Road Detection Based on U-Net Architecture

Team Name: WW

Junkai Wang
Baylor College of Medicine
junkaiw@bcm.edu

Wanli Wang
Baylor College of Medicine
wanliw@bcm.edu

Abstract

Automatic road extraction from remote-sensing imagery plays an important role in many applications. However, accurate and efficient extraction from very high-resolution images remains difficult because of, for example, increased data size and superfluous details, the spatial and spectral diversity for road targets, disturbances (eg. trees and buildings), the necessity of riding weak road edges while avoiding noise, and the fast-acquisition requirement of road information [1]. Here, we use U-Net architecture, a strong network and training strategy that relies on the strong use of data augmentation to use the annotated samples more efficiently.
In this report, we attempt to solve the task of detecting roads from ten thousand satellite images by implementing U-Net. Finally, our team achieved a score of 0.697 in the final prediction and got a final standing at 27.

1. Data

For our task, we have a training set of more than ten thousand satellite images and their related boolean masks. For the testing step, our well-trained model will make predictions on approximately two thousand satellite. All the images and masks used in the project come from DeepGlobe (http://deepglobe.org).
Here, each image covers roughly 16 acres of land in a 256 m by 256 m cube. Each image is of size 512 by 512 resulting in a pixel resolution of 50 cm. For the boolean masks, the road pixels are labeled as one, and non-road pixels are labeled as zero.

2. Data preprocessing

2.1 Resize dataset

Originally, each image in our dataset is of size 512 by 512. With the limitation of the GPU we have, training on the original size of the images will be time-consuming and result in the low batch size in the training process. Therefore at the beginning, we resized all the images and boolean masks into size 128 by 128 before the training. By resizing the images

and masks, we achieved a much better computational efficiency and were able to increase the batch size to 32.

2.2 Data augmentation

Even though we have enough images and corresponding masks for our training purpose, we believe that our network will show a better performance with data augmentation. One purpose of data augmentation is to enlarge the dataset so that the network will be better trained with increased data size. Another reason for the use of data augmentation is to provide the training network with multiple different angles for the same image. With image rotation, flipping and other augmentation methods, the model will be more likely to recognize different forms of roads. Therefore, we used the "imgaug" package (https://github.com/aleju/imgaug), which is a package used for image augmentation for machine learning experiments.

It was the early stage when we decided to implement the data augmentation. As mentioned before, we were using the resized images and masks for the training purpose which turned out to be wrong decision as we realized later. But at that time, in order to achieve better performance of the prediction step, we decided to implement the data augmentation method to enlarge the training dataset. We tried multiple augmentation strategies including horizontal flip, vertical flip and Gaussian Blur. As a result, the size of our training dataset increased to four-fold compared to the original dataset. In return, it took much longer to train the model and the predicted score increased to 0.5. Obviously, this was still far away from a satisfied result.

Then we did some literature review on the road segmentation tasks and figure out that the Gaussian blur actually does harm to our training step. Since our provided training and test images are all of the same quality and high resolution, the blurring transformation will interrupt the balance of resolution among the training dataset. Therefore, we decided to implement horizontal flip, vertical flip and ninety-degree rotation to our dataset as augmentation strategies. But with the limitation of time, we did not implement the data augmentation in our later training models, but we will include this in our future direction. Figure 1 shows one example of original image/mask and additional processed images/masks from data augmentation methods.

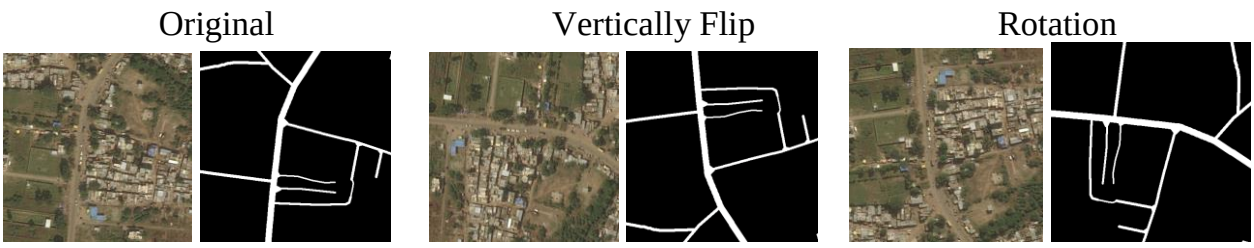


Figure 1. The representative data augmentation example for one satellite image and corresponding mask from training set.

3. Model

3.1 Model selection

Road detection and classification from satellite imagery has been a popular research area the past decade. We consider road detection as a semantic segmentation task. The goal of semantic segmentation is to label each pixel of an image with a corresponding class of what is being represented. Here in our case, the label for each pixel are only two classes: road and non-road.

From the literature review, we knew that a group from University of Freiburg modified and extended fully-convolutional network (FCN) architecture so that it works with very few training images and yields more precise segmentations. This more elegant network is called U-Net [2].

The principle of U-Net architecture is to supplement a usual contracting network by successive layers, where pooling operations are replaced by upsampling operators. Hence these layers increase the resolution of the output. What's more, a successive convolutional layer can then learn to assemble a precise output based on this information. U-Nets have originally developed for biomedical image segmentation tasks, but they are also used for many different applications. We decided to use U-Net to solve our road detection task here.

3.2 U-Net architecture

The U-Net architecture is illustrated in Figure 2. It consists of a contracting path which captures context (left side) and a symmetric expanding path which enables precise localization (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (here we used same padding), each followed by a rectified linear unit (ReLu) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling, U-Net doubles the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution (up-convolution) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLu. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes.

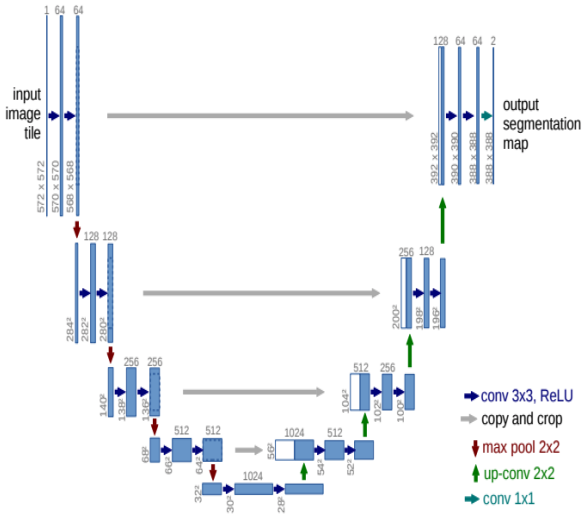


Figure 2: Architecture of U-Net.

3.3 Loss Function

In our training procedure, we use "binary cross-entropy loss" as loss function.
The cross-entropy loss is defined as equation (1).

$$CE = -\sum_{i}^{C} t_i log(s_i) \qquad (1)$$

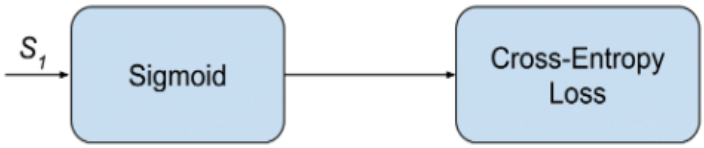
Where t_i and s_i are the ground truth and the CNN score for each class i in C, respectively. Usually an activation function (sigmoid/softmax) is applied to the scores before the CE loss computation, we write f(Si) to refer to the activations.

In a binary classification problem, where C'=2, the cross-entropy loss can be defined also as equation (2).

$$CE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1) \qquad (2)$$

Where it's assumed that there are two classes: C1 and C2. t1[0,1] and s1 are the ground truth and the score for C1, and t2=1-t1, and s2=1-s1 are the ground truth and the scores for C2.

A binary cross-entropy loss, also called sigmoid cross-entropy loss. It is a sigmoid activation plus a cross-entropy loss. Unlike softmax loss, it is independent for each vector component (class), meaning that the loss computed for each CNN output vector component is not affected by other component values. That's why it is used for multi-cable classification, were the insight of an element belonging to a certain class should not influence the decision for another class. The pipeline turns out like Figure 3.



$$CE = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

$$f(s_i) = \frac{1}{1 + e^{s_i}}$$

Figure 3: Pipeline for binary cross-entropy loss.

3.4 Optimization

3.4.1 Learning rate

At the beginning we chose our learning rate as 10^{-4} and we trained for 10 epochs and got loss curve, which decreases exponentially so we just take this as our default learning rate. The curve is shown in Figure 4.

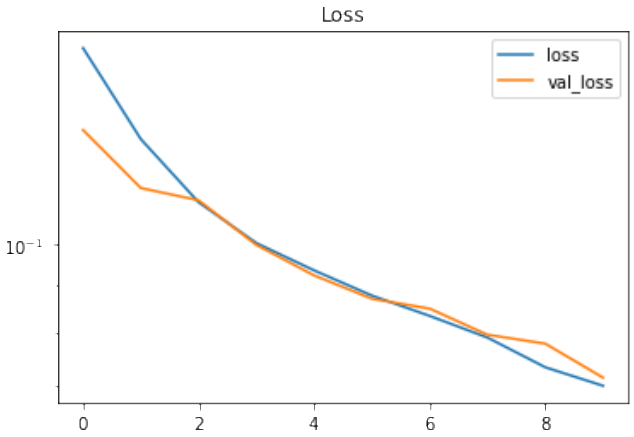


Figure 4: Loss curve for 10 epochs for the decision of the learning rate.

3.4.2 Adam Optimization algorithm

Here, we use Adam optimization algorithm for our deep learning project. Adam optimization algorithm is an extension to stochastic garden descent that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

Adam combines the advantages of two other extensions of stochastic gradient descent (AdaGrad and RMSProp). Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. The initial value of the moving averages, beta1 and beta2 values close to 1.0 result in a bias of moment estimates towards zero. This bias is overcome by first calculation the biased estimates before then calculating bias-corrected estimates.

3.4.3 Batch size

For the batch size, with the limitation of the GPU, we did not get chances to do the comparison among different batch sizes and chose the batch size achieved the best performance. Like when we do the training based on image of size of 128 by 128, we can set the batch size to 32, but when we do the training with the original image size which is

512 by 512, the maximum batch size we can set is just 4 , which is apparently not a widely acceptable one. So we have to sacrifice the depth of our model to set the batch size at 8. And we will talk about the ‘shallow’ and ‘deep’ U-net in the later part of the report.

3.5 K-fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. It is a popular method because it is simple to understand and because it generally results in a less biased than other methods, such as simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly
2. Split the dataset into k groups
3. For each unique group:
    a. Take the groups as a val set
    b. Take the remaining as train set
    c. Fit a model on train set and validate it by val set.
    d. Apply the model to the next training set, and obtain a final model based on all the training processes. Or, retain the evaluation score for every model after one training process and then discard the model, average the scores.

The pipeline for K-fold cross validation is shown in Figure 5.

Figure 5: Pipeline of K-fold cross-validation

We did an eight-fold cross validation to our whole dataset, and did 10 epochs for every iteration. We will have more detailed description for the training process in the final model part.

4. Approaches to implement the model

4.1 Resize the dataset to 128 by 128

At the beginning with the consideration of the limitation of GPU, we decided to resize all the images in the dataset to 128 by 128, which makes it possible for us to set the batch size as 32. We split the training paths into 85 and 15 percentage. 85 percentage of the training dataset is taken as the training set, and the other 15 percentage is taken as the validation set.

Next we used the tradition U-Net to fit the training set, and changed the epochs from 5 to 10 to 20. And for each epoch, we made a prediction for our test dataset, based on the model we got after training and obtained a prediction score for every model.

We found that, first the score increases with the epochs (when we set them within small epochs, like smaller than 20). But the problem is that, since the model is based on images of size 128 by 128, so for our test dataset, the predicted masks for them are still of size 128 by 128. We have to rescale them to 512 by 512 to assess the result on leaderboard.

4.1.1 Mask post-processing

First, we basically used the resize function to change the mask from 128 by 128 to 512 by 512, and of course, we lost many messages in the upsampling. So we searched for some mask processing method, we used the ‘erosion’ and ‘dilation’ functions in the ‘OpenCV’, still, there was not too much improvement for the quality of masks.

4.1.2 Data augmentation

Second, we tried to use the data augmentation to compensate this loss and improved the final results.

As I have mentioned in 2.2, we use the data augmentation of horizontal flip, vertical flip and ninety-degree rotation to extend our dataset to four-time larger than before. Finally, we took the extended dataset and trained it for 20 epochs, also did Erosion and Dilation to our masks resized from the predicted masks. Our best results could just achieve 0.5, which was not a satisfactory result.

4.1.3 Extend the model

Since we did not want to lose the computational efficiency brought by training with images of size 128 by 128. We tried to extend the U-Net model and make sure the output of our model is of shape 512 by 512. Our extended model is shown in Figure6.

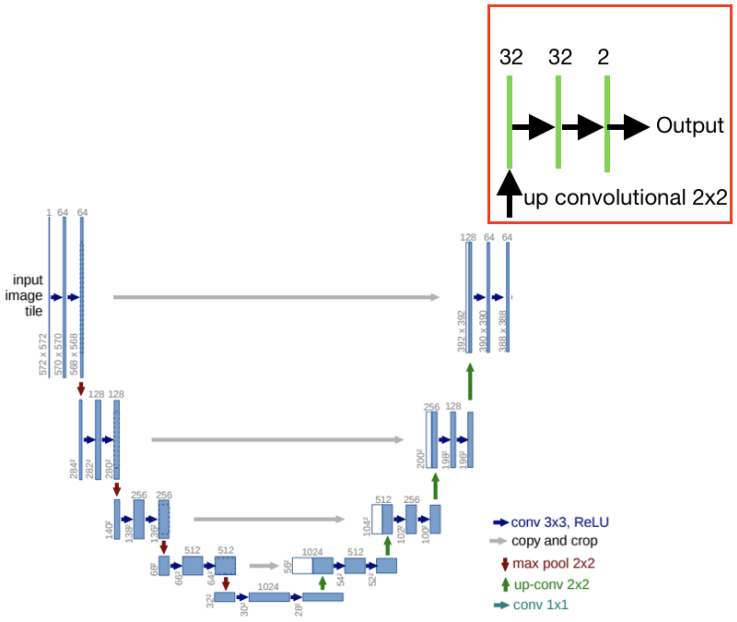


Figure 6: Extended U-net model.

For the extension, we did the up-convolution with filter 4x4 when we came to the last two steps of the original U-Net protocol, and then like the other layers, do the convolutions and ReLu, finally reshape the output. So our input, the processed images in the model are still of size 128 by 128, but the output came to size of 512 by 512.

Based on this extended U-Net, we use the augmented dataset, and trained it for 20 epochs, but there was no improvement for the result, still at around 0.5. Think back on our extension, it is quite like the resize function, upsampling the layer but without the correlated concatenation layer in the contracting path, we also lost many information in the upsampling, thus with no improvement after the extension of the model.

4.2 Training on Images of 512 by 512

No further improvement based on the training of 128x128 images, we started to discard resize and just trained on the original dataset of size 512 by 512. Since the memory of the dataset became larger, we just did the later training based on the original dataset, without the augmentation, for time consuming. Also training with the original dataset had to sacrifice the batch size, for the limitation of GPU, we can just set the batch size at 4 to promise the training. Next, we used the traditional U-Net, set the batch size at 4 and take the original dataset as input. We trained this for 10 epochs, and got a result of 0.65, which is a great improvement than the earlier models.

4.2.1 Simplify the model

Since we were still concerned with the batch size, a batch size of 4 is a little small for training the deep learning neural network. So we thought maybe we can increase the batch size a little bit with the sacrifice of the model complexity.

So for every filters in the convolutional transformation, we reduced the filters by twice. For example, in the original U-Net, the first convolutional step is done with 64 filters (the feature channels), we reduced it to 32, the same reduction for all the other filters. After simplifying the model, we can increase our batch size to 8, more acceptable than 4. As previous one, we used this 'shallow' U-Net, set the batch size to 8, take the original dataset as the input, trained it for 10 epochs. For this, we got a result of 0.63.

Compared to the original U-Net with the batch size of 4, there is a little reduction of the predicted result (from 0.65 to 0.63). But the simplified model saved triple time than the original one. And we call this simplified model as 'shallow' U-net and call the traditional U-net as 'deep' U-Net in the later part.

4.3 Final Training Models

We did the final training based on three models. The summary for the three models shown in Table1.

Table 1

	model 1	model 2	model 3
U-Net	Shallow	Shallow	Deep
First Channel number	32	32	64
Batch Size	8	8	4
Train/Val split method	Simple	K-Fold CV (k = 8)	Simple
Epoch	30	10 * 8	20
Score	0.6562	0.6223	0.6978

4.3.1 Training with 'shallow' U-Net for 30 epochs.

For model1, as we have talked about in the batch size part, since we want to increase our batch size from 4 to 8, we have to sacrifice the depth of our model. We halves all the feature channels in the original U-Net architecture. And we use this model to do 30-eopchs training with early stop. And finally we got a score of 0.65.

4.3.2 Training with K-fold cross-validation.

Consider that, all the previous models are based on a fixed split of 85 percent and 15 percent, so the resulted model is trained more likely to have a better prediction on the images that share more similarities with 85 percentage of images in the training part, but worse for the others. This is the bias added by the fixed split of the data.

To optimize the bias, we decided to have a try on eight-fold cross validation for the final training model. And we think there are two ways to carry out the eight-fold cross validation method. One is we can pass the weight gained after one iteration to the next iteration and get a final model to do the prediction after all the eight iterations. Or we can just separately training every iteration and take the average predicted scores got from every model. Because of the limitation of time, we just carried out the first way to do this eight-fold cross validation training.

In a summary, we do this with a 'shallow' U-Net, and set the batch size at 8. We set epochs to 10 for every training set, and after a model based on one training set finished all the 10 epochs, we save its weight, and applied it to the next training set. Finally, we can obtain this accumulated model, after eight-fold training. We finally got a predicated score of 0.62 for this model.

We thought this not satisfied enough result is due to the small epochs we have set for every iteration. So for every iteration, the model cannot be fully trained on the training set of this iteration, then it has to be uploaded to the next iteration, we can tell from the loss curve in Figure7 that the validation loss will increase a little bit sometimes increase a lot after the change of every iteration.

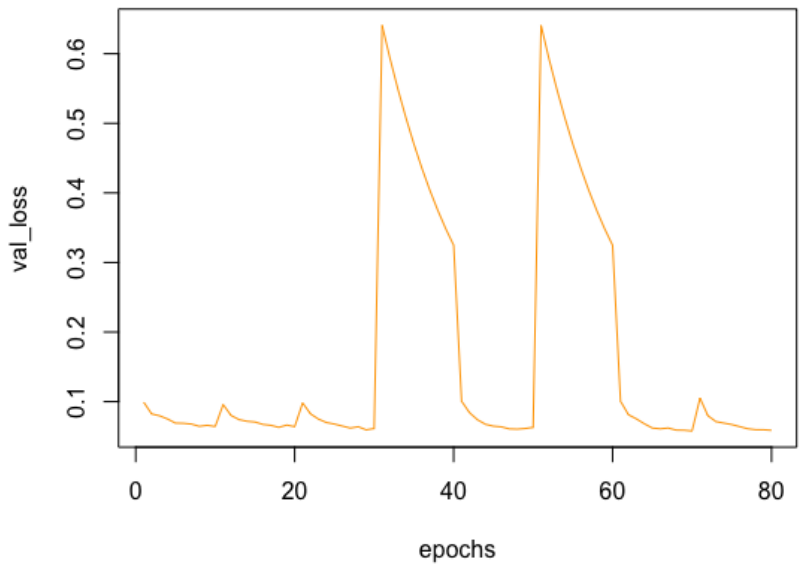


Figure 7: Validation loss curve for eight-fold cross validation training with 10 epochs each fold.

4.3.2 Training with ‘deep’ U-Net for 20 epochs

For the third model, we just take the traditional model the ‘deep’ U-Net, set batch size at 4 and do 20-epochs trainings with early stop. Finally we got a score of 0.697 based on this model, which is also the best score we have achieved. It turns out that the depth of the model counts a lot in the performance of prediction, even 10 epochs less, and batch size is very small, we still got the best score from this model. The loss and accuracy tendency of this model is shown in the Figure8.

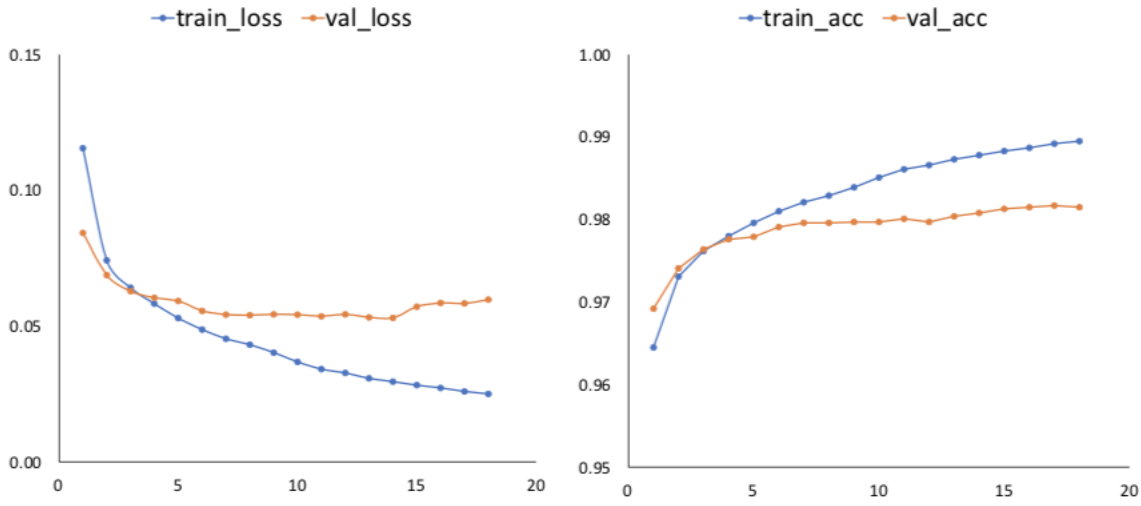


Figure 8: loss and accuracy curve for model3.

5. Summary

We have three final models for our project, and some of the predicted masks carried out by these three models are shown in Figure 9.

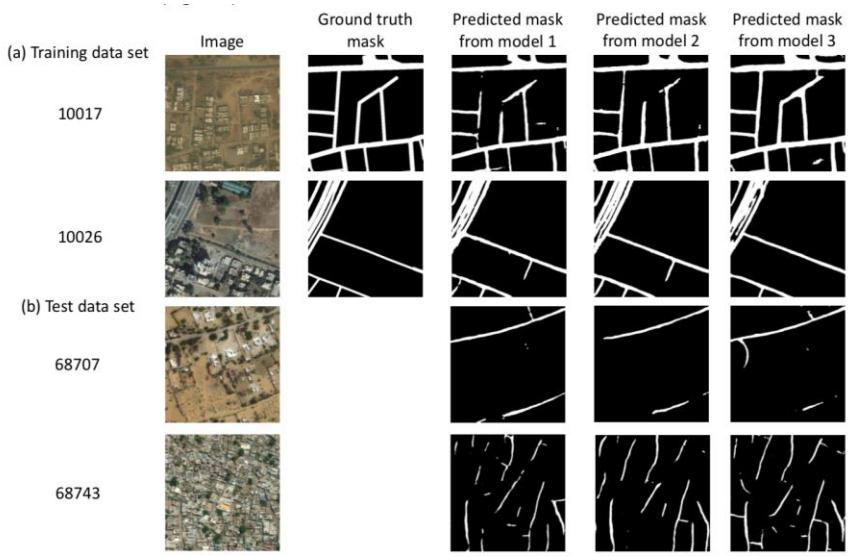


Figure 9: Comparisons among different settings of U-Net Architecture on training data and test data. Representative predictions are shown in training data (a) and test data (b)

We can tell from these predicted masks that, first the best score achieved by the model3 can precisely detect most of the roads in the satellite images, which shows that our model performs quite good. Second, in some of the situations, the roads detected by model1 or model2 cannot be detected by model3, so we can ensemble these models with different weights based on their scores. This may help us to get a better result.

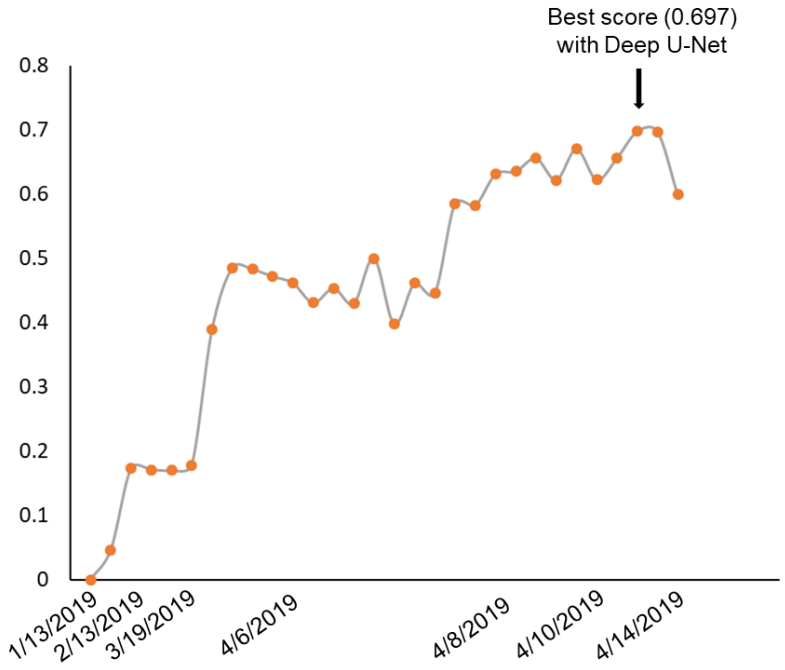


Figure 10: Score changes along with submission time. We have the highest score with the Deep U-Net.

6. Future Directions

We have learned a lot in this project. Since neither of us in the team comes from a computer science background, this the first time we come to this computational project. During this project, we learned that, first, how to find a suitable architecture based on the goal of the project by doing literature review and simple tests. Second, how to optimize the performance of model by adjusting different hyper-parameters. Third, how to deal with the challenges we might meet with during the training procedure, for example the limitation of the GPU, and the wrong directions or sometimes just the unlucky situations. Finally, what we learned most from this project is that, how we can get resources from different areas together to achieve a better performance for this project, like the statistics and the computer science.

For the next steps, hopefully with better server, we would definitely go with the ‘deep’ U-net, and do data augmentation with different degrees of roads in the dataset. Also, we will keep with K-fold cross validation, but set like 50 epochs with early stop for every iteration. Finally, we will also use ensembles to build up the confidence of our result.

7. Acknowledgement

We want to express our sincere gratitude to Professor Devika Subramanian and all TAs of course COMP 540. This project wouldn't be finished without their generous guidance and help. We also thank DeepGlobe (http://deepglobe.org) for making this dataset available for academic use.

Reference

[1] M. Maboudi, J. Amini, M. Hahn, and M. Saati, "Road network extraction from VHR satellite images using context aware object feature integration and tensor voting," *Remote Sens.*, vol. 8, no. 8, 2016.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9351, pp. 234–241, 2015.

- U-Net repository: https://github.com/zhixuhao/unet
- Satellite Images and Masks: http://deepglobe.org/
- Data augmentation package "imgaug" repository: https://github.com/aleju/imgaug