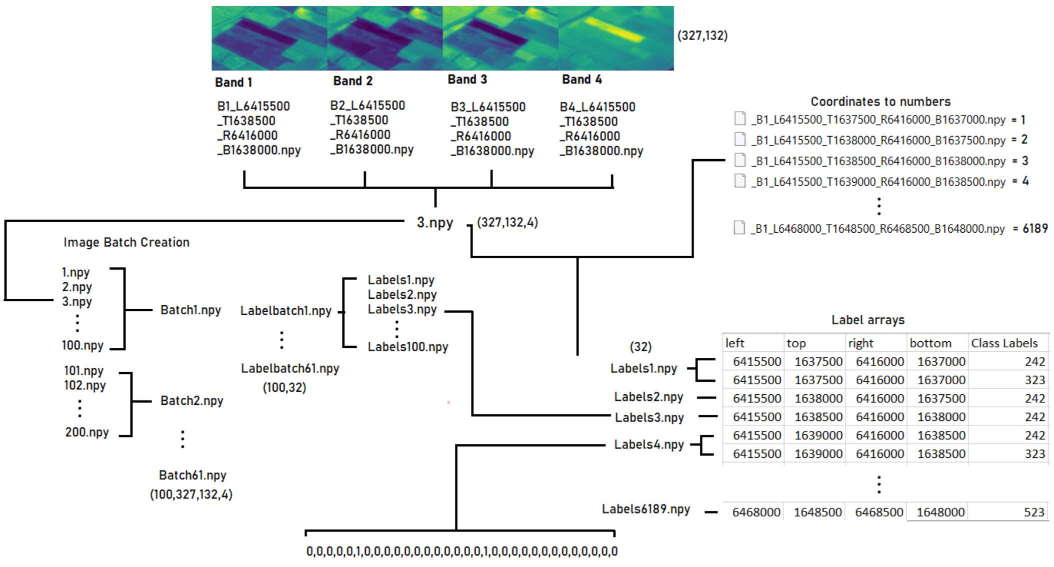
[left, top, right, bottom, class label]

As mentioned, the information in this CSV file must be converted to a binary label array per image. To do so, it must be assessed with which image number the label correlates. As the image numbers are not stored in the CSV file, this is done via the coordinates which are stored in the CSV file. Since the image arrays are ordered numerically and contain coordinates of their patch of land, the coordinates can be used to combine the class label with the number of the image array. To achieve this, a list is made containing the numbers of all images and their respective coordinates. Then, the CSV file is iterated over to see which number this class label correlates with, after which the binary label array is updated for this label from a 0 to a 1. The class labels in the Corine dataset are numbered somewhat odd (also see figure 8). To make numbering more logical, the first binary label in the array correlates with the lowest-numbered Corine label and so on. The final label representation looks approximately like the following:

Image	Label 111	Label 112	Label 121	Label 122
42	0	1	1	0
43	1	1	0	0
44	0	0	0	1

Table 1: Representation of binary label array data structure. In reality, the batches are of size 100 with 32 possible labels, as opposed to the 3x4 array in this example.

The full preprocessing of the data schematically looks as following:


(327,132)
Band 1
Band 2
Band 3
Band 4
B1_L6415500 _T1638500 _R6416000 _B1638000.npy
B2_L6415500 _T1638500 _R6416000 _B1638000.npy
B3_L6415500 _T1638500 _R6416000 _B1638000.npy
B4_L6415500 _T1638500 _R6416000 _B1638000.npy
Coordinates to numbers
_B1_L6415500_T1637500_R6416000_B1637000.npy = 1
_B1_L6415500_T1638000_R6416000_B1637500.npy = 2
_B1_L6415500_T1638500_R6416000_B1638000.npy = 3
_B1_L6415500_T1639000_R6416000_B1638500.npy = 4
_B1_L6468000_T1648500_R6468500_B1648000.npy = 6189
3.npy
(327,132,4)
Image Batch Creation
1.npy
2.npy
3.npy
100.npy
Batch1.npy
101.npy
102.npy
200.npy
Batch2.npy
Batch61.npy
(100,327,132,4)
Labelbatch1.npy
Labelbatch61.npy
(100,32)
Labels1.npy
Labels2.npy
Labels3.npy
Labels100.npy
(32)
Labels1.npy
Labels2.npy
Labels3.npy
Labels4.npy
Labels6189.npy
Label arrays
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0

left	top	right	bottom	Class Labels
6415500	1637500	6416000	1637000	242
6415500	1637500	6416000	1637000	323
6415500	1638000	6416000	1637500	242
6415500	1638500	6416000	1638000	242
6415500	1639000	6416000	1638500	242
6415500	1639000	6416000	1638500	323
⋮				
6468000	1648500	6468500	1648000	523

Figure 13: Schematic representation of the Preprocessing of the PlanetLab dataset. Note that this version assumes a total of 6189 images as opposed to the final 14100.

4.3.2 Orthoimages Dataset

The Orthoimages dataset consists of 1100 PNG images of size 5000x5000. Because of the high detail of the dataset, the total size is 57 GB. Each of these images represents an area of 2500 x 2500 meters. Contrary to the PlanetLab dataset, this has already been adjusted for the earth curvature, thus both axes are of equal size. Also see figure 14 for an example of one of the images:

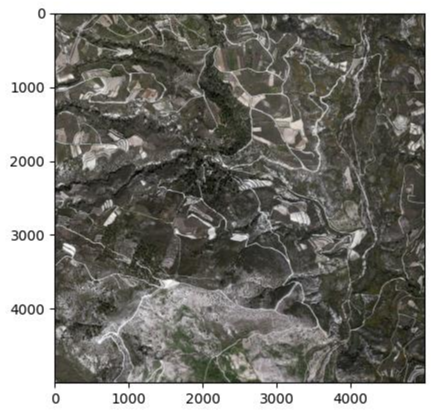


Figure 14: Example of a 5000x5000 image from the Orthoimages Dataset

As the PNG images already hold all 3 colour bands, the images can easily be converted to NumPy arrays of size (5000,5000,3). While this does account for the required (w,h,b) structure for the network, this size of an input image is too big and will cause memory overload within the program. To account for this, the network was run experimentally with images of input sizes 1000x1000 and 500x500, as these are both sizes easily yielded by a division of the original 5000x5000 size. Regrettably, the network failed to run with images of size 1000x1000 as well. Thus, the images were cut to a size 500x500. See figure 15 for an example of such a 500x500 image.

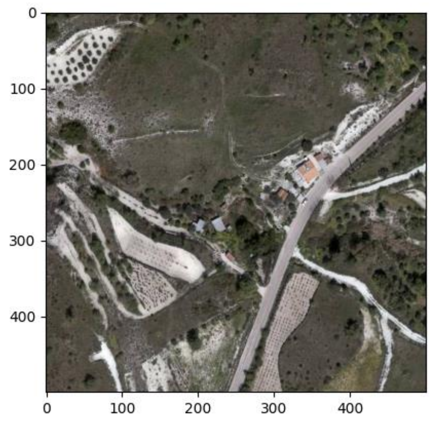


Figure 15: Example of a 500x500 image from the Orthoimages Dataset

The labels are also formatted in 1100 grayscale PNG images of size 5000x5000. In these files, each pixel value represented the label that was present on that location in the image. For example, if a pixel has a value of 18, this corresponds to the 18th label, which is “Pastures”. Since the label numbering in the Corine dataset is not continuous and goes beyond 256, the maximum value a pixel can have, the labels were numbered 1 to 48 based on the same numerical order as in the Corine dataset. For the original ordering, see figure 8. Not all of the 48 labels were present in the dataset, however, as not every type of label occurs in Cyprus. In total, 34 distinct labels are present in this dataset. This is 3 more than the PlanetLab dataset, as this one covers more land in total. This means that every image needs an associated label array of size (34). To do this, the 5000x5000 label images were cut up into images of 500x500. Over these cut-up images, the *np.unique()* function was used, which returns all unique values found within the array. As the values in the images represented the label found in the respective location, this function returns all labels found within the image, so an image that has labels 4, 23 and 31 would return [4,23,31]. However, this array must be converted to a binary array of size 34. This can be done by creating a size 34 array filled with zeros and replacing it with a 1 at the index of each label-1 (as the array starts counting at 0 while the labels start counting at 1). Keeping to the previous example, this would yield the following label array:

[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0].

As each of the 1100 images is cut up into 100 smaller images, the dataset now consists of 110000 images of size 500x500. The cut-up images each show a land area of 250 by 250 meters. This is a small area, as many of the images will hold only one label. Since a key part of the network is finding correlations within the data, images with multiple labels are strongly preferred. To compensate for this and the fact that the original dataset size of 57GB will take very long to train, a subset of images that hold 3 labels or more is selected to be used in training. This shrinks the original dataset down from 110000 images to only 3500, all of a size 500 x 500. This size is however still relatively big compared to the original input of the network, which is 178 x 178. To compensate for the size of these images, the batch size must be adapted as to restrict the batches from becoming too large in size and prevent memory overload. Therefore, the batch size was chosen to be 10 images per batch. Stacking the image arrays and label arrays thus results in batches of dimensionalities (10,500,500,3) and (10,34) respectively making for a total of 340 batches each.

4.4 Additional Methods

There are a number of differences between the facial CelebA dataset, which was originally utilized in the OLR paper, and the satellite datasets. These differences may hinder the performance of the network. To improve results, several additional methods can be utilized in the training of the network, should it perform subpar.

4.4.1 Weighted Classes

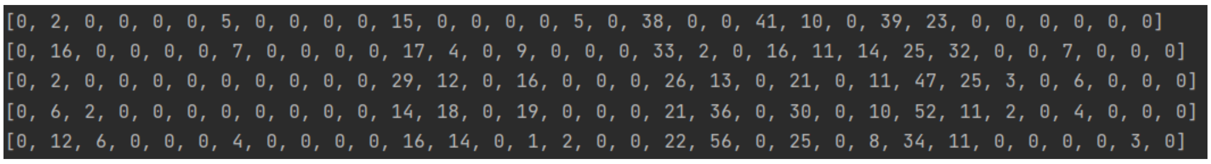
The most obvious difference between the CelebA and Corine dataset is the division of the labels between images. It is quite varied how often labels are present within satellite images. Looking at the PlanetLab dataset, the most prevalent label (non-irrigated arable land) is present in about 38% of images, while the least prevalent label (beaches, dunes, sands) is present in only 10 out of the 14100 images. This means that that the network can potentially get very high accuracies by simply “guessing” labels to be absent from images. To account for this, the classes will be weighted in the network to assign a heavier penalty for evaluating their presence wrongfully to the network. This is done using the Tensorflow method *class_weight()*, a parameter assigned in the training phase of the program. It is important to properly assign these weights so that the network can evaluate them properly. For each batch of images, the class weights are calculated with $\frac{1}{f} \times \frac{t}{2}$ where f is the frequency of the class within the batch and t *is* the total number of labels assigned in the batch [27]. In this formula, the less frequent a label occurs, the heavier it is weighted. If a label were to not occur within a batch at all, it is set to 1, so it is not taken into account heavily for said iteration.

4.4.2 Pruning Labels

Even with weighted classes in place, the least prevalent labels (<0,1%) may prove hard to learn for the network and can skew accuracy. Therefore, it can also be considered to prune these labels from the network, such that they do not influence the training process. This is done by analyzing which labels are present below a certain threshold in the dataset and shrinking the number of labels that can be associated with an image from the original amount N_1 to shrunk amount N_2. When this is done, the network itself also must be adjusted to have N_2 outputs, as the amount of labels an image can be associated with has changed. It is important to note that while this removes sparsely distributed labels, the absolute amount of labels will still go down, and some images may end up with very few (between 0 and 2) labels associated with them. The relative amount of labels will however go up.

4.4.3 Randomized Batches

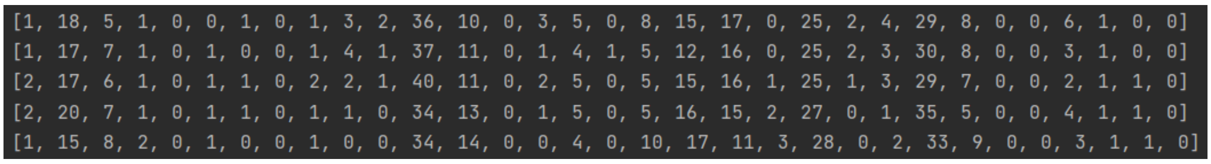
Another problem is that image patches close to each other may share a lot of labels and contain a relatively high amount of absent labels. This is because these are all taken from comparable land. So, when a batch of image arrays is created by iterating over images ordered numerically, a batch does not represent an “average” of the multiple land options present. This can be evaluated by counting the number of times a label is present in a batch of 100 images. When the baches are created “in order”, this results in the following frequencies:



```
[0, 2, 0, 0, 0, 0, 5, 0, 0, 0, 0, 15, 0, 0, 0, 0, 5, 0, 38, 0, 0, 41, 10, 0, 39, 23, 0, 0, 0, 0, 0, 0]
[0, 16, 0, 0, 0, 0, 7, 0, 0, 0, 0, 17, 4, 0, 9, 0, 0, 0, 33, 2, 0, 16, 11, 14, 25, 32, 0, 0, 7, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 29, 12, 0, 16, 0, 0, 0, 26, 13, 0, 21, 0, 11, 47, 25, 3, 0, 6, 0, 0, 0]
[0, 6, 2, 0, 0, 0, 0, 0, 0, 0, 0, 14, 18, 0, 19, 0, 0, 0, 21, 36, 0, 30, 0, 10, 52, 11, 2, 0, 4, 0, 0, 0]
[0, 12, 6, 0, 0, 0, 4, 0, 0, 0, 0, 16, 14, 0, 1, 2, 0, 0, 22, 56, 0, 25, 0, 8, 34, 11, 0, 0, 0, 0, 3, 0]
```

Figure 16: Frequencies of labels within the first 5 batches when batches are created in order. Columns represent different batches, where the first number within a column is the first label, etc.

To combat this, batches can be created which take a random combination of 100 images, instead of those that happen to be in order. That way, the batches should represent more of an average label division within the dataset. Doing this yields the following label frequencies:



```
[1, 18, 5, 1, 0, 0, 1, 0, 1, 3, 2, 36, 10, 0, 3, 5, 0, 8, 15, 17, 0, 25, 2, 4, 29, 8, 0, 0, 6, 1, 0, 0]
[1, 17, 7, 1, 0, 1, 0, 0, 1, 4, 1, 37, 11, 0, 1, 4, 1, 5, 12, 16, 0, 25, 2, 3, 30, 8, 0, 0, 3, 1, 0, 0]
[2, 17, 6, 1, 0, 1, 1, 0, 2, 2, 1, 40, 11, 0, 2, 5, 0, 5, 15, 16, 1, 25, 1, 3, 29, 7, 0, 0, 2, 1, 1, 0]
[2, 20, 7, 1, 0, 1, 1, 0, 1, 1, 0, 34, 13, 0, 1, 5, 0, 5, 16, 15, 2, 27, 0, 1, 35, 5, 0, 0, 4, 1, 1, 0]
[1, 15, 8, 2, 0, 1, 0, 0, 1, 0, 0, 34, 14, 0, 0, 4, 0, 10, 17, 11, 3, 28, 0, 2, 33, 9, 0, 0, 3, 1, 1, 0]
```

Figure 17: Frequencies of labels within the first 5 batches when batches are created randomly. Columns represent different batches, where the first number within a column is the first label, etc.

As can be seen, the amount of zero-frequency labels within batches is lower when this method is applied. The batches now represent the average label division over the entire dataset more accurately. Since the training set used by the classifier also consists of a combination of (usually 2 – 5) batches, the training sample also represents the full set more accurately, making the chance the network overfits on specific labels smaller.

4.4.4 Combining Labels

Another method to deal with sparsely distributed labels without pruning them is to combine labels into more general categories. As can be seen in figure 8, the labels within the Corine dataset are quite specific. For example, all labels referencing urban structures can be combined into one general label called “urban”. It must be noted however that this may also cause the network to have a harder time distinguishing between specific patterns which make up labels if the labels become too generalized or if the combined labels are not similar enough. The choice of which labels to combine must be made carefully, taking this into account.

4.4.5 Focal Loss

Focal loss is a loss function first introduced by Lin et al [28]. This loss function is specifically useful for highly imbalanced data, as it encourages the model to predict a label as being “present” more frequently, even if it is not very sure about this assessment. This is contrary to the more commonly used Binary Cross-Entropy loss, which requires the model to be very certain about its evaluation. While Focal loss does increase the risk of getting a false positive in the evaluation, the model needs to predict positive values more frequently in a dataset that holds very few positive values like the Corine dataset.

4.4.6 Lowering the Binary Accuracy Threshold

The default threshold for the binary accuracy metric is 0.5, meaning that any evaluation of a label with a probability higher than 0.5 is counted as “present” and lower as being “absent”. To encourage the classifier network to evaluate labels as being present more often, this threshold can be lowered. This way, a low evaluation where the network is only somewhat sure a label correlates with an image will still be counted as being present. While this may improve the performance of the classifier, it should be noted that if this method proves successful, the Siamese network still has to be trained with this classifier. As the Siamese network attempts to imitate the (cross product) probabilities output by the classifier, this will also mean that the Siamese network will put out relatively lower probability estimations.

Chapter 5 - Results

To test the OLR network in combination with the datasets, multiple runs were executed of both datasets. Between these runs, various hyperparameters were edited and analyzed for their influence on the training and prediction results of both networks.

5.1 PlanetLab Dataset

The PlanetLab dataset was run with the classifier network in multiple configurations to attempt to get the classifier to properly train on the dataset. This was ultimately to little avail, as the network did not appear to actively improve its classifications much. However, a few configurations did seem to train somewhat successfully. The ones which did were selected to train with the Siamese network, after which it was evaluated whether the network was able to find realistic correlations between labels. For the classifier network, a successful result is seen as the network training for higher accuracy, as this indicates it learns the underlying patterns and semantics of the data. If the network starts with a high accuracy which it does not improve, this is not deemed successful, despite high accuracy.

5.1.1 Results First Run

The first training run of the classifier network did not indicate it learning to recognize data, as the classifier plateaued in its training accuracy after only a couple of iterations. Analyzing this run for its label classifications reveals that the classifier learned to classify every label as being absent, also see table 2:

True Positive	True Negative	False Positive	False Negative	Total Accuracy
0	14979	0	1021	0.9361875

Table 2: Classifiers' evaluation of the test set after the first training run.

This is an indication that the sparsity of labels hinders the classifier's ability to recognize labels. Therefore, the earlier discussed methods to deal with spare label division are applied to the network to assess whether they improve its performance. Consequently, this will investigate if the sparse division of labels is indeed the cause of its subpar classifications. If these methods do not improve performance, the cause may be different.

5.1.2 Pruned Labels, Weighted Classes and Randomized Batches

The methods applied to the dataset/classifier to check for better results are weighing classes, pruning labels and randomized batches. The methods were applied in all possible configurations, for a total of 8 different networks. For pruning, a threshold of 100 'hits' was chosen. So, if a label appeared less than 100 times within the network, it was purged. This slimmed the network down from 32 classes to 16. The result of these runs can be found in table 3:

Pruned Labels	Weighed classes	Randomized Batches	Starting accuracy	Final Accuracy	Improvement	Iterations	Run ID
Yes	Yes	Yes	0.8851250	0.8852500	0.0001250	9	1
Yes	Yes	No	0.8406250	0.8801250	0.0395000	5	2
Yes	No	Yes	0.8850000	0.8860000	0.0010000	42	3
Yes	No	No	0.8403750	0.8801250	0.0695000	3	4
No	Yes	Yes	0.9383125	0.9383750	0.0000625	5	5
No	Yes	No	0.9230000	0.9361875	0.0131875	2	6
No	No	Yes	0.9383125	0.9385625	0.0002500	51	7
No	No	No	0.9316250	0.9361875	0.0045625	8	8

Table 3: The results of the 3 different method configurations applied to the classifier network. The number of iterations refers to the iteration on which the best performing (final) accuracy was reached.

From this table, it can be gathered that pruning the labels does indeed lower the initial accuracy of the network. However, all configurations stop training around 88%for the pruned labels and 94% for the unpruned labels. This is likely because these approximately represents the percentage of absent labels within the test set. Thus, when the network trains towards guessing absent, these percentages are reached. This is confirmed when evaluating the predictions done on the training set by these networks, as they all have a very low number of present (positive) guesses. Also see table 4:

Pruned Labels	Weighed classes	Randomized Batches	True Positive	True Negative	False Positive	False Negative	Run ID
Yes	Yes	Yes	7	7075	6	912	1
Yes	Yes	No	2	7039	2	957	2
Yes	No	Yes	65	7023	58	854	3
Yes	No	No	0	7041	0	959	4

No	Yes	Yes	2	15012	1	986	5
No	Yes	No	0	14979	0	1021	6
No	No	Yes	62	14955	58	925	7
No	No	No	0	14979	0	1021	8

Table 4: Label evaluations of the 3 different method configurations applied to the classifier network.

A number of observations can be made from these classifications. Firstly, label pruning does not appear to influence the performance of the network apart from shifting the accuracy range in which it operates. Secondly, the runs which appeared to perform best based on accuracy improvement (runs 2, 4 and 6) did so merely by shifting towards more negative evaluations. It is also interesting to note that runs which lack randomized batches evaluated labels as being present the least, as 3 out of the 4 runs never evaluate any label as being present. This is likely because non-randomized batches have their label division more focused on specific labels which show a high frequency, rather than an average division. Thus, these have relatively more labels always absent, also see figure 16.

Two runs have a relatively higher number of positive evaluations, runs 3 and 7. These runs both reached their best performing accuracy on a relatively late iteration (see table 3) and are both runs that lack weighted classes but do have randomized batches. This implies that this combination encourages the network to learn to evaluate labels as being positive relatively more. Closer inspection of these runs reveals that most of these positive evaluations were done for the same labels, namely the labels “Non-irrigated arable land” (151/243) and “Scleropyllous vegetation” (27/243). The remaining labels evaluated as positive were a variety of labels within the dataset. The label “Non-irrigated arable land” is the most common label within the dataset, appearing within 37% of images. This may imply that the network has learned to recognize this label within images. It is not plausible that it evaluates this label as being present most of the time as to simply get a high accuracy, as to get a higher accuracy the network can still guess this label as being absent and get it right for the majority of the images evaluated. Indeed, when looking at the evaluations for only “Non-irrigated arable land” the two networks had a combined accuracy of 65.8%. While this is far from state-of-the-art CNN performance, it does indicate that the positive evaluations for this label are not arbitrary but learned. This does indicate that the network is capable of learning when a label is present more frequently, as this label indeed is, however, this result is only reached with randomized batches and without weighed classes. It must also be noted that while these networks performed relatively well on one label, overall, the networks still followed the trend of outputting mostly absent labels, as they also have a high number of false negatives.

This also shows that weighing classes does not improve the performance of the network, but on the contrary, decreases performance. This is likely because the weight given to the classes is inversely proportional to their frequency. Thus, the labels which are absent more of the time will weigh higher when calculating the loss function. While the intended outcome of this is that the network learns these labels better, it is shown that the result is the opposite. The network is encouraged to still evaluate these labels as absent most of the time, as these evaluations also weigh heavier.

5.1.3 Combined Labels, Lowered Threshold and Focal Loss

As these three methods do not appear to sufficiently solve the problem, additional methods are incorporated into the classifier program. Since random batch distributions did appear to increase performance slightly, all runs from here on will utilize this method. The additional methods are combining labels, lowering the binary accuracy threshold and focal loss. For combining the labels, the previous 32 labels were combined into a total of 7 new label categories. For the conversion, see Appendix C3. For lowering the binary accuracy threshold, a new threshold of 0.2 was chosen. These methods resulted in the following performance for the classifier:

Method	Starting accuracy	Final Accuracy	Improvement	Iterations
Combined Labels	0.81314284	0.83742857	0.0242857	287
Lowered Threshold	0.8882083	0.9250208	0.0368125	407
Focal Loss	0.75135416	0.919375	0,16652781	33

Table 5: Results of the Combined Labels, Lowered Threshold and Focal Loss methods applied to the classifier network training phase. The iteration refers to the iteration on which the Final Accuracy was reached.

These results are more promising than the previously applied methods. Namely, the Lowered Threshold and Focal Loss methods appear to increase the performance of the network significantly. Note that Focal Loss has a far lower number of iterations as this network had to be run on Google Colab instead of the UT server cluster, as the server cluster did not support the library required to run Focal Loss. For this reason, the Focal Loss dataset was also run using a smaller test dataset compared to the Lowered Threshold.

Looking at their evaluations for the test batch yields the following:

Method	True Positive	True Negative	False Positive	False Negative
Combined Labels	1588	7205	693	1014
Lowered Threshold	876	44059	1028	2037
Focal Loss	840	25638	1309	1013

Table 6: Label evaluations of the Combined Labels, Lowered Threshold and Focal Loss methods applied to the classifier network.

Despite having only little improvement in accuracy, combining labels does appear to be relatively good at assessing positive labels. This is likely due to the labels within the dataset now being represented more often, and positive labels now being more prevalent within the dataset. It assessed the different categories as such:

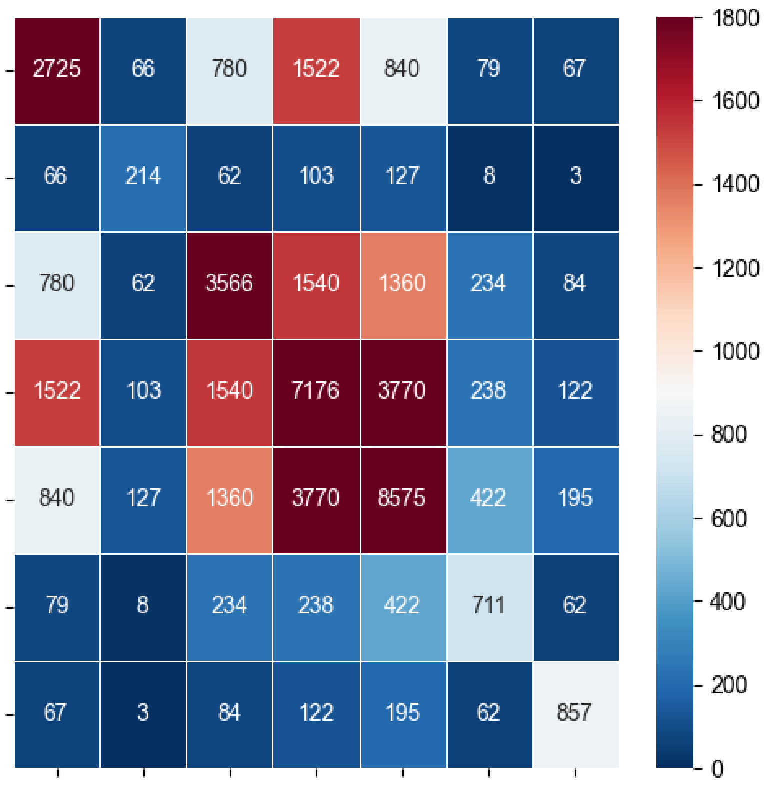
	True Positive	True Negative	False Positive	False Negative
Urban	57	1167	4	272
Special Sites	0	1472	0	28
Non-irrigated arable land	83	1040	33	344
Agriculture	650	290	440	120
Forests	750	410	197	143
Rocky / Mountainous areas	0	1415	0	85
Bodies of Water	48	1411	19	22

Table 7: Performance of the Combined Label classifier per label.

This reveals that the classifier performed well on the labels which were well represented within the dataset. It did especially well on the label “forests”, where the classifier got 83% of the present images correct as well as 68% of the absent images. Interestingly, whenever the classifier assessed an image to contain the “urban” label, it got it correct 93% of the time. However, it still missed most of the urban structures within the dataset. This may imply that there is some specific urban structure that the classifier succeeded at recognizing which accounts for the 57 True Positives. This may be an effect of the overly broad categorization of combining labels. Similarly, the classifier was able to correctly assess whether an image contained a body of water 97% despite its relatively low representation within the dataset, as opposed to merely estimating it to be absent every time.

The Lowered Threshold and Focal Loss methods appear to perform somewhat better as well, although relative to the absolute number of labels they still access a small amount as being positive. It can also be seen that focal loss indeed has a higher number of false positives as was expected of the method.

Since the classifiers trained with these methods appeared to perform decently well, they are used to train the Siamese network. If the Siamese network performs well, it should find correlations between labels that resemble the co-occurrence matrices of the datasets used. For each of these 3 classifiers, this is a different co-occurrence matrix. They can be found here:



2725	66	780	1522	840	79	67
66	214	62	103	127	8	3
780	62	3566	1540	1360	234	84
1522	103	1540	7176	3770	238	122
840	127	1360	3770	8575	422	195
79	8	234	238	422	711	62
67	3	84	122	195	62	857

Figure 18:The Combined Label co-occurrence matrix. Labels have been removed to increase legibility, for each label see Appendix C3.

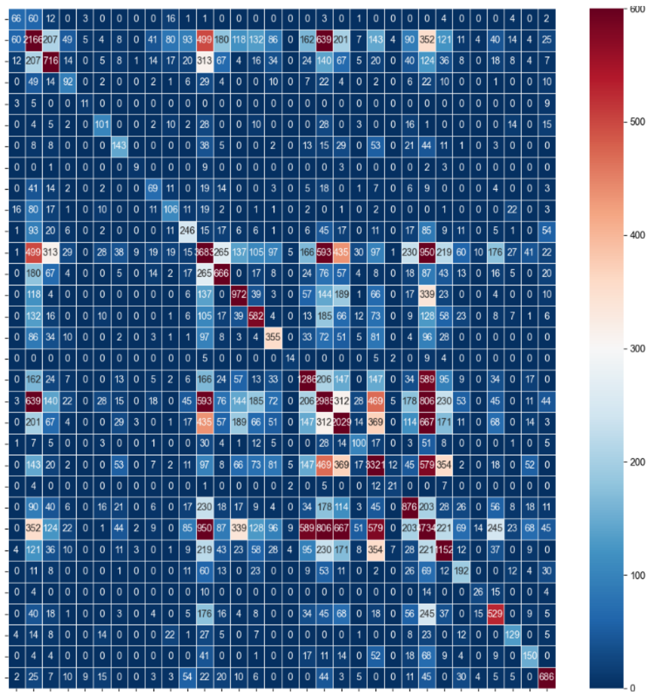


Figure 19: The 32 label co-occurrence matrix, as used in Lowered Threshold and Focal Loss. Labels have been removed to increase legibility, for each label see Appendix C1.

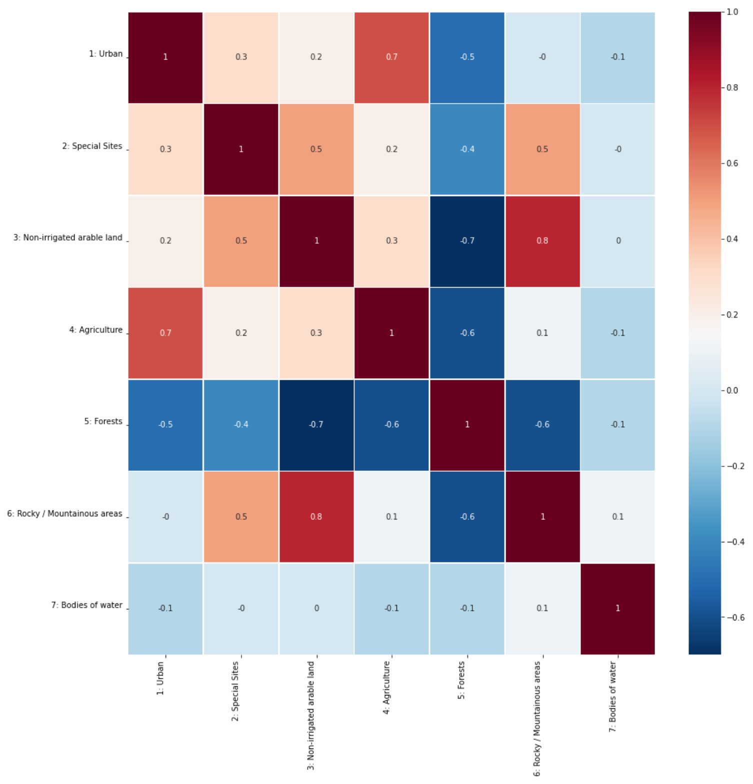
The Siamese network is trained utilizing each of these classifier networks. It used the MSE to access its training results. The final results can be found in table 7:

Classifier used	Starting MSE	Best MSE	Iteration
Combined Labels	0.060082	0.00591019	27
Lowered Threshold	0.00366061	0.001401	95
Focal Loss	0.0029063	0.0024499	39

Table 7: Results of training the Siamese network with classifiers trained on Combined Labels, Lowered Threshold and Focal Loss.

It is interesting to note that the combined labels Siamese network both started and ended with a relatively high MSE. This implies that the error of this network stayed relatively higher compared to other Siamese network iterations, as well as started higher. This may be caused by the relatively high amount of positive classifications made by the Combined Labels classifier.

Since the classifier for these networks has been trained, they can be evaluated by their label correlation output. To do this, the Siamese classifier is run over a sample image test set which it can evaluate for their label correlations. The resulting correlation matrices for Combined Labels, Lowered Threshold and Focal Loss can be found in figures 21, 22 and 23 respectively.



	1: Urban	2: Special Sites	3: Non-irrigated arable land	4: Agriculture	5: Forests	6: Rocky / Mountainous areas	7: Bodies of water
1: Urban	1	0.3	0.2	0.7	-0.5	-0	-0.1
2: Special Sites	0.3	1	0.5	0.2	-0.4	0.5	-0
3: Non-irrigated arable land	0.2	0.5	1	0.3	-0.7	0.8	0
4: Agriculture	0.7	0.2	0.3	1	-0.6	0.1	-0.1
5: Forests	-0.5	-0.4	-0.7	-0.6	1	-0.6	-0.1
6: Rocky / Mountainous areas	-0	0.5	0.8	0.1	-0.6	1	0.1
7: Bodies of water	-0.1	-0	0	-0.1	-0.1	0.1	1

Figure 20: Correlation matrix based on the correlations found by the Combined Label Siamese network.

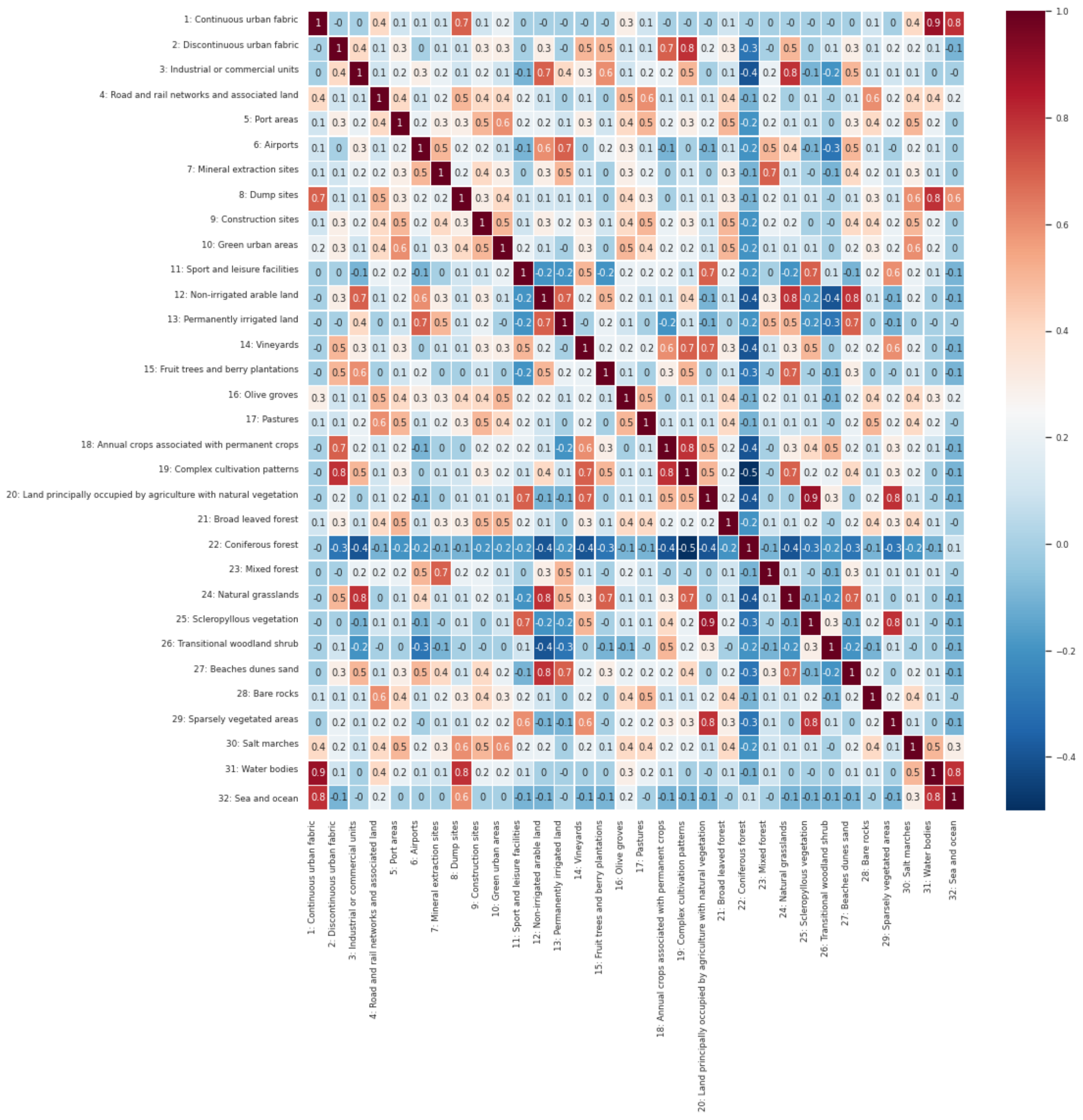


Figure 21: Correlation matrix based on the correlations found by the Lowered Threshold Siamese network.

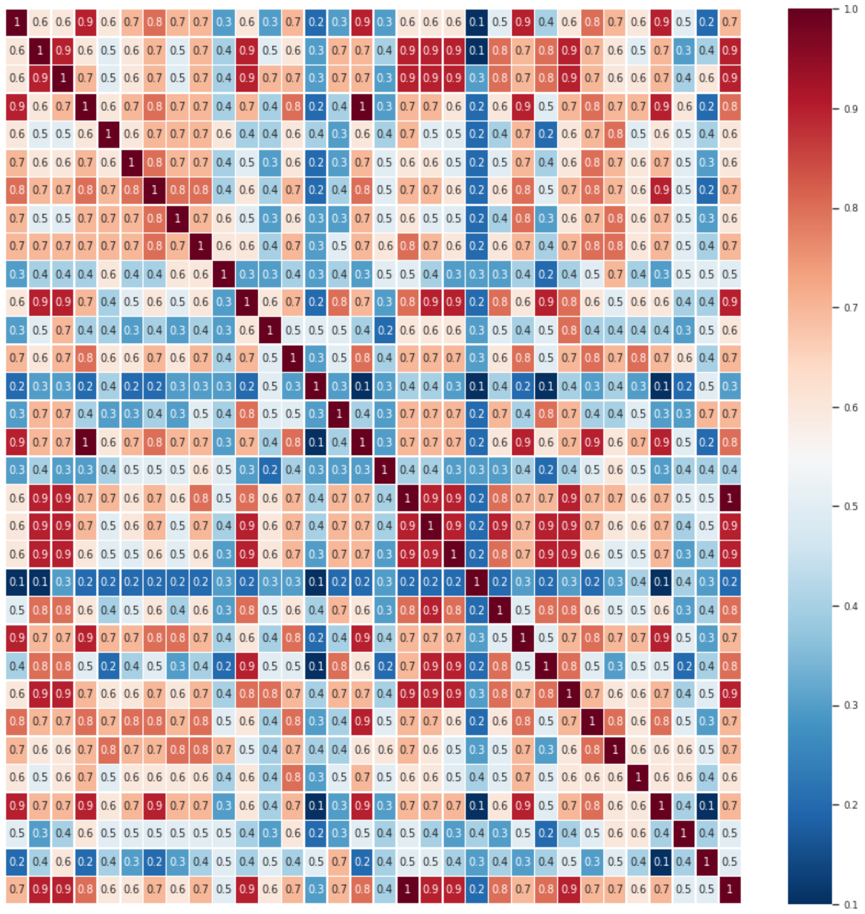


Figure 22: Label correlation heatmap for the Corine dataset on Cyprus based on the Siamese network trained by the Focal Loss classifier network. For labels see Appendix C1..

These matrices in themselves do not tell us much. They must be compared to the co-occurrence matrices of the associated datasets. Doing this reveal some interesting findings. First and foremost, the Lowered Threshold and Focal Loss matrices seem somewhat nonsensical. There are a large number of random samples that can be taken from each respective matrix which do not correlate with the associated co-occurrence matrix. For example, in the Focal Loss matrix, the labels “Sea and Ocean” (32) and “Annual crops associated with permanent crops” (18) appear to have a correlation of 1, indicating that these two are always found together. Analyzing the co-occurrence matrix however shows that out of their 686 and 2985 respective occurrences, only 44 images hold both labels. Thus, the labels hold a rather low correlation, contrary to what the Siamese model found.

If the classifiers were properly trained, their correlation matrices would likely resemble each other somewhat, which is not the case. However, they do seem to agree that the label “Coniferous Forest ” has a very low correlation with all other labels. This is the second-most common label within the entire dataset, which may imply that this label was learned to recognize successfully by both networks, and since other labels are quickly labelled as being absent, a negative correlation with this specific label is logical.

The correlation matrix for the Combined Label network appears to make a bit more sense when compared to its co-occurrence matrix, although it still misses several obvious correlations. It agrees with the previously discussed networks that “forests” has a low correlation with all other labels by merit of its sheer size. It also found a high correlation between labels agriculture and urban, which is correct when looking at the co-occurrence matrix. It however also missed several important correlations, for example, agriculture and forest are found together quite often. Despite this, the network assessed their correlation to be -0.6, the strongest negative correlation given to any label. Considering most of the correlations found are similarly questionable, it is unclear whether the sensical correlations were learned by actual training or simple luck. The number of nonsensical correlations does indicate that the latter is more likely.

Training the classifier and running the full OLR with the PlanetLab dataset did not yield any promising results. While it did give an insight into which method can support the analysis of a satellite dataset, the classifier ultimately tended towards primarily outputting only absent labels and failed to learn the semantics of the provided data. The PlanetLab dataset itself may be inherently unlearnable for a general case classifier network, perhaps because it is of too low a definition. If this is the case, a higher definition dataset could prove fruitful in the analysis of the Corine dataset.

5.2 Orthoimages Dataset

The Orthoimages dataset has a resolution of 0.5 m per pixel, which is far more resolute than the PlanetLab dataset. If the bottleneck within the evaluation of the Corine dataset is indeed the detail found within the satellite imagery, the Orthoimage dataset should perform better. Due to time constraints, the Orthoimages dataset was not analyzed as thoroughly as the PlanetLab dataset since it could only be acquired much later. Despite this, the dataset was prepared and ran, once without any additional methods, once with a Binary Accuracy Threshold of 0.2 and once with Combined Labels. For the exact labels used, see Appendix C5.

Model	Starting Accuracy	Final Accuracy	Improvement	Iterations
Standard	0.912381	0.912381	0	0
Threshold 0.2	0.8338095	0.8841905	0.050381	1
Combined Labels	0.83	0.816	0.014	12

Table 8: Results of running the Orthoimages classifier network applying different methods.

Threshold	True Positives	True Negatives	False Positives	False Negatives
Standard	0	9580	0	920
Threshold 0.2	401	8883	700	516
Combined Labels	187	221	53	39

Table 9: Evaluations of the trained Orthoimages classifier networks applying different methods.

The first two networks do not appear to be training at all. The standard model classifies every label as being absent and does not improve anymore beyond the first iteration. The 0.2 threshold version classifier improves only once, which is not enough for a CNN to legitimately learn to recognize labels. This is reflected in its evaluations as well. While not always absent, the positive evaluations appear to be random as well, as they are most often false.

The Combined Label version once more proves to generate better results. The test set is somewhat small because of the combining of labels, but both for positive and negative labels it manages to generate good accuracies. Table 10 shows a more specific overview of the network's performance per label

Label	True Positive	True Negative	False Positive	False Negative
Urban	64	9	24	3
Agriculture	80	1	19	0
Forests	42	25	10	23
Rocky / mountainous area	0	89	0	11
Bodies of water	1	97	0	2

Table 10: Results per label by the Orthoimages combined label classifier.

As can be seen, the network performs especially well on the first three labels: Urban, Agriculture and Forest. The last two labels appear to be very underrepresented in the dataset, and can therefore not be properly learned by the classifier.

Only the results of the classifier ran with Combined Labels appears to have trained one the dataset, and is therefore used to train the Siamese network. This performed as follows:

Starting MSE	Final MSE	Iterations
0.08682765066623688	0.009012744274167787	20

Table 11: Result of training the Orthoimages Siamese network with Combined Labels

The resulting Siamese network is used to evaluate for correlations within the dataset and compared to the associated co-occurrence matrix. These can be found in figure 23 and 24:

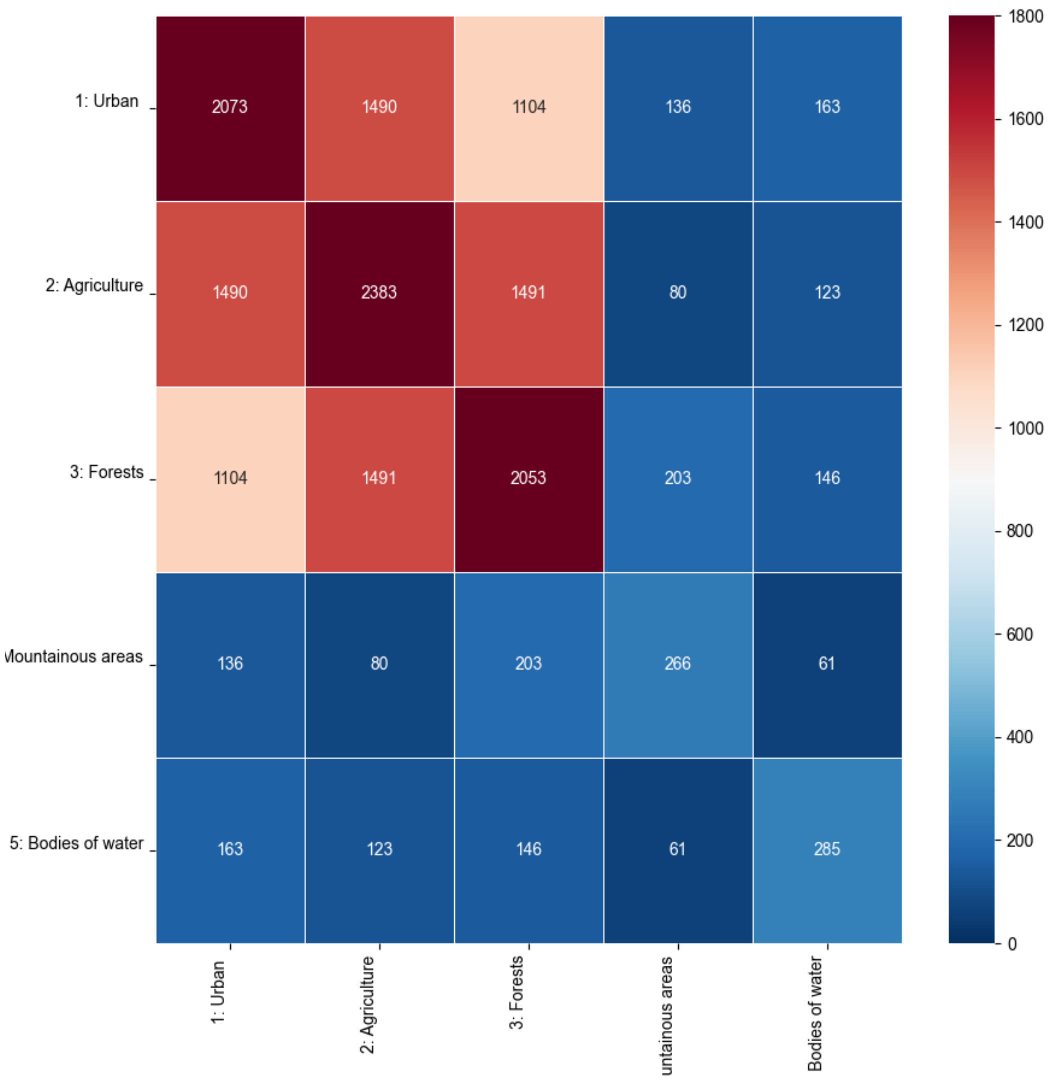


Figure 23: Co-Occurence matrix for the combined label Orthoimages dataset.

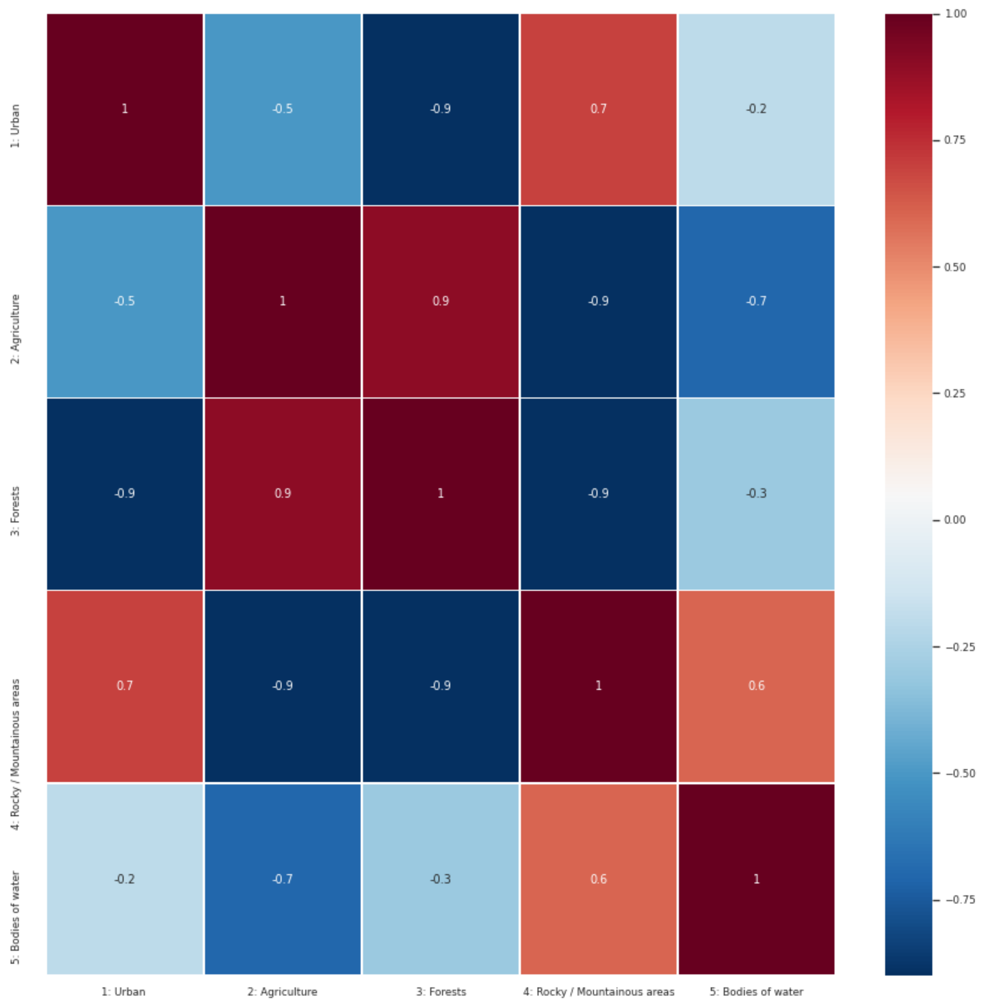


Figure 24: Correlations found by the Siamese network trained with the Combined Label classifier.

Comparing these two matrices shows that the Siamese model did not learn the correlations very well. While it did manage to assess that indeed forest and agriculture have a strong correlation, most are nonsensical. For example, Urban and Forests correlate by -0.9, the lowest given by the network, even though a sizeable number of images share the label. Regrettably, the Combined Label OLR performed similarly for the Orthoimages dataset as it did the PlanetLab dataset. It performed decently on the classifier but failed to find correlations with the Siamese model.

Chapter 6 – Evaluation

The OLR failed in finding accurate correlations within the Corine dataset, both combined with the PlanetLab dataset as well as combined with the Ortoimages dataset. The main reason for this appears to be the classifier networks inability to train on the dataset and learn to recognize multilabel satellite data. While it is impossible to know the exact cause of this without further research, it is possible to hypothesize about the plausible causes for this. It is also possible that it is not one, but a combination of these problems caused the classifier to fail in training. The most obvious direction to look at in assessing the failure of the network is the difference in the CelebA dataset and the Corine data(sub)set used in the research, as the first did succeed in yielding correlation while the latter did not. Several important distinctions can be pointed out between the datasets.

6.1 Size

To train a CNN network correctly, it requires a sizeable dataset to learn the structures which make up that which it wants to classify. For a multi-label dataset such as this one especially, the dataset needs to have various examples of each label as to learn to recognize said labels outside of the training context as well. The CelebA dataset originally used in OLR consisted of over 200000 images, giving the classifier network a big dataset to train and learn the intricacies of the data. In contrast, the PlanetLab dataset used in this research consisted of about 14100 images, and the OrthoImages dataset of 3500 after preprocessing. The classifier may have been unable to learn to recognize labels as it did not have enough examples per label. This is also exemplified by the fact that the label “Non-irrigated arable land”, the most common label in the dataset, appeared to be recognized somewhat by the PlanetLab classifier trained with random batches enabled. However, it must also be noted that there are numerous examples of (albeit small) neural networks learning to recognize data from fewer data points, and this explanation does not account for the apparent inability of the classifier network to improve its evaluations at all.

6.2 Label Division

While the amount of training examples is important for the network to learn to recognize labels, the quality of said examples is of even higher importance. In the CelebA dataset, labels are distributed across images in relatively high density. An image in the CelebA dataset often has between 6 and 12 labels associated with it, and all labels are well represented within the dataset. In contrast, the Corine dataset combined with the satellite image datasets has a very sparse label distribution. This is to be expected of a general-purpose satellite dataset that assesses only one label per area, as it distinguishes very broad categories of land into large areas. Ergo, big patches of land will only contain few associated labels. The PlanetLab dataset contained about 4 or fewer labels per image, Implying about 28 or more absent labels per image. This makes it harder for the network to train on said labels, especially those which are underrepresented. On top of this, it may lead the network to assess a big number of labels as simply always being absent to still achieve high accuracy.

However, several methods were applied in this research to the PlanetLab dataset to attempt to solve this problem, which was proved ultimately unsuccessful. Firstly, weighing classes appeared to have the opposite effect, discouraging the network from learning to recognize labels. Secondly, the purging of highly imbalanced labels, bringing the dataset from 32 to 16 labels in total, also did not change the performance of the network. This may imply that the imbalance of the data is not the actual cause of the classifiers inability to learn labels. However, the purging of these 16 uncommon labels does not change the fact that even the more commonly found labels are still relatively sparse, and in turn difficult to learn for the classifier. The third method which was attempted to deal with this issue is combining labels into broader categories. This method did see a slight success in improving the performance of the classifier on the dataset, as it was able to correctly classify about 61% of positive labels. It performed especially well on the Forests label, which was represented within the dataset the most, as well as getting decent results on the Urban, and Bodies of Water labels. These labels were not represented in the dataset much however, a possible explanation could be that they were highly recognizable by the classifier, which seems plausible considering the nature of these categories of land use. While this method improves classifier performance somewhat it seems, this does not directly translate into good OLR performance. Since the combining of labels within the dataset causes labels to co-occur within images less, it subsequently becomes harder to find correlations within the dataset. This was found for both the PlanetLab as well as the Orthoimages datasets.

The final methods which attempt to deal with the unbalanced data were focal loss and lowering the binary accuracy threshold. Both methods encourage the classifier to classify a label as being “present” more frequently. While this was achieved, the methods ultimately did not cause much improvement in the performance of the classifier, still getting the majority of present labels wrong, and outputting mainly absent label classifications.

6.3 Type of Classifier

The architecture of the classifier in combination with the dataset used is also of importance. The classifier used in this research was a slightly modified version of the CNN originally created and evaluated with the CelebA dataset. While it is a quite general image classification CNN in its structure, it is possible that this classifier simply works better for facial recognition as opposed to satellite image recognition. State of the art research also shows that the classification of general case multi labelled satellite data is still a tough challenge. On top of this, the metric used to evaluate the performance of the network may also play a role in its classification results. As binary accuracy rewards the network for getting as many labels correct as possible, it will learn to mostly output zeros. While there was an attempt to deal with this in the form of the lowered binary accuracy threshold, perhaps a different evaluation metric altogether may prove beneficial for the classification performance.

6.4 Satellite Datasets

The PlanetLab dataset suffered from quite unclear data. Most of the pixel values within the dataset lay close together, which may have caused details within the dataset to be difficult to pick up by the classifier. This is likely the result of the low resolution of the dataset. However, if this were the biggest problem within the evaluation of the image data, the Orthoimages dataset should have been able to train the classifier well, as this dataset has a much higher resolution. This is not the case, however, which implies that either resolution does not play a role within the classification of the data, or the Orthoimages dataset suffers from other issues as well (e.g. sparse labels, dataset size, classifier used) making it unable to utilize its higher resolution. As the Orthoimages dataset also put out mainly absent labels, the latter option seems more plausible. Only when other issues are resolved, is it possible to completely evaluate the importance of high definition within the dataset.

Another issue with combining independently collected datasets, as was done in this research, is that they are all collected at different times. Where the Corine dataset stems from 2018, the PlanetLab and Orthoimages datasets were collected in 2021 and 2012 respectively. The land may have changed between these periods of acquisition, making the datasets less accurate in themselves in their combination of land and labels. Still, it is assumed that this period will not have changed the land drastically enough to have a detrimental impact on the accuracy of the dataset, but it is an important concern nonetheless.

Chapter 7 - Conclusion and Discussion

This research set out to investigate whether OLR can be combined with geospatial data. Whether this is possible or not remains ambiguous, as OLR requires a well-trained classifier to be able to train its Siamese network. The type of classifier required by OLR, one which can accurately classify multi-labelled images, is challenging to apply to geospatial data. The nature of geospatial data is such that labels are divided over land sparsely, and labels may not always be recognizable from the satellite data. The dataset used in this research, the Corine dataset, is found to be too general-purpose, consisting of many labels of various categories of which most are underrepresented on a large scale. In retrospect, attempting to apply a general-purpose Multi-Label CNN to a land use dataset was ambitious, as this is a challenging task in itself that does not even involve the actual OLR yet, but rather is a step in the process of applying OLR. A better approach would have been to find a state-of-the-art classifier for a more specific subset of land data, which has already been shown to perform well and apply it to OLR. Not all is in vain, however, as the Combined Label classifier ran with the PlanetLab dataset managed to learn some correlations within the dataset, showing that there may be potential for OLR combined with land data yet, as long as labels are present frequently enough within the dataset. This is additionally exemplified by the PlanetLab classifier trained with random batches, which did learn to recognize the “Non-irrigated arable land” label, most likely since this label was most frequently present within the dataset.

The methodology with which the research was done has not always been up to desired standards. Much of the process consisted of trying new methods on the datasets just to see whether it would change anything. A more precise methodology would have been beneficial to the process, as it was observed that a consistent method of analyzing results made comparing said results easier. Many experiments were repeated near the tail end of the research to be able to compare these results, after all, leading to interesting insights. Had this way of working been applied from the start, these insights may have been discovered faster. It is however also inherent to this type of research to attempt many types of methods and techniques, as there are many out there which may help. This can cause the focus to be shifted from a strict methodology to an attempt to try as many methods in as short a timeframe as possible.

Related to this, the addition of the Orthoimages dataset was quite a last-moment effort to attempt at getting valid results. Due to this, the dataset could not be explored to its fullest potential like how it was done with the PlanetLab dataset. Currently, this dataset appeared to perform quite bad, but perhaps any of the methods applied to the PlanetLab dataset would have made a difference in this regard.

In conclusion: OLR could theoretically be applied to general-case geodata, but to do so a classifier that can recognize many different types of land would have to be developed first, as OLR requires a well working classifier before being able to train the Siamese network. Should such a classifier be developed, it would be interesting to repeat this experiment with said classifier and associated dataset and see whether it manages to operate successfully.

Chapter 8 - Future Research

The research in this thesis is non-conclusive, and as such the findings provide much space for future research. A number of directions for future research can be considered.

8.1 State-Of-The-Art Geospatial Classifier

To investigate whether OLR could be combined with geospatial data, it is recommended to find a state-of-the-art MLL classifier network made specifically for geospatial data. Ultimately a general-purpose classifier will likely provide the most interesting results if this research is done, as it may yield insight into label correlations that may not have been considered before. However, a contemporary network that is more specialized in a specific type of land data appears to be the more realistic option as of right now. This will still provide the deep learning community with the information of whether OLR can be applied to geospatial data, albeit a specific subtype. This may open the door towards a general-purpose classifier being applied to geospatial OLR at some point in the future. Additionally, may a strong general-purpose geospatial classifier be developed, it would be interesting to repeat this research and analyze its results. As this research did point out methods that can prove beneficial in the training of the classifier and Siamese network, these can be applied to future research as well.

8.2 Running OLR with different types of data

Since the only real criteria for OLR is that it combined the efforts of a CNN classifier network with an occlusion based Siamese network to find meaningful data representations and find correlations, it can be applied to many types of data, as long as it is multi labelled. The most obvious step is to keep within image data and attempt to find correlations within objects other than faces. Examples may include animals, plants, or any other image data for which correlations can prove useful. It is however also possible to apply this network to other forms of spatial data, such as audio or video. As of now Multi-Label Learning is less explored within these domains, so finding a good dataset may be difficult. Still, if the opportunity arises to run OLR with spatial data other than an image, it would be very interesting to see how the network performs.

8.3 Analyzing the Orthoimages dataset more thoroughly

Since the Orthoimages dataset could not be analyzed to its full potential, it could be interesting to see how it would perform if it ran with the various discussed methods enabled. This would also prove how important the resolution used is in the geolocation dataset, as it can be compared to this research which used a less resolute dataset for the most part. It is expected to perform better for its higher amount of detail, but it would require very powerful GPUs to fully explore the dataset as it is very large. Alternatively, an attempt can be made to lower the resolution by decreasing the image dimensions. This way, the dataset will still have a higher resolution compared to the PlanetLab dataset, but it will also be possible to fit more land, and thus more labels, on each image while still being runnable on a medium-range GPU.

Appendices

Appendix A: The Classifier Model

```
from tensorflow.keras import backend as keras
from tensorflow.keras.optimizers import *
import numpy as np
import os
import time
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.activations import *

batch_size = 64

gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)


def model1(input_size=(178, 178, 3)):
    input = Input(input_size)
    conv1_1 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(input)
    conv1_2 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(conv1_1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1_2)          # 87

    conv2_1 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(pool1)
    conv2_2 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(conv2_1)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2_2)          # 41

    conv3_1 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(pool2)
    #drop1 = SpatialDropout2D(0.5)(conv3_1)
    conv3_2 = Conv2D(64, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(conv3_1)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3_2)         # 18

    conv4_1 = Conv2D(128, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(pool3)
    ##drop2 = SpatialDropout2D(0.5)(conv4_1)
    conv4_2 = Conv2D(128, 3, activation='relu', padding='valid',
                    kernel_initializer='he_normal')(conv4_1)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4_2)         # 7
    conv5_1 = Conv2D(256, 3, activation='relu', padding='valid',
                     kernel_initializer='he_normal')(pool4)
    ##drop3 = SpatialDropout2D(0.5)(conv5_1)
    conv5_2 = Conv2D(256, 3, activation='relu', padding='valid',
                     kernel_initializer='he_normal')(conv5_1)  # 3
    # pool5 = MaxPooling2D(pool_size=(2, 2))(conv5_2)

    # conv6_1 = Conv2D(128, 3, activation='relu', padding='same',
    # kernel_initializer='he_normal')(pool5)

    # drop4 = SpatialDropout2D(0.5)(conv6_1)
    # conv6_2 = Conv2D(128, 3, activation='relu', padding='same',
    # kernel_initializer='he_normal')(conv6_1)
    # pool6 = MaxPooling2D(pool_size=(2, 2))(conv6_2)            # 3

    # embeddings = Conv2D(128, 3, activation='relu', padding='same',
    # kernel_initializer='he_normal')(conv6_1)

    embeddings_flat = Flatten()(conv5_2)
    #drop5 = Dropout(0.5)(embeddings_flat)
    dense1 = Dense(2000, activation='relu')(embeddings_flat)
    drop6 = Dropout(0.5)(dense1)
    dense2 = Dense(1000, activation='relu')(drop6)
    output = Dense(38, activation='sigmoid')(dense2)
    model = Model(inputs=input, outputs=output)

    model.compile(optimizer=Adam(lr=1e-4),
                  loss='binary_crossentropy', metrics=['accuracy'])

    model.summary()
    return model


model = model1()

TestSetData1 = (np.load('98.npy') /
                255.0).astype(dtype=np.float32)
TestSetData2 = (np.load('99.npy') /
                255.0).astype(dtype=np.float32)

TestSet = np.concatenate([TestSetData1, TestSetData2], 0)

TestSetLabels1 = np.load('labels98.npy')
TestSetLabels2 = np.load('labels99.npy')
# REMOVE LABELS Wearing_Necklace + Wearing_Necktie
TestSetLabels1 = np.delete(TestSetLabels1, [37, 38], axis=1)
TestSetLabels2 = np.delete(TestSetLabels2, [37, 38], axis=1)
TestSetLabels = np.concatenate([TestSetLabels1, TestSetLabels2], 0)
```

```
Best = 0
for epoch in range(100000000):
    for fileid in range(1, 98):
        fnameD = str(fileid) + '.npy'
        fnameL = 'labels' + str(fileid) + '.npy'
        X = np.load(fnameD)
        X = (X/255.0).astype(np.float32)
        Y = np.load(fnameL)
        # REMOVE LABELS Wearing_Necklace + Wearing_Necktie
        Y = np.delete(Y, [37, 38], axis=1)

        rm = True
        for i in range(np.shape(X)[0]//batch_size):
            indices = np.random.choice(np.shape(X)[0], batch_size)
            trn = model.train_on_batch(
                x=X[indices, :], y=Y[indices, :], reset_metrics=rm)
            rm = False
    if epoch % 1 == 0:
        flag = 0
        results = model.evaluate(
            TestSet, TestSetLabels, batch_size=100, verbose=0)
        if results[1] > Best:
            Best = results[1]
            model.save('Model.h5')
            flag = 1
            #prediction = model.predict(TestSet)
            #np.save('prediction', prediction)
            #np.save('predictionLabels', TestSetLabels)

        print('epoch ' + str(epoch) + '  TRN: ' +
              str(trn[0]) + ' ' + str(trn[1]) + '   TST: ' + str(results), '  *' if flag == 1 else ' ')
```

Appendix B: The Siamese model

```
from tensorflow.keras import backend as keras
from tensorflow.keras.optimizers import *
import numpy as np
import os
import sys
import time
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.activations import *

batch_size = 100
OcclusionWindowMin = 120
OcclusionWindowMax = 130
```

```
learning_rate = 0.001

gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)


def dp(vests):  # [None,40,64]
    x, y = vests
    return keras.sum(x * y, axis=-1, keepdims=True)


def cosine_distance(vests):  # [None,40,64]
    x, y = vests
    x = keras.l2_normalize(x, axis=-1)
    y = keras.l2_normalize(y, axis=-1)
    return -keras.mean(x * y, axis=-1, keepdims=True)


def CreateBatch(data, labels, bs):

    Indexes = np.random.choice(np.shape(data)[0], 2*bs, replace=False)
    Occlusionx = np.random.choice(np.shape(data)[1], 2*bs, replace=True)
    Occlusiony = np.random.choice(np.shape(data)[2], 2*bs, replace=True)
    OcclusionWindowX = np.random.randint(
        low=OcclusionWindowMin, high=OcclusionWindowMax, size=2*bs)
    OcclusionWindowY = np.random.randint(
        low=OcclusionWindowMin, high=OcclusionWindowMax, size=2*bs)

    Images = data[Indexes]
    labels = np.zeros([bs, 38])
    for i in range(len(Occlusionx)):
        Images[i, Occlusionx[i]:Occlusionx[i] + OcclusionWindowX[i],
               Occlusiony[i]:Occlusiony[i] + OcclusionWindowY[i], :] = 0

    sourceImages = Images[:bs]
    targetImages = Images[bs:]

    predictSI = classifier.predict_on_batch(sourceImages)
    predictTI = classifier.predict_on_batch(targetImages)

    labels = predictSI * predictTI

    return sourceImages, targetImages, labels


def model1(input_size=(178, 178, 3)):
    inputX = Input(input_size)
    inputZ = Input(input_size)

    model = Sequential()
```

```
model.add(Conv2D(64, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(Conv2D(64, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))          # 87

model.add(Conv2D(128, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(Conv2D(128, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))          # 41

model.add(Conv2D(128, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
# model.add(SpatialDropout2D(0.5))
model.add(Conv2D(256, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))          # 18

model.add(Conv2D(256, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
# model.add(SpatialDropout2D(0.5))
model.add(Conv2D(256, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))          # 7

model.add(Conv2D(512, 3, activation='relu', padding='valid',
            kernel_initializer='he_normal'))
# model.add(SpatialDropout2D(0.5))
model.add(Conv2D(4864, 3, activation='relu', padding='valid',  # 38 (classes) x128 = 4864
            kernel_initializer='he_normal'))

encode1 = model(inputX)
encode2 = model(inputZ)

x1 = GlobalAvgPool2D()(encode1)
x2 = GlobalAvgPool2D()(encode2)

features1 = Reshape([38, 128])(x1)  # [None, 38,128]
features2 = Reshape([38, 128])(x2)

DP = Lambda(dp)([features1, features2])

ModelSiamese = Model([inputX, inputZ], DP)

ModelSiamese.compile(optimizer=RMSprop(lr=learning_rate),
              loss='mean_squared_error')

ModelSiamese.summary()
return ModelSiamese
```

```
siamese = model1()
print('Loading Saved Model...')
classifier = tf.keras.models.load_model('Model.h5')
classifier.summary()
print('\n\n\n\n\n')

TestSetData1 = (np.load('98.npy') /
        255.0).astype(dtype=np.float32)
TestSetData2 = (np.load('99.npy') /
        255.0).astype(dtype=np.float32)
#TestSetData1 = np.reshape(TestSetData1, [np.shape(TestSetData1)[0], 64, 64, 3])
#TestSetData2 = np.reshape(TestSetData2, [np.shape(TestSetData2)[0], 64, 64, 3])

print(np.shape(TestSetData1), np.shape(TestSetData2))
TestSet = np.concatenate([TestSetData1, TestSetData2], 0)
TestSetPreds = classifier.predict(TestSet)

TestSetLabels1 = np.load('labels98.npy')
TestSetLabels2 = np.load('labels99.npy')
TestSetLabels = np.concatenate([TestSetLabels1, TestSetLabels2], 0)
TestSetLabels = np.delete(TestSetLabels, [37, 38], axis=1)

Best = 100000
for epoch in range(100000000):

    if os.path.exists('learningRate.txt'):
        f = open('learningRate.txt')
        l = f.readline().splitlines()
        newl = np.float(l[0])
        if learning_rate != newl:
            learning_rate = newl
            keras.set_value(siamese.optimizer.learning_rate, learning_rate)
            print('Learning rate = '+str(newl))

    trainLoss = []
    for fileid in range(1, 98):
        fnameD = str(fileid) + '.npy'
        fnameL = 'labels' + str(fileid) + '.npy'
        X = np.load(fnameD)
        data = (X/255.0).astype(np.float32)
        #data = np.reshape(X, [np.shape(X)[0], 64, 64, 3])
        labels = np.load(fnameL)
        labels = np.delete(labels, [37, 38], axis=1)

        for i in range(np.shape(data)[0]//batch_size):
            X, Z, Y = CreateBatch(data, labels, batch_size)
            trn = siamese.train_on_batch(
                x=[X, Z], y=Y, reset_metrics=True)
            trainLoss.append(trn)
    trn = np.mean(trainLoss)

    if epoch % 1 == 0:
        tstLoss = []
        flag = 0
        for i in range(np.shape(TestSet)[0]//batch_size):
            X, Z, Y = CreateBatch(TestSet, TestSetLabels, batch_size)
            test = siamese.test_on_batch(x=[X, Z], y=Y)
            tstLoss.append(test)

        test = np.mean(tstLoss)
        if Best > test:
            Best = test
            flag = 1
            siamese.save('siamese.h5', siamese)

    print('epoch ' + str(epoch) + '  TRN: ' +
          str(trn) + '  TST: ' + str(test), '  *' if flag == 1 else ' ')
```

Appendix C: Label Divisions

1: Standard label division PlanetScope

1: Continuous urban fabric

2: Discontinuous urban fabric

3: Industrial or commercial units

4: Road and rail networks and associated land

5: Port areas

6: Airports

7: Mineral extraction sites

8: Dump sites

9: Construction sites

10: Green urban areas

11: Sport and leisure facilities

12: Non-irrigated arable land

13: Permanently irrigated land

14: Vineyards

15: Fruit trees and berry plantations

16: Olive groves

17: Pastures

18: Annual crops associated with permanent crops

19: Complex cultivation patterns

20: Land principally occupied by agriculture with natural vegetation

21: Broad leaved forest

22: Coniferous forest

23: Mixed forest

24: Natural grasslands

25: Scleropyllous vegetation

26: Transitional woodland shrub

27: Beaches dunes sand

28: Bare rocks

29: Sparsely vegetated areas

30: Salt marches

31: Water bodies

32: Sea and ocean

2: Pruned Label Division PlanetScope:

2: Discontinuous urban fabric

3: Industrial or commercial units

10: Green urban areas

12: Non-irrigated arable land

13: Permanently irrigated land

15: Fruit trees and berry plantations

16: Olive groves

18: Annual crops associated with permanent crops

19: Complex cultivation patterns

20: Land principally occupied by agriculture with natural vegetation

22: Coniferous forest

24: Natural grasslands

25: Scleropyllous vegetation

26: Transitional woodland shrub

29: Sparsely vegetated areas

32: Sea and ocean

3: Combined Label Division PlanetScope:

1: Urban
2: Special Sites
3: Non-irrigated arable land
4: Agriculture
5: Forests
6: Rocky / Mountainous areas
7: Bodies of water

4: Standard Label Division Orthoimages

1: Continuous urban fabric
2: Discontinuous urban fabric
3: Industrial or commercial units
4: Road and rail networks and associated land
5: Port areas
6: Airports
7: Mineral extraction sites
8: Dump sites
9: Construction sites
10: Green urban areas
11: Sport and leisure facilities
12: Non-irrigated arable land
13: Permanently irrigated land
14: Vineyards
15:Fruit trees and berry plantations
16: Olive groves
17: Pastures
18: Annual crops associated with permanent crops
19: Complex cultivation patterns
20: Land principally occupied by agriculture with significant areas of natural vegetation
21: Broad-leaved forest
22: Coniferous forest
23: Mixed forest

24: Natural grasslands

25: Moors and heathland

26: Sclerophyllous vegetation

27: Transitional woodland-shrub

28: Beaches dunes sands

29: Bare rocks

30: Sparsely vegetated areas

31: Burnt areas

32: Inland marshes

33: Salt marshes

34: Sea and ocean

5: Combined Label Division Orthoimages

1: Urban

2: Agriculture

3: Forests

4: Rocky / Mountainous areas

5: Bodies of water

References

[1] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, “Recent advances in convolutional neural network acceleration,” *Neurocomputing*, vol. 323, pp. 37–51, 2019, doi: 10.1016/j.neucom.2018.09.038.

[2] J. Gu *et al.*, “Recent advances in convolutional neural networks,” *Pattern Recognit.*, vol. 77, pp. 354–377, 2018, doi: 10.1016/j.patcog.2017.10.013.

[3] X. Geng, “Label Distribution Learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1734–1748, 2016, doi: 10.1109/TKDE.2016.2545658.

[4] S. Karatsiolis and A. Kamilaris, “Converting image labels to meaningful and information-rich embeddings,” in *ICPRAM 2021 - Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods*, 2021, pp. 107–119, doi: 10.5220/0010375801070119.

[5] R. Shao, N. Xu, and X. Geng, “Multi-label Learning with Label Enhancement,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2018-Novem, pp. 437–446, 2018, doi: 10.1109/ICDM.2018.00059.

[6] X. Rong, “word2vec Parameter Learning Explained,” *CoRR*, vol. abs/1411.2, 2014, [Online]. Available: http://arxiv.org/abs/1411.2738.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” 2013, Accessed: May 14, 2021. [Online]. Available: http://ronan.collobert.com/senna/.

[8] M. L. Zhang and Z. H. Zhou, “A review on multi-label learning algorithms,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1819–1837, 2014, doi: 10.1109/TKDE.2013.39.

[9] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” *Proc. - 2005 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition, CVPR 2005*, vol. I, pp. 539–546, 2005, doi: 10.1109/CVPR.2005.202.

[10] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1701–1708, 2014, doi: 10.1109/CVPR.2014.220.

[11] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” *Proc.*

IEEE Int. Conf. Comput. Vis., vol. 2015 Inter, pp. 3730–3738, 2015, doi: 10.1109/ICCV.2015.425.

[12] M. Drusch *et al.*, “Sentinel-2: ESA’s Optical High-Resolution Mission for GMES Operational Services,” *Remote Sens. Environ.*, vol. 120, pp. 25–36, 2012, doi: https://doi.org/10.1016/j.rse.2011.11.026.

[13] G. Büttner, J. Feranec, G. Jaffrain, L. Mari, G. Maucha, and T. Soukup, “The CORINE land cover 2000 project,” *EARSeL eProceedings*, vol. 3, no. 3, pp. 331–346, 2004, Accessed: May 14, 2021. [Online]. Available: http://terrestrial.eionet.eu.int.

[14] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov, “Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data,” *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 5, pp. 778–782, 2017, doi: 10.1109/LGRS.2017.2681128.

[15] R. Khatami, G. Mountrakis, and S. V Stehman, “A meta-analysis of remote sensing research on supervised pixel-based land-cover image classification processes: General guidelines for practitioners and future research,” *Remote Sens. Environ.*, vol. 177, pp. 89–100, 2016, doi: https://doi.org/10.1016/j.rse.2016.02.028.

[16] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, “Deep Learning-Based Classification of Hyperspectral Data,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 7, no. 6, pp. 2094–2107, 2014, doi: 10.1109/JSTARS.2014.2329330.

[17] W. Zhao and S. Du, “Learning multiscale and deep representations for classifying remotely sensed imagery,” *ISPRS J. Photogramm. Remote Sens.*, vol. 113, pp. 155–165, 2016, doi: https://doi.org/10.1016/j.isprsjprs.2016.01.004.

[18] B. R. Felton, G. L. O’Neil, M. M. Robertson, G. M. Fitch, and J. L. Goodall, “Using random forest classification and nationally available geospatial data to screen for wetlands over large geographic regions,” *Water (Switzerland)*, vol. 11, no. 6, 2019, doi: 10.3390/w11061158.

[19] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi, “Training CNNs with Low-Rank Filters for Efficient Image Classification,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, Nov. 2015, Accessed: Apr. 11, 2021. [Online]. Available: http://arxiv.org/abs/1511.06744.

[20] I. Shendryk, T. C. Scientific, C. Ticehurst, and T. C. Scientific, “Deep Learning - a New Approach for Multi-Label Scene Classification in Planetscope and Sentinel-2 Imagery,” no. June 2019, 2018, doi: 10.1109/IGARSS.2018.8517499.

[21] V. Divarak, “Understanding Backpropagation,” Nov. 19, 2018. https://blog.quantinsti.com/backpropagation/ (accessed Apr. 05, 2021).

[22] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks,” doi: 10.1145/2001269.

[23] “CLC 2018 — Copernicus Land Monitoring Service,” May 04, 2021. https://land.copernicus.eu/pan-european/corine-land-cover/clc2018 (accessed May 14, 2021).

[24] “Land use & land use change.” http://www.clima-project.eu/case-studies/faleriinovi/land-use/ (accessed Jun. 22, 2021).

[25] “PlanetScope - Earth Online.” https://earth.esa.int/eogateway/missions/planetscope (accessed Jun. 28, 2021).

[26] “Republic of Cyprus | Department of Land and Surveys.” https://portal.dls.moi.gov.cy/en-us/homepage (accessed Jul. 28, 2021).

[27] “Classification on imbalanced data | TensorFlow Core.” https://www.tensorflow.org/tutorials/structured_data/imbalanced_data (accessed Jun. 27, 2021).

[28] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection.” 2018.