Neural Networks 2018

Assignment 3

Group 20

May 16, 2018

1 Introduction

Remote sensing data acquisition in recent years, such as satellite and aerial imagery, has been a significant global breakthrough in enhancing our understanding of computer vision applications. *Google maps*[1] provides an immense load of geodesic data that can be exploited to automate procedures which were done before manually. In addition, the recent technological developments, drones for example, are capable of generating aerial imagery data that can be subject of some useful applications. However, extracting useful information from such images is an often-encountered problem in computer science. To this extend, the continuous modification of the urban areas require robust methods that would be skillful to keep track of these frequent changes with a minimal computational cost. Despite the complexity of the task, over the last few decades, *Neural Networks* have risen to become the superior algorithm to tackle this problem (Mokhtarzade and Zoej, 2007).

P. Doupe proposed a *Convolutional Neural Network*[2] that is capable of estimating the population of an area based on the landscape infrastructure. The *LANDSAT* prototype predicts the population of a given location in the USA from satellite images and the results are compared with the *CENSUS* database to estimate the accuracy of the method. Another application of *Convolutional Neural Networks* is applied on Geodesic data to identify water sources on satellite images[3]. The algorithm is able to detect the presence of water in the satellite images with an accuracy of 96%. Another fascinating application of *Neural Networks* on geodesic images is street number recognition. An early attempt of Teschioni, Oberti, and Regazzoni (1999) using *Neural Networks* aimed to detect moving objects in color sequenced images. Their goal was not only to detect moving objects but also to predict dangerous situations according to their behaviour. Goodfellow et al. (2013) proposed an algorithm which can automatically detect the street numbers from panoramic Google street view imagery. Although breakthrough techniques have been developed in *Optical Character Recognition (OCR)*, identifying multi-text in street images proves to be a challenging task due to the many parameters that can affect it, such as text characteristics, environmental conditions, acquisition factors et cetera.

In this project, we will attempt to apply *Semantic Segmentation* on *Google maps* satellite images using different architectures of *Neural Networks*. The aim of the project is to train a *Neural Network* to automatically detect roads and generate road-maps from satellite images.

2 Project idea

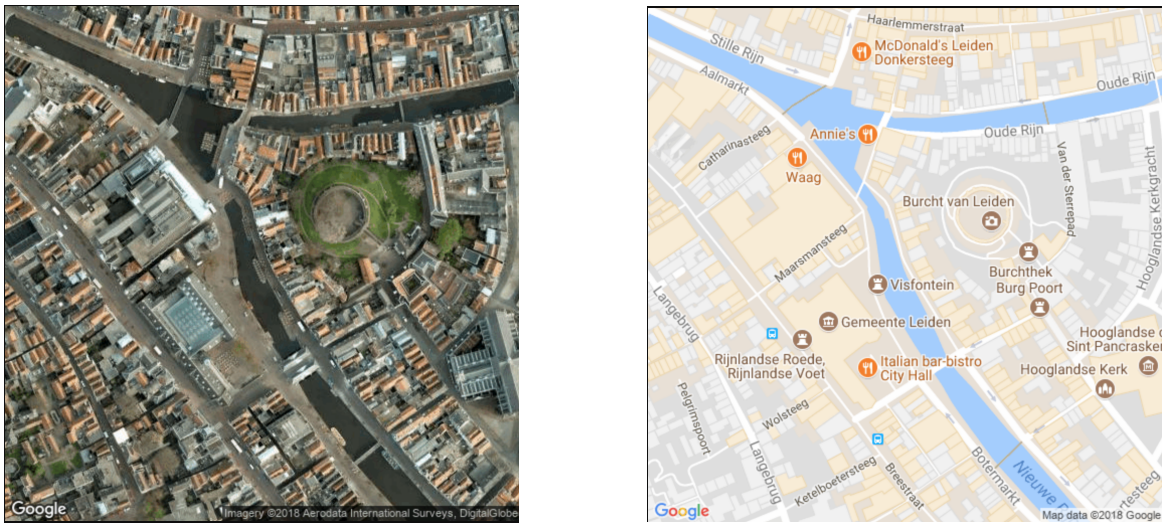
The purpose of this project is to develop an algorithm that is able to automatically detect the road network of a given satellite image. Satellite images provide a wide range of information that describe the landscape of a given location, containing residential and industrial areas, road networks, the natural environment and so on. Exploiting all this information, one can extract useful patterns that can be interpreted in a such a way that could be helpful in the question at hand. In this project we use color satellite images and with the aid of *Neural Networks* we attempt to identify the road network in each image based on the training images. The training images include both satellite and road network imagery from the same geographic coordinates as it is shown in Figure 1. Based on the traits of each pixel in the satellite image and having as a reference the road network image of that specific location, the algorithm learns to detect the existence or absence of streets and generate a new image accordingly. An additional attribute that is learned from our algorithm is that it also learns to detect the presence of water in the satellite images and identify it in the produced road network images. To obtain the

[1] `https://cloud.google.com/maps-platform/`

[2] `https://blog.insightdatascience.com/exploring-deep-learning-on-satellite-data-a17bf11781dc`

[3] `https://github.com/treigerm/WaterNet`

required outcome in this project we experimented with a number of different Neural Network architectures in order to achieve the most accurate predictions.



(a) Satellite view.

(b) Road network view.

Figure 1: Images *a* and *b* show an example of the task at hand where image *a* shows the satellite view of one location whilst image *b* depicts the road network which is the desired outcome of our approach.

3 Data

The first, and often most difficult obstacle faced in computer science projects is to acquire the desired data. The data used in this project were acquired from the *Google Maps Platform*[4] provided by the *Google Developers*. The images are two dimensional RGB static images cut out from an area of approximately 400,000km^2 starting from Leiden to East of Warsaw and the French city of Nancy to the border of Ukraine, Slovakia and Hungary as it is depicted in Figure 2. Later, an additional area was added in order to increase the amount of training data and consequently the accuracy of our model. As it is shown in Figure 3 this area is approximately 45,000km^2 from the North-East of Birmingham to the East coast and down to Eastbourne and to the South of Bristol.

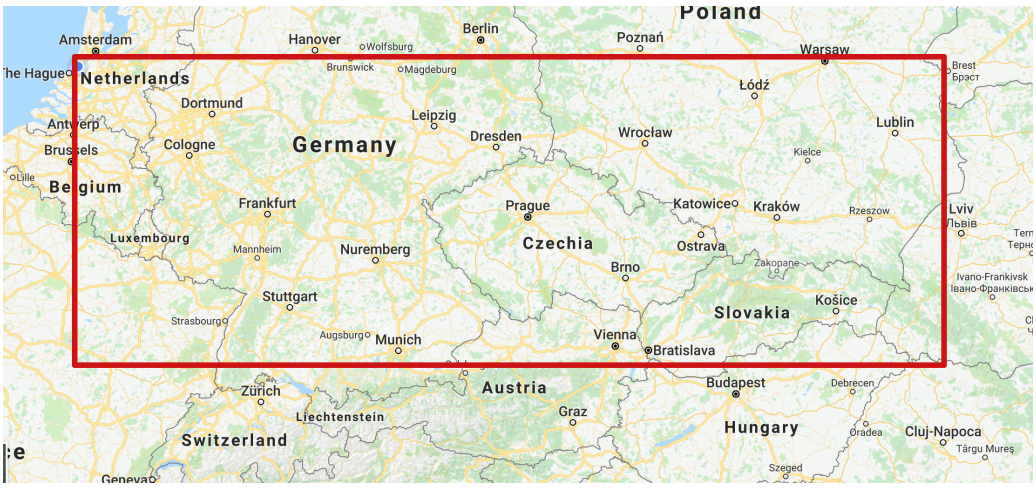


Figure 2: This image depicts the area used to collect the first, part of the second, and the third dataset. The first dataset contains 12,000 28×28 images in this area. 24,000 images of size 200×200 were collected from this area for the second dataset. And lastly, the third dataset has 12,100 512×512 images in this part of Europe.

From these areas, three separate data sets were collected. The first is a dataset consistent of 28×28 pixels Red, Green and Blue (RGB) images. This dataset contains 12,000 examples. The second dataset contains 36,000

[4] `https://cloud.google.com/maps-platform/`

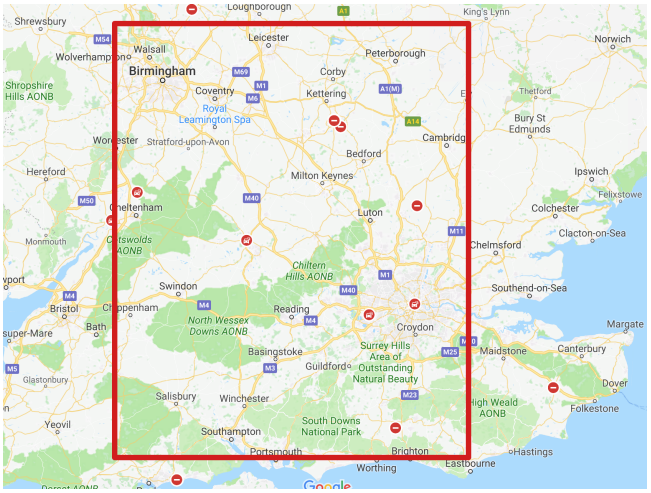


Figure 3: The image depicts the area used to collect additional data for our second dataset. 12,000 additional training examples were downloaded of size 200×200. The images cover an area of approximately 45,000km^2 .

200×200 pixels RGB images. The third and last dataset contains 512×512 pixels images and has 12,100 examples. All the captured images are taken at zoom level 15. The first and last dataset were collected from the first area as it is depicted in Figure 2. The 200×200 images dataset (24,000 images) were collected from this area as well, but to increase the sample size another area was added. As it is show in Figure 3, 12,000 additional images were later collected from the UK.

4 Methods

Our approach to this problem is as follows. We start by designing a simple *MLP-auto-encoder* to work on 28x28 pixel images, to get a baseline for the performance of our network and see whether there is a learn-able connection between the input and output images. Secondly, we will enlarge our first network to receive 200x200 pixels as input, and evaluate its performance. Finally, we will switch completely to a *Convolutional Neural Network*, and evaluate this from the ground up again. That is, we will train a CNN on the small 28×28 images, upgrade it to 200×200 images, and finally test it on 512×512 images. Unless otherwise specified, the number of training images is always split into 1/4 for testing (which we call validation) and 3/4 for training.

4.1 Multiclass perceptron

A multi-class perceptron is an extended version of the simple perceptron, which is a binary classifier. The perceptron takes as input n variables $(x_1, ..., x_n)$, computes the weighted sum of these inputs to a 'node', after which a certain activation function (e.g the Heaviside step function) determines whether the value in the node becomes either a 0 or a 1. In this way, a certain example i with features $(x_1, ..., x_n)$ can be classified using n input parameters.

In case there are more than two possible outcomes, for example in digit classification, we can generalize this approach by essentially stacking multiple perceptrons on top of each other. If we stack as much perceptrons as there are classes then each output node j $(0 < j < 10)$ can be viewed as giving a probability of some sort that the given class is digit j. In the case of outputting images, a multi-class perceptron would need as much output nodes as there are pixels. This can be viewed as a continuous classification problem for every pixel in the output image, i.e. the network has to classify a value for every output class (pixel).

We will start by using the 28×28×3 images with a very simple MLP. We flatten the images to an array of 2352 values. The network architecture is then very simple, the input layer is a dense layer of 2352 input values and 294 output values (nodes), and uses the *relu* activation function. As second (and last) layer a dense layer with 2352 nodes and *sigmoid* activation function is used, to be able to reproduce the final 28×28×3 image with pixel values between 0 and 1, as it is required. This very simple network already has 1,385,622 trainable parameters, due to the dense layers with the high amount of nodes required for this task.

The second step is to increase the image size to 200x200 pixels. This increases the number of trainable parameters immensely, due to the fact that both the input and output layer must now be of dimensionality 200×200×3 = 120,000. We have decided to make the hidden layer contain 58 nodes, since this roughly the largest amount possible while still being able to train the network in memory. The increase in the total number of trainable parameters is about fourteen-fold, to 14,040,058.

4.2 Convolutional Neural Network

Convolutional Neural Networks are a special kind of *Neural Network* that were influenced from the animal visual cortex. As the name suggests, a convolutional layer in a network consists of a filter, or kernel that is convoluted with the input image, after which a feature map is produced. *Convolutional Neural Networks* are characterized for their ability in visual recognition with minimal pre-processing. In this project we use two different different CNN architectures.

Initially we apply a convolutional Neural Network consistent of three layers. The first layer takes the input images of shape (28,28,3) and applies 16 5×5 kernels, with the *relu* activation function. The second and third contain respectively 8 and 3 kernels of size 5×5 with activation function *relu* and *sigmoid*, to make sure our output image has 3 filters, with values between 0 and 1, as is required for RGB images. All images are padded so that the size after the convolutional layers stays the same (in Keras: padding='same'), since we require an output image which has the same size as the input image. The total number of trainable parameters is 5,027, much lower than the MLP as this architecture does not contain any dense layers.

In the next attempt we use images with size 200×200×3 and size 512×512×3. Since the complexity of the images increases exponentially with increasing image size, we decide to define a deeper architecture. The architecture now consists of a total of 6 convolutional layers, all with padding='same', where in the first five layers we apply the *relu* activation function with kernels of size 5×5. The amount of kernels is as follows: 8, 16, 32, 16, 8. The final layer must again contain 3 kernels and the *sigmoid* activation function, to make sure we have an output image of size 200×200×3 or 512×512×3. The same kernel size of 5×5 is used in the final layer.

5 Results

In our initial experiments we tried different variations of optimizers[5] and loss functions[6], among which are the RMSprop, adam and adadelta optimizer and the mean squared error, mean absolute error and binary cross-entropy loss. The best initial results were found with a combination of the RMSprop optimizer and per-pixel binary cross-entropy loss function. Therefore, every architecture in this section will be compiled with the RMSProp optimizer, set to a learning rate of 0.001 and will use per-pixel binary cross-entropy loss.

5.1 MLP - 28x28

The network is trained with 12,000 images for 500 epochs, and outputs a randomly chosen predicted image from the training set every 50 epochs to see how the algorithm is doing. The training and test loss can be viewed in Figure 4 and the output of the network from epoch 200 to 450 can be viewed in Appendix I, Figure 10. As can be concluded from both of these images, the training loss converges rather quickly to about 0.369, while the test loss oscillates around 0.368. In imagery, this means that the algorithm can identify the general colour of the output (see e.g. Figure 10d). But the network clearly has trouble identifying shapes and does not even attempt to identify roads. In consecutive runs of the algorithm, there are a few interesting examples, the best of which are shown in Figure 5. From the best cases, it is clear this algorithm is equipped to identify water features, and the general color of the images, but it is still far from identifying roads.

[5] `https://keras.io/optimizers/`
[6] `https://keras.io/losses/`

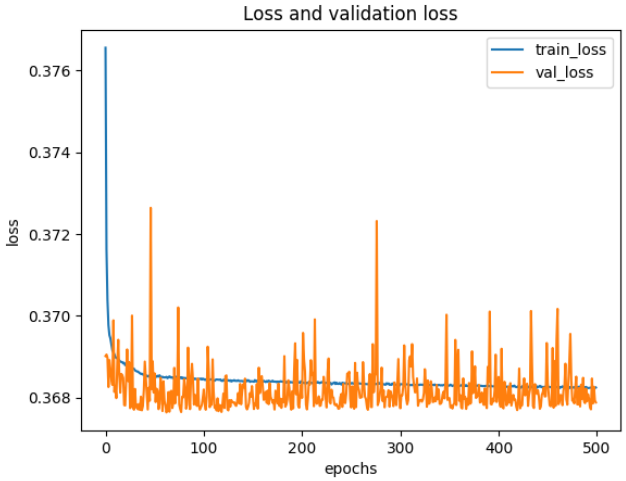
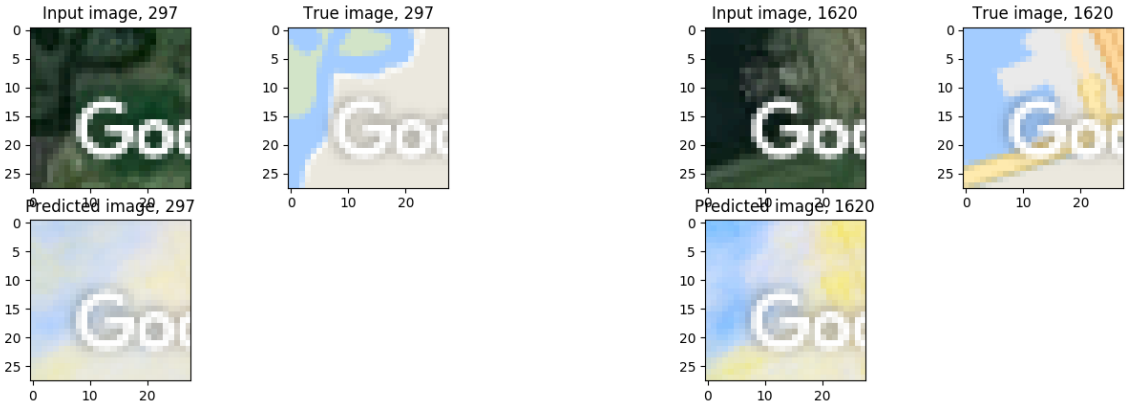
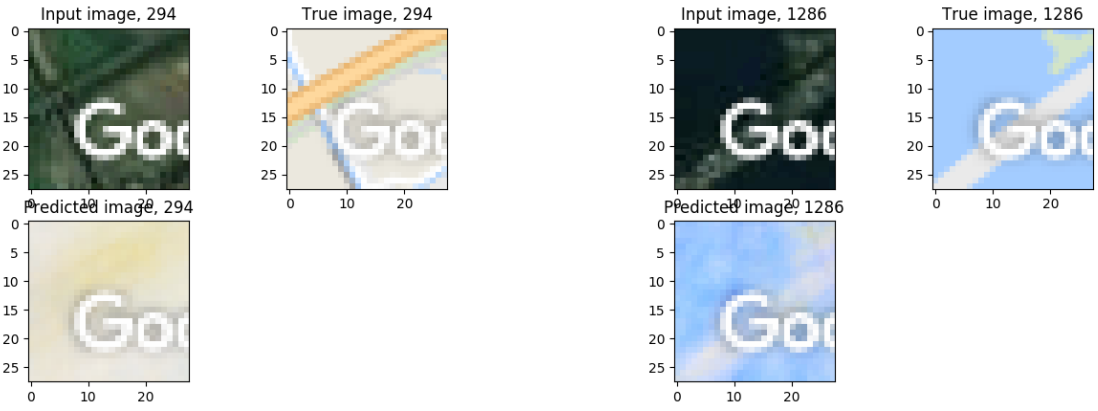


Figure 4: Loss and validation Loss for MLP in 12.000 28 × 28 images and 500 epochs.



(a) (b)



(c) (d)

Figure 5: Best predictions of the simple MLP on 28×28 pixel images.

5.2 MLP - 200x200

The steep increase in the number of parameters proves a difficult task for the MLP architecture. Even when using twice as much input examples, 24,000, and 1,000 epochs, the output of the network does not make sense. Since the number of trainable parameters (and the compression factor) is so high the network cannot learn correctly from this (in comparison to the number of parameters) relatively small sample of input images. This

Figure 6: Predicted 200×200 image after 1000 epochs of training, it is clearly visible that the input image shows no correlation with the output image, and the output image is some sort of averages of roads that the network has learned.

causes the network to output averages of true images independent of the input image. An example is shown in Figure 6, where the input and output of the network show no overlap. It is clear that the output image is some average of images that the network has seen. Apparently, the best way for the network to minimize the loss, given this many trainable parameters, is to produce an average output image independent of the input.

5.3 CNN - 28x28

The loss function for a 1,000 epochs of training on 12,000 small images is shown in Figure 7. The loss is shown to be converging at the same value as the MLP, but the validation loss is much more stable, indicating better generalization. The output image every 50 epochs, of the last 300 epochs can be viewed in Appendix II, Figure 11. From this figure it becomes clear that a CNN performs much better in shape detection than MLPs, as can be expected. However, the images seem too small for roads to be detected, which is why we will swiftly move on to the larger images.

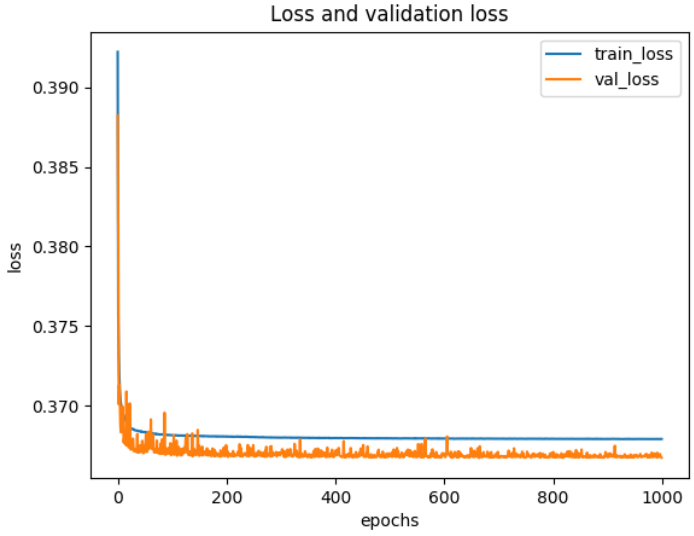


Figure 7: Loss and Validation Loss for CNN with 12.000 28×28 images trained in 1000 epochs

5.4 CNN - 200x200

This CNN is more complex than the network in the previous subsection, as previously stated in Section 4.2. Therefore, 36,000 images are used as training input and the network is trained for 100 epochs, as the deeper network slows down training significantly. The loss function, shown in Figure 8, seems to be converged after 100 epochs, and the same applies to the validation loss. Some of the best predictions are shown in Appendix III, Figure 12. These predictions show that the convolutional neural network architecture is very efficient at retrieving the shapes that are required and deleting the unnecessary information from the satellite images. The roads are still more vaguely colored than in the true images, but in most cases, the road is identified and colored whiter than the surrounding gray map.

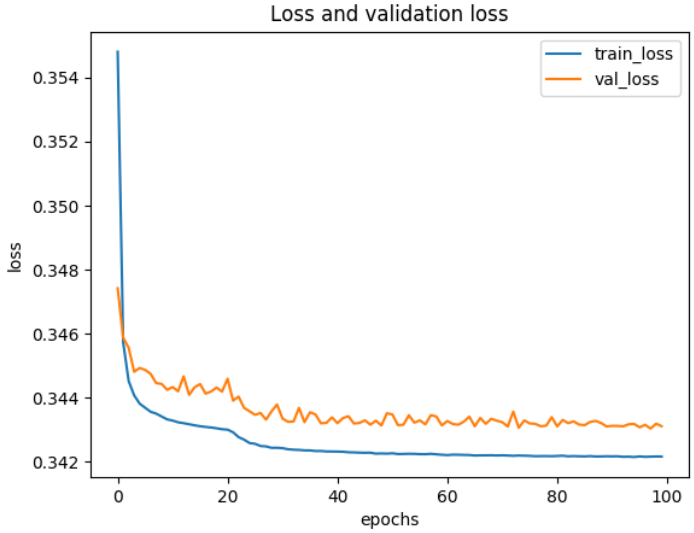


Figure 8: Loss and Validation Loss for CNN with 36.000 200×200 images trained in 100 epochs

As a final test for this network, we will download 200x200 images that the network has never seen before, 100 from Leiden and 100 from Amsterdam. These images naturally contain a high density of streets and houses, so this will validate the performance of the network on highly populated areas. As can be immediately seen from Appendix IV, Figure 13, the network performs much poorer than on the training images. The roads that the network identifies are still in the correct place, but are much vaguer than in the training images. Besides that, even obvious water features are lost in the output images. For the complete set of predictions on this validation set we kindly direct the interested reader to the following link, `https://imgur.com/a/MHxFBVu`.

5.5 CNN - 512x512

Finally, the largest images are fed to the network. Since time is finite, and the memory of the available GPUs is finite as well, we are only able to train the network with 8,800 of our input examples, for 50 epochs. The final loss function plot of this paper is shown in Figure 9. In accordance with Figure 8, it seems that larger images provide a lower final loss value. In this case, the validation loss converges neatly to the same value as the training loss, showing that the network can generalize very effectively. As the predictions of every 5 epochs for the last 30 epochs shows, in Appendix V, Figure 14, a lower loss value does not necessarily translate immediately to a higher effectiveness in recognizing roads. The network seems to be able to predict only the largest features given in the input image. It is most likely that this is due to the relatively low number of input images. To get the same result as the 200×200 pixel images, we would need at least *more* input examples than in the previous section, but we have trained the network with less, so a lower accuracy is in line with expectations. A separate validation set of Leiden and Amsterdam is shown to this network as well, and we ask the interested reader to view this at the following link: `https://imgur.com/a/C88Y5Fn`.

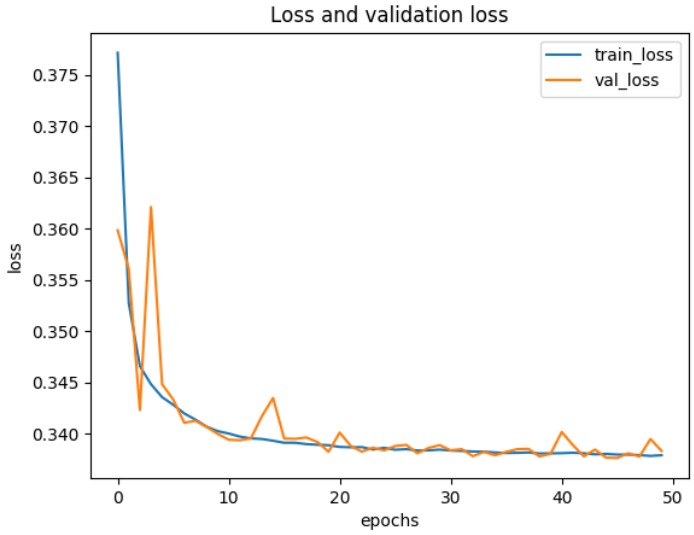


Figure 9: Loss and Validation Loss for CNN on 512×512 images.

6 Conclusions

In this project we experimented with a number of different *Neural Network* architectures in order to predict the road maps from satellite images. Thanks to the *Google Developers*, we obtained three different data-sets of satellite images and road maps that cover an area of $445.000Km^2$, each of them with different image sizes. Then, we applied different *Neural Network* architectures and setups in order to obtain the most accurate model capable of identifying the streets from the satellite images.

Initially we experimented with *Multi-Class Perceptrons*. The setup of the first MLP network had 12.000 28×28 images trained for 500 epochs. This network seemed to learn the basic colors of the output image, but failed to predict any streets. The MLP network trained on 24,000 200×200 images and 1,000 epochs seemed to overtake the barrier the previous setup faced, however it was not able either to predict a single image, but a concatenation of many training images. We attribute this failure to the large number of tune-able parameters and very high compression factor, combined with a relatively small training set.

Next, we used the more complicated architecture of *Convolutional Neural Networks* first with 12,000 28×28 images trained for 1000 epochs and then with 36,000 200×200 images and 100 epochs. The first model, similar to the MLP, learns the basic structure of the image but it cannot predict the presence of the roads. In contrast the second experiment outperforms all the previous experiments conducted. This algorithm is capable of detecting most of the roads in the satellite image although the quality of the output images is a bit blurry. Looking closer at the output images, one can see that the places where the algorithm fails to detect the streets are those that even humans have difficulty to identify, due to shadows or barriers that cover the streets. The last experimental setup we tried used 8,800 512×512 images. It has obtained some promising results although they were not sufficient for our goal. This is mainly because of the time shortage and the availability of data. We conclude that the amount of training data is of crucial importance to produce highly accurate results, and the amount of required data is heavily dependent on the resolution of the image. The final conclusion is that the architecture used for the 200×200 images seems effective enough to make sensible predictions on the trained images. When showing a separate validation set the network performs more poorly, but still identifies some roads in satellite images. We believe that the CNN architecture is the right one for this problem, and the network could be greatly improved by adding more training examples.

Some hot-spots that need to be improved in our approach are the environmental - urban obstacles, like trees, building, shadow that prevent our algorithm to detect a continues road. This can be solved by connecting "continuous lines" that are at some points broken. In addition, the dataset has to be extended to different cities which have different morphological landscapes to improve the overall accuracy and generalizing ability of the network.

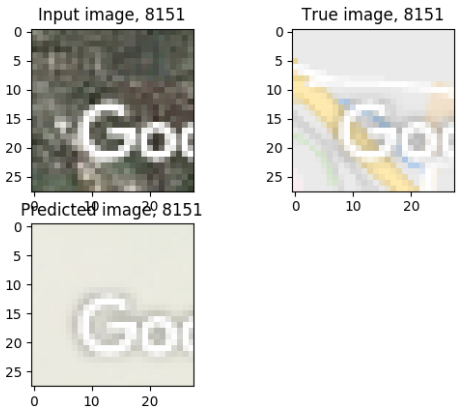
References

Goodfellow, Ian J. et al. (2013). "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: pp. 1–10. arXiv: 1312.6082. URL: http://arxiv.org/abs/1312.6082.

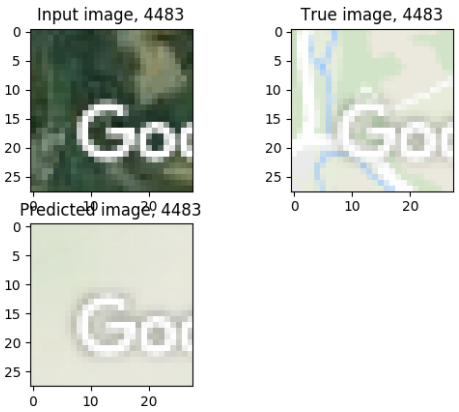
Mokhtarzade, M. and M. J Valadan Zoej (2007). "Road detection from high-resolution satellite images using artificial neural networks". In: *International Journal of Applied Earth Observation and Geoinformation* 9.1, pp. 32–40. ISSN: 15698432. DOI: 10.1016/j.jag.2006.05.001.

Teschioni, Andrea, Franco Oberti, and Carlo Regazzoni (1999). "a Neural-Network Approach for Moving Objects Recognition in Color Image Sequences for Surveillance Applications". In: *Neural Networks*.

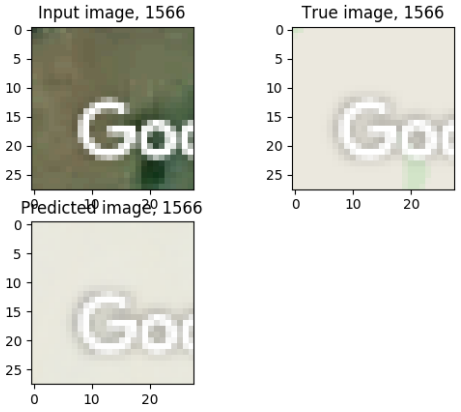
Appendix I



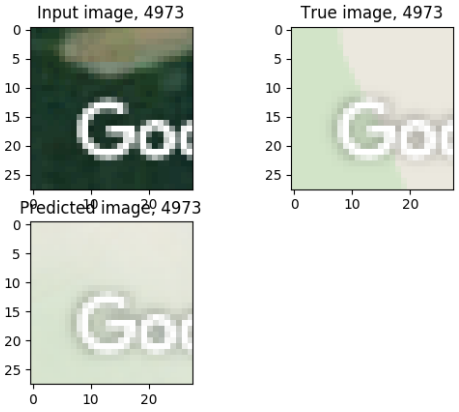
(a) First subfigure - 200^{th} epoch



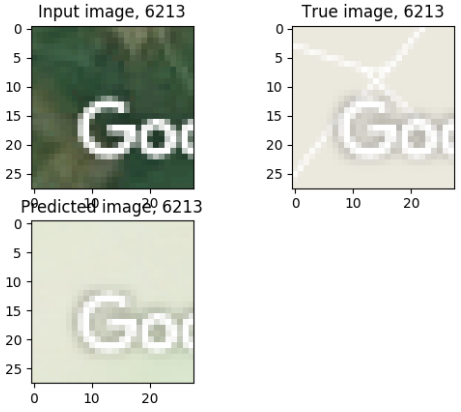
(b) Second subfigure - 250^{th} epoch



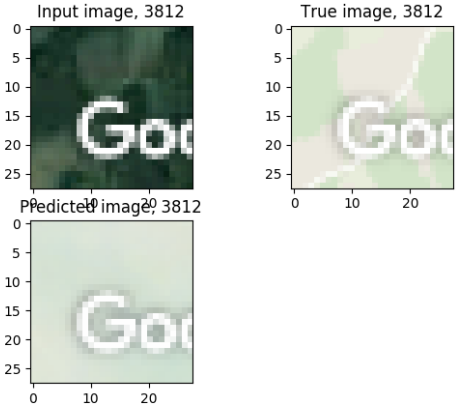
(c) Third subfigure - 300^{th} epoch



(d) Fourth subfigure - 350^{th} epoch



(e) Fifth subfigure - 400^{th} epoch

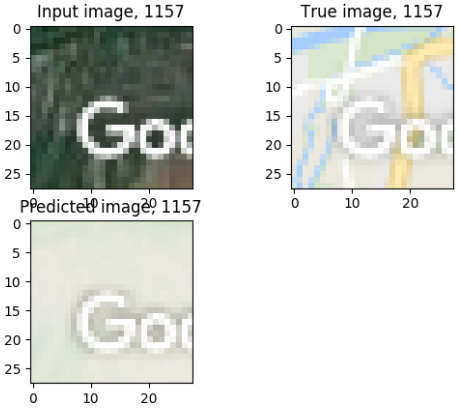


(f) Sixth subfigure - 450^{th} epoch

Figure 10: Predicted images with MLP in 12.000 28 $\times$ 28 images and 500 epochs.

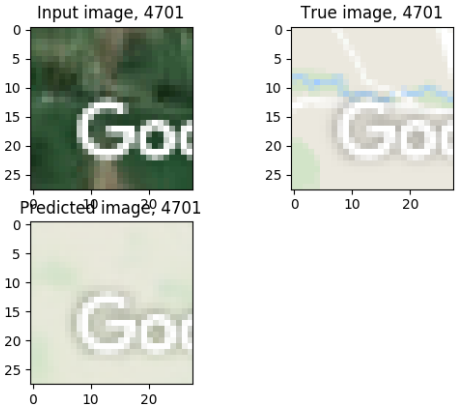
Appendix II

(a) First subfigure - 200^{th} epoch

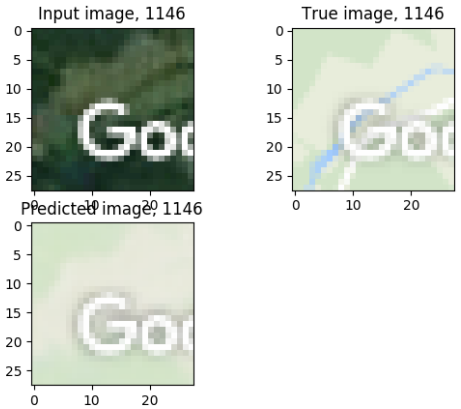


(b) Second subfigure - 250^{th} epoch

(c) Third subfigure - 300^{th} epoch



(d) Fourth subfigure - 350^{th} epoch



(e) Fifth subfigure - 400^{th} epoch

(f) Sixth subfigure - 450^{th} epoch

Figure 11: Predicted images with CNN in 12.000 28×28 images and 500 epochs.

Appendix III

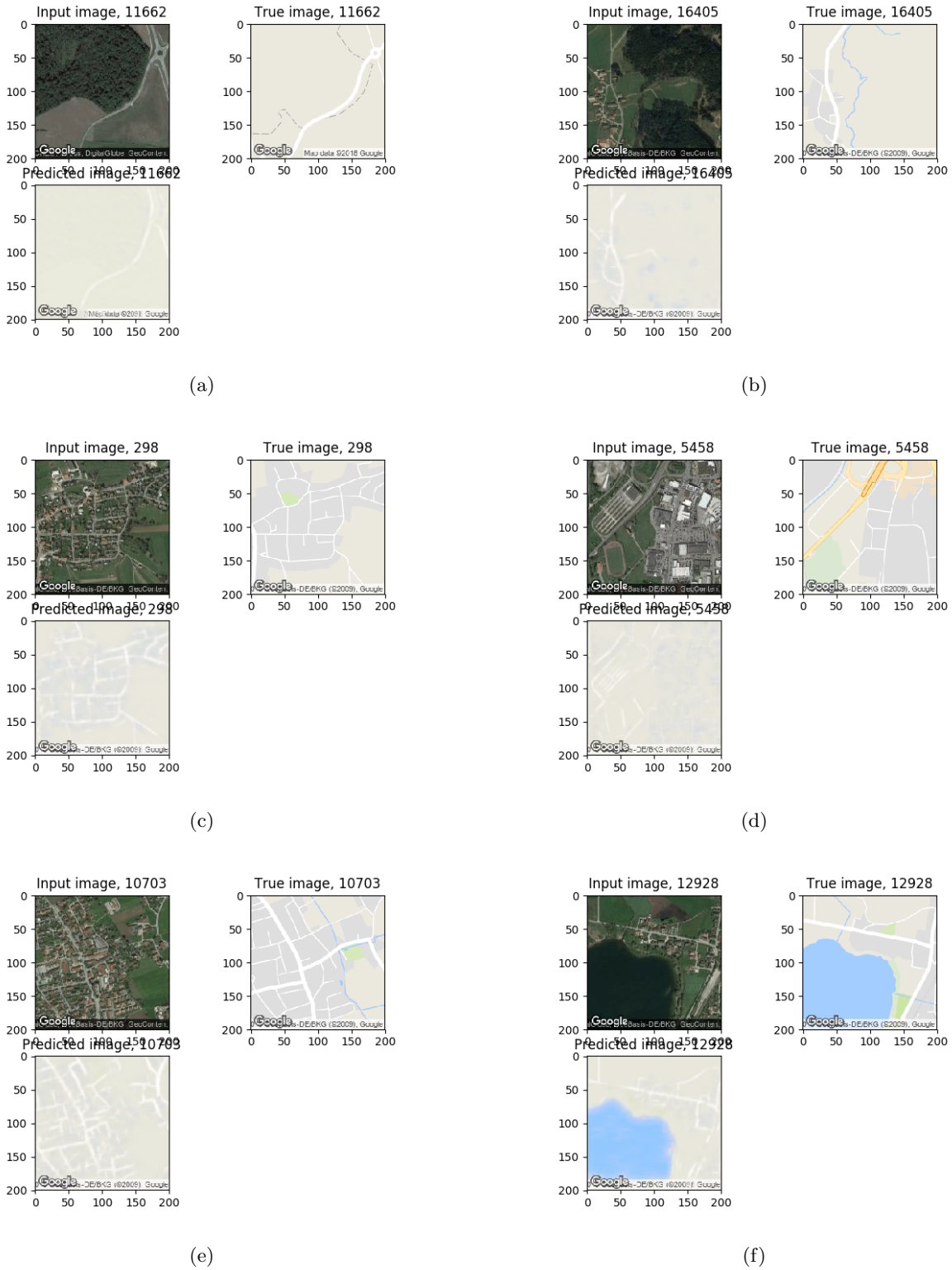


Figure 12: Predicted images with the CNN architecture in 36.000 200×200 images and 100 epochs.

Appendix IV

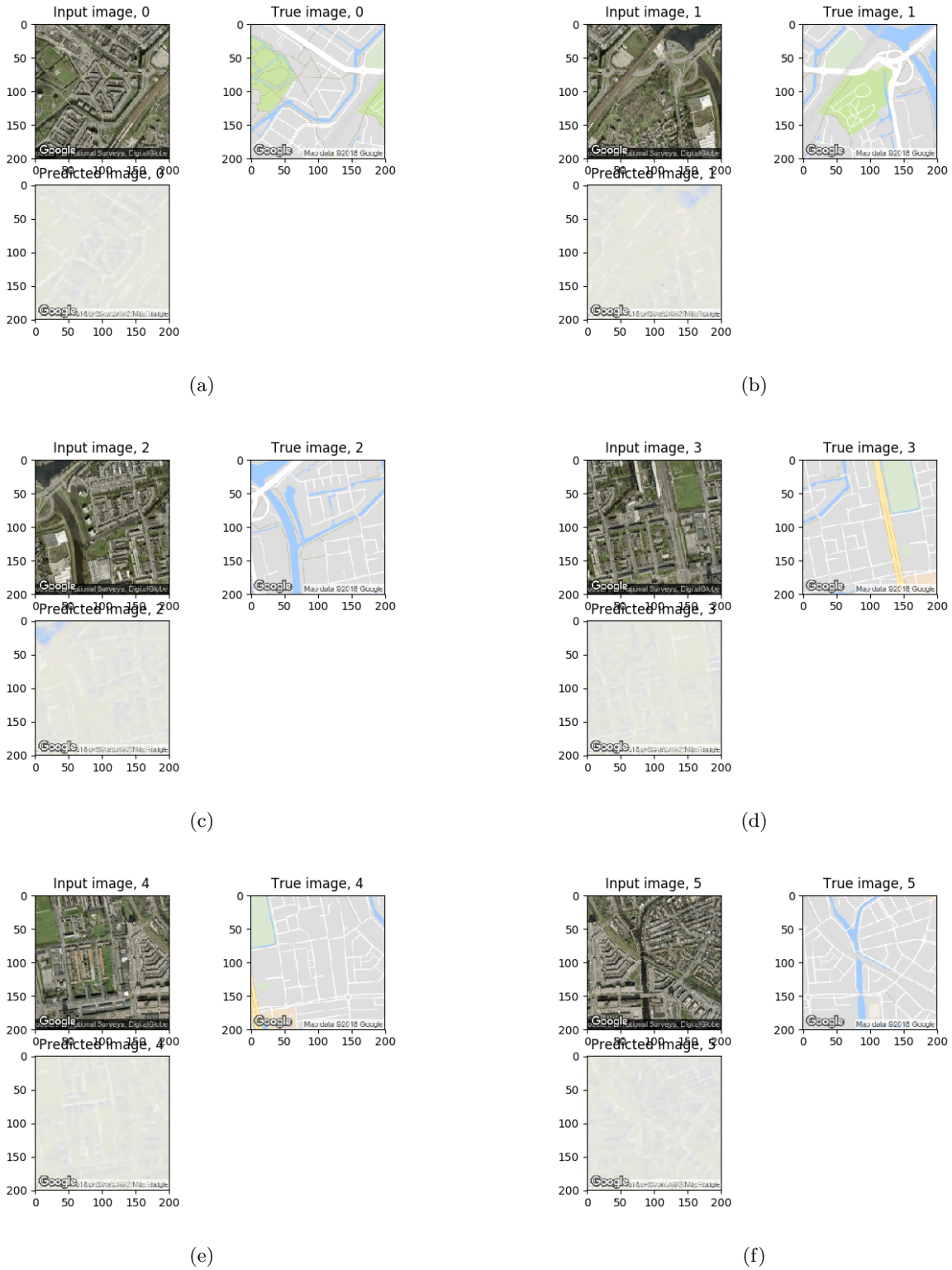
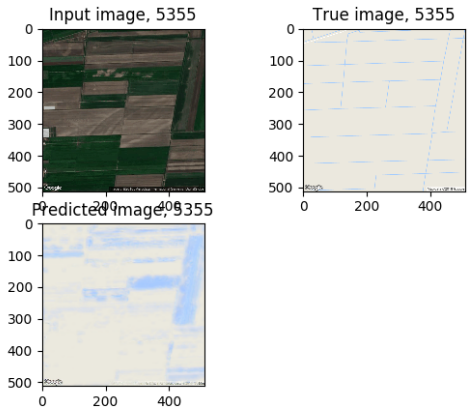
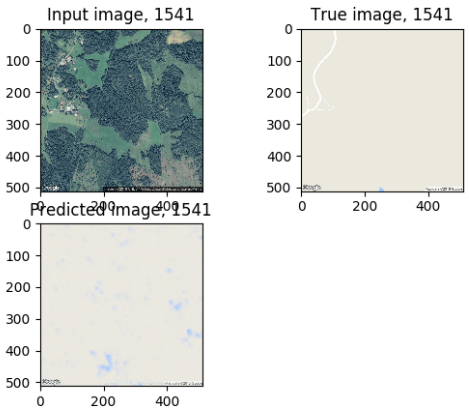


Figure 13: Validation in a new dataset composed of 100 200×200 new images from Leiden and 100 200×200 from Amsterdam using the CNN architecture as described in the text.

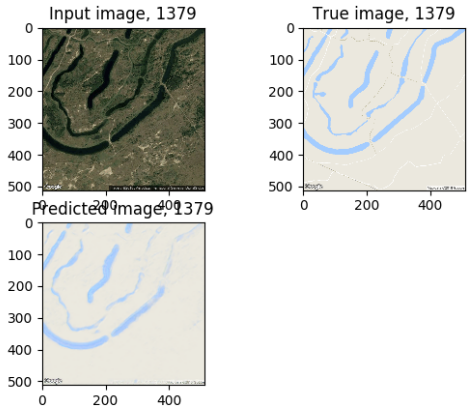
Appendix V



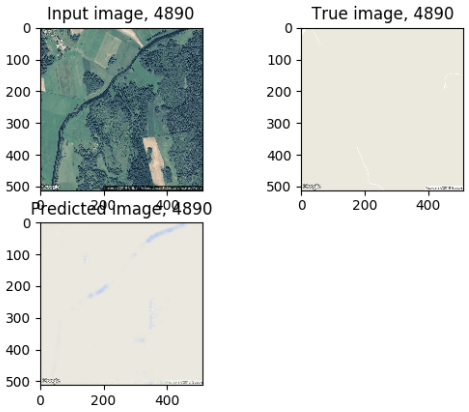
(a) Epoch 20



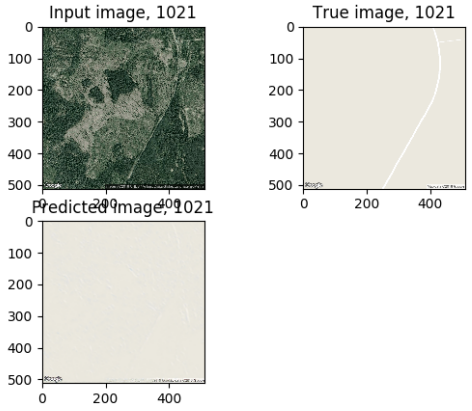
(b) Epoch 25



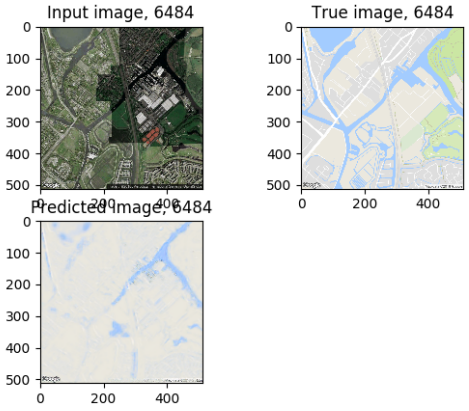
(c) Epoch 30



(d) Epoch 35



(e) Epoch 40



(f) Epoch 45

Figure 14: Predicted random training image every 5 epochs for the last 30 epochs that the CNN is trained on 512×512 images.