Road Segmentation with Satellite Images

Gjems, M. S., Johansen, A., and Vold, M.
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—In this report, we will introduce our implementation of a convolutional U-Net to classify roads from satellite images. The U-net uses convolution layers to contract the images a smaller size and transposed convolution layers to expand the images to the original size. The model is augmented with techniques like batch-normalization, dilated kernels and dropout to improve accuracy. 100 training images were provided, so a data augmentation process was used to increase the amount of images to 2,100, where the original 100 images were rotated, flipped, scaled, etc. to improve the training of the model. The best model achieved a F1 score of 0.863 on the CrowdAI test set.**

I. Introduction

In the latest years, deep convolutional networks have outperformed the state of the art in many visual recognition tasks. While convolutional networks have already existed for a long time, their success was limited due to the size of the available training data and computational power.

The typical use of convolutional networks is on classification tasks, where the output of an image is a single class label. However, in our image processing task, the desired output should include localization, meaning that a class label is supposed to be assigned to each pixel.

Provided with a set of satellite/aerial images acquired from Google Maps, and corresponding ground-truth images where each pixel is labeled as {road, background}, our goal is to train a classifier to segment roads in these images, i.e. assign a label {road=1, background=0} to each pixel. The provided data set contained 100 images of size 400x400, with ground-truth images of the same size. The test set contained 50 images of size 608x608.

Even though our model predicts a value for each pixel in the picture, the final evaluation method will evaluate 16x16 patches of pixels and assign each patch either a value of 0 for background, or a 1 for road. With use of a threshold, the values are assigned based on the mean of the pixels in the 16x16 patch.

II. Baseline Model

A simple baseline model was given along with the project description. This model consists of a convolutional neural network (CNN) with two convolutional+pooling layers, and a soft-max loss function. This model got a F1 score of 0.653.

III. Model Architecture

All of our implemented models are U-Nets, inspired by the work of Ronneberger et al. (2015) [1]; convolutional neural networks that consist of a contracting path and an expansive path, which gives them the U-shape. The architecture is described in Figure 1. The contracting path is a typical convolutional network with repeated application of convolutions, each followed by a Rectified Linear Unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive path combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path. Our implementation is generalized for colour images, images with three colour channels (RGB), and for input images of size 400x400. The architecture that was explored was a network consisting of four contractions layers and four expanding layers, with a bottleneck (the part of the network which is in between the contracting and expanding parts) layer. Each layer consists of two convolution layers were each is followed by a ReLU activation function. This model will in total consist of five layers, like the architecture in Figure 1, and approximately 2,000,000 trainable parameters. For each layer this U-Net architecture will reduce the size of the images by half in the contracting part, and increase it with a factor of two for the expanding part. This results in images of shape 25x25x256 for the bottom layer.

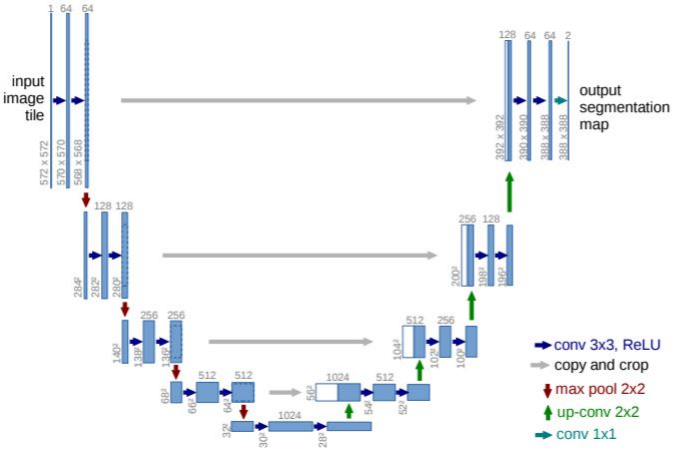

Fig. 1: The U-Net Architecture

The output of the model is tensors with a shape of 400x400x1. This means that there is only one channel for representing colours, resulting in gray-scaled images. The model uses the Adam optimizer, binary cross-entropy as its loss function, and accuracy as the metric for evaluation. Three different callbacks are also used:

1. Model Checkpoint is used to keep the best model from all the different epochs the model goes through. This ensures that we keep the best weights for the model, and not the one from the last epoch, which are not necessarily the best ones.

2. Early stopping is used with a patience of 10, and it acts as a form of regularization to avoid overfitting the model. It monitors the validation loss for each epoch, and if 10 epochs are detected in a row without an improvement in validation loss, it will end the training early. Model Checkpoint also monitors validation loss.

3. The last callback reduces learning rate on plateau (ReduceLROnPlateau). This reduces the learning rate once the learning stagnates for three epochs, which models often benefits from when reaching a plateau.

Different optional layers were added to the model, with an option to use the layer or not, to give the architecture the ability to use different layers with different parameters. Such layers are Batch Normalization layers, found after every convolutional layer, Dropout layers, found after every max-pooling layer in the contraction part of the model and after every concatenation layer in the expanding part.

One more option for our convolution layers was the ability to use a dilated kernel [2]. The reason for this was that we noticed a trend for some of the predictions; it was, for example, sometimes difficult to predict smaller roads, as well as roads obscured by cars or trees. The dilation of the kernel allows us to get local, pixel-level accuracy, such as precise detection of edges, and to integrate knowledge of the wider, global context. The goal with this is to look at a pixel that an undialated kernel would not be able to take into consideration, hopefully getting a better prediction as more of the context around the kernel can be used.

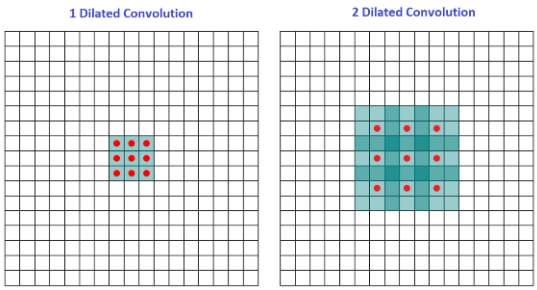

Fig. 2: Left image: Dilation rate = 1. Right image: Dilation rate = 2

One final note before going over to data augmentation. When predicting the test images of size 608x608, these images needs to be scaled down to a size equal to the training images for the model to be able to do predictions on them. Since the model also returns predictions of the same shape as the input images, the resulting predictions needs to be scaled up to 608x608 before they are ready for the final threshold evaluation.

IV. Data Augmentation

When training deep networks one can archive much better results by simply having a larger set of annotated training samples. In order to use the available annotated samples more efficiently, our approach relies heavily on the use of data augmentation to expand the training set.

We were originally given 100 satellite images with corresponding ground-truths. As data is valuable, and the more data we have, the better we can train, we generated new images from the ones we were given. With horizontal and vertical flips, rotations, zoom in and out, we were able to expand the data set with 2,000 new images, also with corresponding ground-truths. In addition, we had the option of adding salt-and-pepper noise to the images. This significant increase of training images is essential for training the network.

The augmented pictures were created with a few different methods. The possibilities are horizontal and vertical flip, shifting the image in height or width, rotation and zoom. The shifting range is set to 0.2, which means that images can be shifted either direction for up to 20% of the height and width, rotation range is set to 20, so images can be rotated by as much as 20 degrees. Images could also be zoomed in on by 25% and out by 10%. All these different augmentations could happen at the same time, so the images generated may not be very similar to the original images. Since the images do not contain any information on the content outside the boundaries of the image, the pixels that are close to the boundaries will be reflected over to the parts of the picture that doesn't really exist. By using the reflection, we can still keep some information about the pictures outside the bounds. This is illustrated by Figure 3. On the left side of the image one can clearly see that the roundabout is reflected over to the left, creating two roundabouts very close to each other. This is not ideal, but it is better for the model to train on full sized images rather than using border pixels for everything outside the edges of the image. One can also observe that the image also have been rotated, as the reflection can be seen in the top of the image too.

Another image augmentation technique that was tested was adding artificial noise to the training images. The theory here is by adding noise we controlled and knew the underlying ground-truth image the model will become more robust and be better at predicting images that contain natural noise, e.g. cars, people, vegetation, etc.

V. Hardware and Software

Keras, version 2.1.6 [3], was used as the high-level neural network API running on TensorFlow, version 1.12.0 [4], backend to develop our model. The models were trained on a single computer with an Intel Core i7-8550U CPU and a NVIDIA GeForce GTX 1050 GPU with 4 GB of dedicated RAM. This made the longest run time for any model around 10h with 100 epochs. Due to this RAM constraint, the batch size for both training and prediction is set to 4 so as to not encounter a memory error due to the GPU not having enough RAM to store the arrays of images.

Fig. 3: Example of an augmented image

VI. Models

For different models we tried using different configurations of parameters. These parameters are: use augmented images or not, if used how many, salt and pepper noise, batch normalizing layer, dropout and dilated kernel.

Changeable parameters are how many augmented images to include, the dropout rate, and whether to use batch normalization and/or salt and pepper noise.

For validation we split the training data in a training set and a validation set, with 80% for training, and 20% for validation. This leaves us with 1,680 images for training and 420 for validation at each epoch. The results in Table I show the F1 score that each model achieved when submitting the predictions on CrowdAI. One aspect not shown in this table is that every model except model number 7 (the one with the best score), used a threshold of 0.25 when creating the submission. This model, however, used a threshold of 0.3. This is because this threshold seemed to yield the most accurate results, but due to the submission method we were unable to test this extensively.

The models were validated locally by using the accuracy of the model, calculated by number of correct predictions divided by all predictions. This gives a pretty good estimate of how good the model is, however it does not take the difference between the two classes into account. As there is a far greater number of pixels classified as background, than there are pixels for roads, the classes are quite unbalanced, possibly causing some bias. This implies that F1 score, the metric used for the test set on CrowdAI, is a more appropriate choice for evaluation. It takes both precision and recall into account, by computing the harmonic mean of the precision and recall. It is also worth noticing that the evaluation for the validation set, locally, does this pixel-wise, while the evaluation for the submission is done for each 16x16 patch in the prediction. As F1 score is the harmonic mean, it will penalize wrongly predicted patches harder than the accuracy metric will.

A summary of all the models are given in table I.

Model	Imgs	S&P	BN	DO	DK	Run time	Epochs	Result
1	100	✗	✗	-	-	2 min	32	0.703
2	100	✓	✗	-	-	3-4 min	41	0.706
3	100	✗	✓	0.1	-	7 min	100	0.739
4	100	✗	✓	0.1	2	7 min	66	0.764
5	100	✓	✓	0.1	2	10 min	44	0.707
6	2100	✓	✓	0.1	2	10 h	100	0.857
7	2100	✗	✓	0.1	2	10 h	70	0.863
8	2100	✗	✓	0.2	2	10 h	97	0.859

Table I: Imgs: number of training images, SP: salt and pepper, BN: batch normalization, DO: dropout, DK: dialated kernel

Fig. 4: Row 1: Test image 30 and 12. Row 2: Predictions for image 30 and 13. Row 3: Image 30 and 12 with the corresponding prediction overlay.

VII. Results

An iterative process was used to attempt to improve our model. We started off with the most basic model, and only the 100 original images that were provided. The results with using only 100 images were better than expected and beat the baseline model. From there, different permutations of parameter settings were tested with models trained on the 100 images. We observed that salt-and-pepper noise didn't do much for our model when only 100 images were used. What

(a) Image 22 (b) Image 25

(c) Image 43 (d) Image 48

Fig. 5: Different test images with the corresponding prediction as an overlay.

did give a noticeable increase in the F1 score, is the model with batch normalization, a dropout of 0.1, and a dilated kernel with a dilation rate of 2.

We then decided that we should use these parameters in the next iterations of the process. Next, for every original image 20 new images are created. This, together with the original images, gave us 2,100 images to train on, which in turn increased the F1 score to 0.863. Other parameter settings like a dropout probability of 0.2, and salt-and-pepper noise, were not as successful.

In Figure 4 you can see an example of two images from the test set with their predictions, and the predictions as an overlay in red color over the original image. Here we can see what the model is able to predict given an image. For picture 30, the left one, it predicts the roads in an excellent way. The predictions are almost an exact match to the roads and they are well defined with no parts missing. For picture 12, the right one, this is not the case. We can see that there are large parts of the roads that hasn't been predicted as roads, part of "parking lots" which is predicted as roads, and generally very unclear lines.

VIII. Discussion

By looking at the predicted images in Figure 5, we observed that the model had troubles recognizing very narrow roads, roads covered by trees, and could sometimes mistake straight lines like railways as roads. Poorly defined roads, surrounded by a lot of parking space for example, also posed a challenge. Over all, streets were very well captured. This is visible on every image with overlay shown. The middle of image a) shows that part of the railway was mistaken for part of the road. Depending on the use of the predictions this can be a good thing, e.g. in creation of a map that will be used for navigating the roads, and you want to know if the road goes under a bridge or a small tunnel. However, in this case it's not the goal, and therefore a false positive. In the top of the image we can see the opposite. Where the railroads split up into three is not predicted as a road, most likely because the distance between the roads got to large, even for the dilated kernel. Furthermore, one can notice that the road in the bottom of the image, which has a darker colour, is not predicted as a road. This might be that the sudden change in colour from old asphalt to new, threw the model off and made it predict the wrong values. In image b) we can see a small parking space being predicted as road. Parking spaces comes in all kinds of shapes and sizes, so getting enough of them to train on from 100 images is unlikely. It can be discussed if this parking space should be counted as a road or not. Image c) shows the partial prediction of the left side of the railroad, the railroad itself is not predicted as road. Finally, for d) we can see the effect of narrow roads that are angled approximately 45 degrees off the x-axis. This isn't the only image with predictions like this for that type of roads, and it seems like this was the biggest area of wrongly predicted roads. This might be a result of only rotating images 20 degrees during the data augmentation phase resulting in a model that preforms poorly on roads that have this orientation.

IX. Summary & Future Work

Even though the results we got using this model were satisfactory, there are still room for improvements. The first thing that could have been done is rotating the augmented images more so that we could have gotten a higher representation of roads that we saw the model had a hard time predicting, 45 degree roads. As we are training on images that are rotated and moved around, it would have been possible to rotate the test images by different degrees so as to get a better prediction representation for each image. After predicting using the same image in different angels, rotate them back, and then take the average of all images pixel-wise. It is also possible to train a second model that uses image recognition on our grey-scale predictions to increase the confidence of our final prediction.

References

[1] Philipp Fischer Olaf Ronneberger and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.

[2] Ferenc Huszár. Dilated convolutions and kronecker factored convolutions. 2016.

[3] François Chollet et al. Keras. https://keras.io, 2015.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.