Road Segmentation

Minimal Effort
Manuel Vidigueira, Sabrina Kall, Emily Jamieson

***Abstract*—Given a dataset of satellite images of various areas and a set of corresponding groundtruth images highlighting the roads, training was carried out to allow detection of roads in general satellite images. To create such a tool, training data was used to create a baseline logistic regression classifier, whose parameters were then optimized. A variety of different classifiers were tested and optimized, culminating in a neural network taking color images augmented with 45 rotation as input, with an F1 score of 0.897.**

I. Introduction

The advantages of being able to identify roads on satellite images are obvious - any task from creating maps to plotting trip routes would become trivial. A set of 100 images of size 400x400 and their hand-annotated solutions - where the "road" pixels are marked as with the binary value "1" and the other pixels with "0" - allowed us to do supervised training of a classifier which can then identify roads on new, unknown satellite images. The quality of the classifiers were evaluated using the F1 Score ($F1 = \frac{2 * precision * recall}{precision + recall}$) tested on a series of 608x608 images.

II. Methods and Results

The data training set used was relatively straightforward, consisting of 100 regular png-format image files, which are basically arrays of pixels made of RGB values. We could manipulate these using Python's PIL library [1]. As features to train the classifiers, we generally used a combination of the mean and variance of pixels' RGB and greyscale values over a patch of adjacent pixels. Indeed, we were more interested in the relationship between adjacent pixels than their individual values, because we were trying to identify roads, represented as specific strips of the image, with continuously identical or similar pixel values. We computed these patches as overlapping squares with a given length and height of pixels and it was on these patches, rather than individual pixels, that we trained the classifiers. As described below, this given square side length is a hyperparameter which we can optimize.

In order to be able to locally test a classifier without needing to resort to the CrowdAI platform every time, we computed the F1-score (as well as the accuracy, precision and recall) using functions from the scikit-learn library [2], once using a simple train-test split and once doing k-fold cross-validation, which also gave us the variations of the measures. (Note that local validation was limited by the amount of time some of the more complex models required for training, which could make cross-validation prohibitively expensive.)

As a baseline, we used scikit-learn's logistic regression trainer and a patch size of 16. This resulted in an F1-score of 0.458 on the crowdAI platform.

We then improved our classifier using feature selection and expansion, and different types of classification algorithms, each time varying relevant parameters to optimize the returned results.

A. Feature Expansion and Parameter Optimization

Initially, the regularization strength of the logistic regression model was varied to try and obtain the best value and avoid any potential overfitting of this model. A large range of values were tested but they did not have an effect on the classifier's predictions, suggesting that with the low number of features we were dealing with, additional regularization was not necessary.

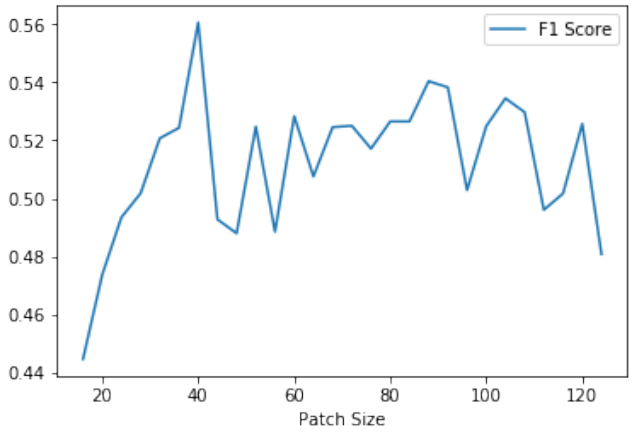


Figure 1: Estimated F1 score of a Logistic Regression classifier given patch sizes [16, 128]

One of the next parameters we addressed was the patch size. As can be seen in Figure 1, the best patch size for the training data was quite obviously 40, and a submission to crowdAI with this value rather than 16 gave us an F1 Score of 0.48. Note that while the graph in Figure 1 promises a mean F1 score of 0.56, computing the variance of this score with 200-fold cross-validation gave us a result of +/- 0.24, suggesting a large amount of overfitting. We also saw that while the training images have a side length of 400, a multiple of 40, this was not the case for the test images,

which were 608x608, meaning that the patch-to-pixel ratio for the test images was not optimal. Therefore, we found the divisor of 608 closest to 40, that is, 38, and indeed, using this patch size resulted in another F1 Score increase on the crowdAI platform (0.484), even though the cross-validation F1 Score found locally fell.

Another avenue we explored was whether using both the greyscale and color means and variation, instead of just one or the other (so far we had used only greyscale) would improve our classifier. This test proved successful as, combined with the patch size of 38, this vector of 8 features yielded an F1 Score of 0.514.

Various image enhancement methods such as contrast enhancement and histogram equalization were tested, adding an enhanced image in addition to the 8 features already in use. Adaptive histogram equalization was one enhancement method which showed the highest local F1 score, so the clip-rate used in this classifier was then optimized resulting in an F1 score of 0.516.

Finally, a number of different classifiers were compared to the earlier logistic regression model. K nearest neighbours was one model which demonstrated a high F1 score locally. The number of neighbours was then optimized, resulting in a model with 25 neighbours and an F1 score of 0.548.

Additional image enhancement methods or classifiers could have been experimented with, however our focus turned to neural network models which offered greater accuracy than those used up to this point.

	Submission	Training F1	Test F1
1	Baseline (Logistic Regression)	0.44 ± 0.1	0.458
2	Vary Regularisation	-	-
3a	Vary Patch Size (40)	0.56 ± 0.17	-
3b	Vary Patch Size (38)	0.54 ± 0.17	0.484
4a	Colour + Greyscale	0.50 ± 0.09	-
4b	Colour + Patch Size 38	0.58 ± 0.15	0.514
5a	Image Enhancement - Enhance Contrast	0.53 ± 0.19	-
5b	Image Enhancement - Adaptive Histogram Equalisation	0.58 ± 0.14	0.516
6	K Nearest Neighbours (25)	0.61 ± 0.19	0.548

Figure 2: Summary of Classifiers and F1 Scores. 60x cross-validation was used for all of the local Training F1 scores.

B. Neural Networks

Following the success of Convolutional Neural Networks (CNN) like VGG [3], GoogLeNet [4] and ResNet [5], we decided to apply the same techniques in the hopes of improving our current models.

We started by running the neural network implementation provided in the project files. The model itself is a simple convolutional neural network made up of two blocks of 2D Convolution, ReLU and MaxPool layers, followed by a final fully connected layer (ReLU). It uses tensorflow's "softmax cross entropy with logits" loss function. More details can be found in the code. After training it for 100 epochs, without any type of data augmentation, (using 80 of our images for training and 20 for validation) we achieved an accuracy of 74.9% and an F1 score of 0.812 on our validation set. The test F1 score (CrowdAI) was 0.648. This was a vast improvement over our previous models and showed promise that a Neural Network based approach was the best direction to pursue. We decided to keep this as our new baseline as we experimented with other, more advanced, CNN models.

When researching, we came across another image segmentation task: segmentation of neuronal membranes in electron microscopy images. Even though the context is significantly different, we noticed that the characteristics of the problem and the data were very similar. Both tasks involved 2-class segmentation: road/not road and membrane/not membrane. Furthermore, aside from the difference in linearity between cell membranes and roads (roads tend to be straight), we can make a very faithful analogy between roads and membranes, and between cell interiors and city blocks. Figure 3 compares the two problems' datasets.

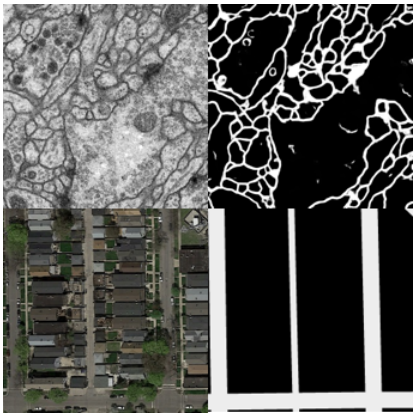


Figure 3: Comparison between an image for neuronal membrane segmentation (top) and an image for road segmentation (bottom), and their respective groundtruths. Membrane images taken from [6], with groundtruth inverted.

Research on the methods used to solve this analogous problem yielded several promising model candidates. Among them were variations of Fully Convolutional Networks [7], Deep Neural Networks (DNN) [8] and U-Nets [6]. In the end, we decided to experiment with the U-Net architecture due to a few key factors:

- It was designed to work with low amounts of training data, which suits our task.
- It claims to be faster to train than previous, state-of-the-art, approaches.
- It has achieved very good results in competitions (ISBI cell tracking challenge 2015).

- It is very simple to understand and implement, and there are several open source implementations available online.

We implemented our U-Net using Keras [9], with a tensorflow [10] backend. As a starting point, we used an open source implementation of the U-Net of the original paper [11]. Dropout [12] is applied before the fourth max pool layer, and before the first up sampling, to prevent overfitting. Figure 4 summarizes the networks architecture.

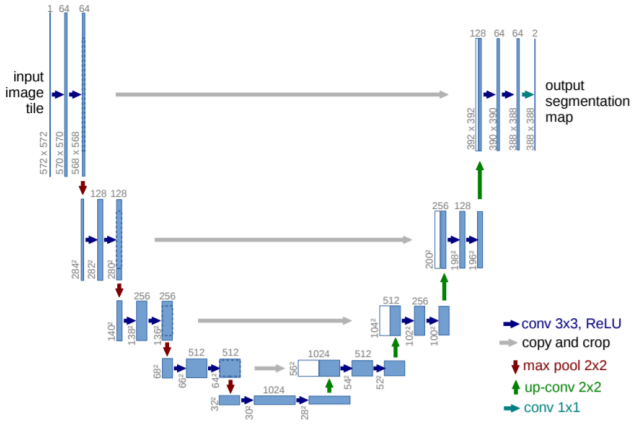


Figure 4: Each blue box represents a multi-channel feature map, with the number of channels on top, and the x-y size on lower left edge. White boxes represent copied feature maps. Taken from [6]

With an average training time of 5 hours per epoch, totalling to around 25 hours for 5 epochs, we decided to postpone local validation and use all the data in the training set, as the model would prove too costly and time-consuming to run both for every modification we planned to try. To decrease the risk of overfitting the test set, we applied dropout, extensive data augmentation, and defined our general methodology before doing crowdAI submissions, using these only as final results.

We first ran this neural network on our 100 training images, converted to greyscale, without any data augmentation. We used the Keras' built-in measure system to obtain accuracy values on the training set for each epoch. It had reached 99% by the third epoch.

However, computing the F1-score on the test set using the CrowdAI platform showed a slightly different story.

With an F1 score of 0.799 at epoch 3, it was a vast improvement over our baseline. However, the sinking scores of epoch 4 and epoch 5 indicate potential overfitting on the training data, likely caused by the small amount of images provided to train on. As neural networks require large amounts of input for training, we turned to data augmentation to fill the demand for more data.

1) Data Augmentation: As we were given only a small number of images, data augmentation is a must if we plan

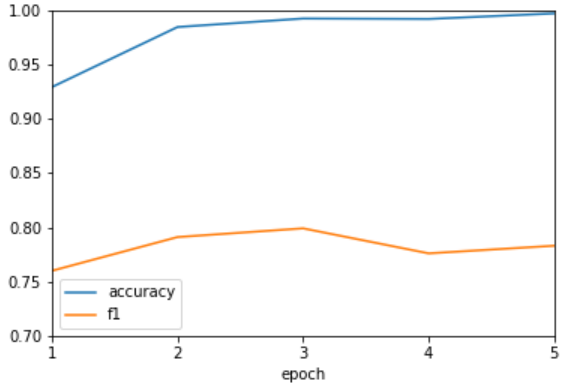


Figure 5: F1 Score and accuracy per epoch with classifier trained on unaugmented greyscale images

to use a DNN. By applying small scale rotations, shifts, zooms (rescales), flips and shears, the hope is to significantly increase the number of training samples, while maintaining a degree of realism in the images, so as to not degrade the generalization performance (after all, the test set is composed of real images)

We use the ImageDataGenerator ([9]) from Keras to facilitate the data augmentation process.

Each image is transformed together with its corresponding mask (so that each pixel remains correctly labeled).

Care has been taken when selecting valid transformations:

- Since we are using satellite images, vertical and horizontal flips still remain fairly realistic.
- Different training images were also taken at different altitudes (can be seen from the average size of vehicles), so zoom in is also possible.
- Zoom out should be limited as the sides are padded, which might degrade segmentation performance on the image's borders.
- Rotations are very useful since our training set has a distinct lack of diagonal roads. We limit rotations to 45 degrees [-45, 45]. Also, considering our data, it's natural to think that our model should be fairly rotation invariant.

The first time we used data augmentation for our training we applied no vertical flips, no rotations, and the fill mode was set to nearest. Running it on greyscale images, the model attained an accuracy of 98% on the training set and an F1-Score of 0.841 on the test set (5th epoch). Note that the accuracy, measured during the computation of each epoch, first converged rather quickly to 89% during the first epoch and 96% during the second epoch, but yielded diminishing returns afterwards, only gaining 1% during the third and fourth epochs, and 0.3% during the fifth. We noted 5 epochs as a good trade-off between computation time and generalization score.

Computing the F1-Scores on the test images for each epoch gave us a similar distribution.

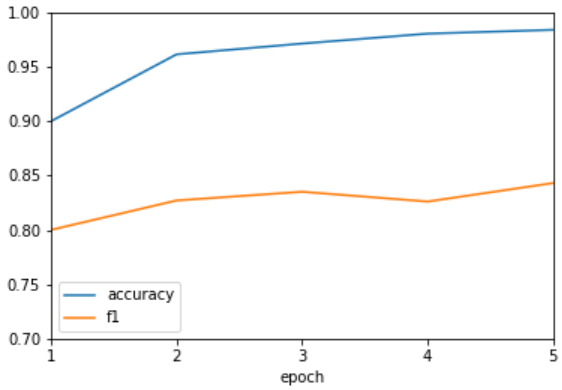


Figure 6: F1 Score and accuracy per epoch with training using augmented greyscale training images

Upon examination of the predictions of training images, we noticed that while horizontal and vertical roads were displayed flawlessly, the model occasionally classified brown patches of dirt as roads due to the lack of color in the greyscale images (Figure 7). We therefore reran the neural network with colored images as training input, which gave us an improved F1 score of 0.86 after 5 epochs.

Figure 7: Test Image #12

Original Image Grey training Color training

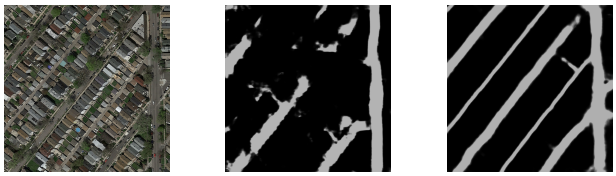
We also noticed that the classifier had problems segmenting diagonal roads (Figure 8). We therefore augmented the training images by rotating them by 45 degrees (and so creating more ”diagonal” roads to learn) and re-trained the network. We also added vertical flips, and changed the fill mode to reflect, in order to produce realistic padded sections when doing large rotations. This gave us another marked improvement, reaching an F1 Score of 0.888 by the fifth training epoch.

We then proceeded to combine colors and rotation in the training images, which culminated in an F1 Score of 0.897, our best and final result. We tried running a sixth epoch, but it ended up yielding a lower score (0.887), suggesting overfitting.

The data augmentation for our final model used the following parameters:

- rotation range: 45

Figure 8: Test Image #51



Original Image No rotation Rotation

- width shift range: 0.05
- height shift range: 0.05
- zoom range: 0.05
- horizontal flip: True
- fill mode: ’reflect’
- vertical flip: True

The submission for the crowdAI system had to be done using patches of 16x16 pixels, rather than output mask predictions. A ”foreground threshold” was applied to the proportion of background or foreground pixels to determine whether the overall patch was road or background. All the results presented were done with a threshold of 0.25. We experimented with other values around this one, but the improvements were insignificant (a threshold 0.23 proved to be the best for our final model, with an F1 score increase of 0.002). It’s important to be careful when assessing the meaning of these results, as it could easily be considered as overfitting the test set. For the sake of reproducing the best score, we keep the threshold at 0.23 for our run.py file.

	Submission	Best F1 Score
1	Given CNN	0.648
2	Unaugmented Greyscale U-net	0.799
3a	Augmented Greyscale U-net	0.841
3b	Augmented Color U-net	0.860
4a	Augmented Greyscale 45° rotation U-net	0.888
4b	Augmented Color 45° rotation U-net	0.897

Figure 9: Summary of Neural Networks and F1 Scores

III. Conclusion and Discussion

The optimal method developed, described in Section II-B, enabled high quality road identification with an F1 score of **0.899**. The model performed very well on the majority of straight stretches of road, however it still had some trouble identifying areas that closely resembled roads such as parking lots and train tracks.

To further develop this model in the future, we would like to experiment with other loss functions that optimize the F1 score during the training process. In addition, additional training datasets could provide more diverse data and enable more accurate predictions for unknown images. Further image preprocessing and enhancement (such as the Adaptive Histogram Equalization or Contrast Stretching as used in Section II-A) could be implemented and optimized to facilitate the training process.

REFERENCES

[1] A. Clark and Contributors, "Pillow (pil fork) library," https://pillow.readthedocs.io/en/5.3.x/index.html.

[2] Open-source, "Scikit-learn library," https://scikit-learn.org/stable/.

[3] K. Simonyan and A. Zisserman., "Very deep convolutional networks for large-scale image recognition." *ICLR*, 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf

[4] C. Szegedy, "Going deeper with convolutions." *CVPR*, 2015. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf

[5] S. R. K. He, X. Zhang and J. Sun., "Deep residual learning for image recognition." *CVPR*, 2016. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

[6] P. F. O. Ronneberger and T. Brox., "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, p. 234241, 2015. [Online]. Available: http://people.idsia.ch/~juergen/nips2012.pdf

[7] E. S. J. Long and T. Darrell., "Fully convolutional networks for semantic segmentation," *CVPR*, 2015. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

[8] L. M. G. D. C. Ciresan, A. Giusti and J. Schmidhuber., "Deep neural networks segment neuronal membranes in electron microscopy images." *NIPS*, p. 28522860, 2012. [Online]. Available: http://people.idsia.ch/~juergen/nips2012.pdf

[9] Keras 2.2.4. https://keras.io/.

[10] Tensorflow 1.12.0. https://www.tensorflow.org/.

[11] Z. Xuhao. https://github.com/zhixuhao/unet.

[12] A. K. I. S. N. Srivastava, G. E. Hinton and R. Salakhutdinov., "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, p. 19291958, 2014. [Online]. Available: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf