Project 2: Road Segmentation from Satellite Images Using Convolutional Neural Network

Jangwon Park, Jean-Baptiste Beau, Frédéric Myotte

I. INTRODUCTION

Image segmentation is an increasingly popular technique in many applications in computer vision. The proposed challenge here is a special case of this field and entails segmenting roads in satellite images obtained from Google Maps. For the purpose of this task, we are concerned with classifying patch images of 16x16 pixels from the original 400x400 pixel images as either 1 (road) or 0 (background).

Linear methods are significantly limited in learning complex relationships that are often needed in this type of challenge. Quick literature review on the problem of road segmentation reveals that convolutional neural networks (CNNs) are the state-of-the-art methods due to their ability to learn local features within images and map non-linear relationships.

However, CNNs are very complex and have an infinite number of variations. As such, the purpose of this document is two-fold: 1) understanding the effects of the hyper-parameters, and 2) constructing a model that performs reasonably well based on these findings.

II. EXPLORATORY DATA ANALYSIS

The satellite images obtained from Google Maps are particularly challenging due to the closeness of road and background pixels in terms of colors. Figures 1 and 2 present example images in which the parking lot, houses and buildings are very close to the concrete color of the roads themselves, which could potentially make classification more difficult. To distinguish them, the roads have been shaded in red. Furthermore, Figure 2 shows that the trees could possibly serve as obstacles in classification by covering the parts of the pixels that should really be classified as roads. In light of this finding, we used a technique called image padding to take the surrounding context of the 16x16 patch image into account. We describe this technique in the section below.

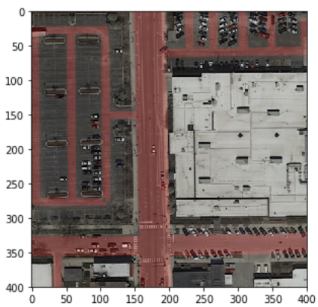

Fig. 1: Parking lot

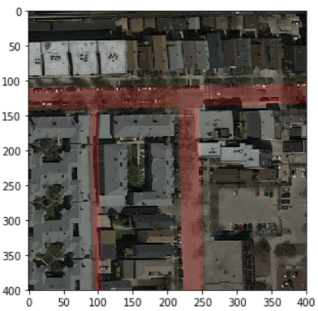

Fig. 2: Trees and buildings

III. DATA PROCESSING

Image segmentation is typically accompanied by a number of data processing steps. In our project, the only data processing we do is image padding, a technique that significantly improved the performance of our model.

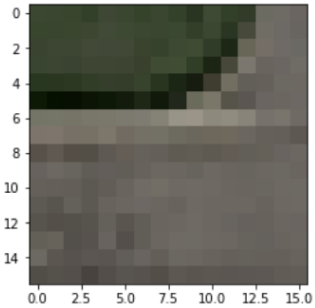

Fig. 3: Example 16x16 patch image. Hard to tell if it belongs to a road.

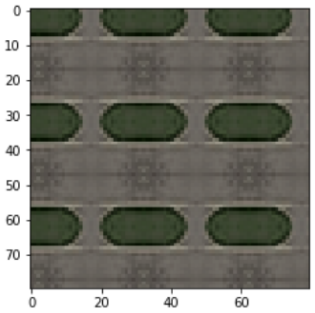

Fig. 4: Padded version of Figure 3. Easier to tell if it belongs to a road.

A. Image Padding

Image padding is a technique which consists of extending the 16x16 patch image to a larger image by means of mirror-reflecting each pixel, keeping the 1:1 aspect ratio. This way, the indiscernible patterns in 16x16 patch images are enhanced and easier to visualize. Figure 3 is an example of a 16x16 patch image whereas Figure 4 presents its padded version. Augmenting the image with padding comes with the price of increasing exponentially the computational costs. In this project, we use padding size of 16 to ensure that every pixel in the original 16x16 patch image is reflected at least once. Under certain network architectures, however, more padding may be required to make sure the spatial dimensions are not zero at any point in the network.

IV. OPTIMIZING HYPERPARAMETERS

Reaching an optimal network architecture requires considerable effort due to the large number of hyper-parameters involved in CNNs and the infinitely many variations of them. In this section, we summarize and explain important experimental findings that led us to the types of architectures used in our final model. The performance of a network is measured by the F-score as the classical definition of accuracy may sometimes be misleading in image segmentation tasks.

A. Convolutional Layers

Many architectural choices are inherent to convolutional layers, such as the size of the filters, the number of filters, and the number of convolutional layers. To learn the effects of these choices, we present three models described in the following naming convention:

1) Model 1: C4[3x3]-32-32-64-64-F1-128
2) Model 2: C4[3x3]-64-64-128-128-F2-128-256
3) Model 3: I1-64-32-F2-128-256

The first model is made of 4 convolutional layers with 32, 32, 64, and 64 filters of size 3x3 respectively. By default, the architectures are characterized by the conventional structure

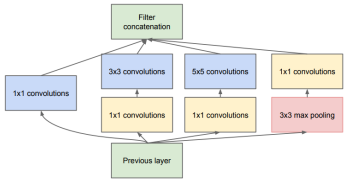

Fig. 5: Structure of GoogleNet's inception layer [1]

of alternating convolutional and max pooling layers. At the end, there is one fully connected layer which links to the final output layer of two neurons, representing the two labels of our data. Input shapes of the training data are 16x16 pixels for these experiments, and rectified linear (ReLU) activation functions are used at every layer by default. The second model is identical to the first in all aspects except that it has twice as many filters in each convolutional layer as well as an extra fully connected layer. Finally, the third model is characterized by an *inception layer* as described in GoogleNet and in Figure 5 [1]. By performing parallel convolutions of different filter sizes, it effectively bypasses having to decide on the optimal filter size. While keeping the number of parameters similar between models 2 and 3, we have compared the effect of these architectural choices.

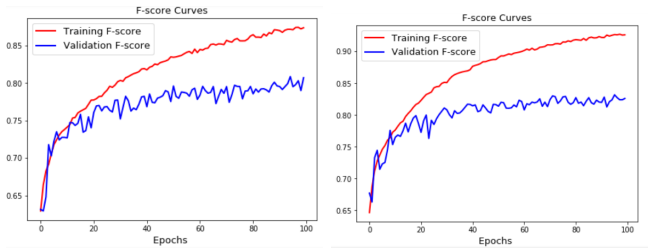

Fig. 6: Comparison of validation F-scores throughout training between model 1 (left) and 2 (right) as described in Section IV.

Comparing models 1 and 2 is de facto assessing the viability of larger, more complex model for the task of road segmentation. All else being equal, having greater number of filters in convolutional layers and an additional fully connected layer led to higher validation F-scores as evident in Figure 6. Furthermore, the F-scores achieved by model 2 nearly at every epoch along the way outperform model 1, demonstrating the effectiveness of a more complex architecture for road segmentation.

On the other hand, model 3, with the use of an inception layer and therefore a variety of differently sized filters simultaneously, converges very rapidly with a clear sign of a plateau in the training F-score with a smaller gap between the training and validation F-score. However, we quickly realize that there is numerical instability with model 3's architecture, which is best demonstrated in Figure 7 and by comparing their final 10-fold cross validation results in Table I. The validation F-score curves tend to overestimate the true F-scores as they are computed based on continuous probabilities as opposed to forced 0 or 1 labels. Therefore, Table I presents more accurate

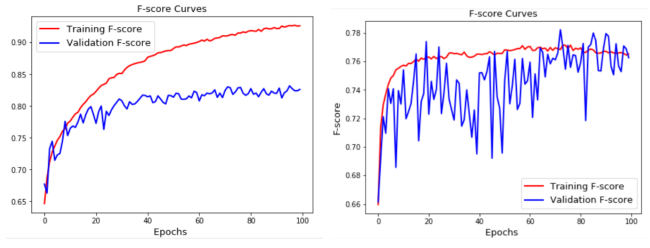

Fig. 7: Comparison of validation F-scores throughout training between model 2 (left) and 3 (right) as described in Section IV.

TABLE I: Validation F-scores of the three models in Section IV

Model	Cross Validation F-scores $\pm$ σ
C4[3x3]-32-32-64-64-F1-128	0.664 $\pm$ 0.023
C4[3x3]-64-64-128-128-F2-128-256	0.690 $\pm$ 0.021
I1-64-32-F2-128-256	0.599 $\pm$ 0.036

results. Model 3's architecture is in fact highly ineffective as the true cross validation scores are the lowest with the highest standard deviation even in comparison to a much simpler architecture such as model 1. As such, model 2 is considered the most promising.

These experiments with various architectural hyper-parameters lead us to a few insights which provide the grounds for our final choices. The conventional approach of alternating convolutional and max pooling layers appears to be promising compared to other structures such as the inception layer. Furthermore, there is promise in scaling the network to a larger size as there are no clear signs of overfitting given the same amount of data. The conventional structure would limit us to the maximum of four sets of alternating layers, provided max pooling layers reduce each spatial dimension by half each time. This in fact provides us with a reasonable scope of research; any less would restrict the capability of CNNs for road segmentation while any more is directly concerned with computational feasibility especially without a powerful GPU that can compute massive tasks in parallel (training can take several days on our laptops with regular CPUs).

B. Max Pooling Layer

Employing pool size of 2x2 pixels is by far the most dominant choice in nearly all applications of CNNs. As such, we chose a window size of 2x2.

C. Dropout Layer

Dropout is a very popular method for preventing overfitting in neural networks in general, which works by arbitrarily dropping or excluding a number of neurons with some probability. [2] compares the placement of dropout layers and the effect of probabilities in CNNs at various points along the network for image classification task on MNIST data. Here, p represents the probability of *keeping* a neuron. In summary, as presented in Figure 8, dropout layers placed immediately after max pooling layers were the most effective with optimal dropout probabilities being either 0.5 or 0.3 depending on the application. With some trial-and-errors, we discovered that dropout layers at every max pooling layer with dropout rates 0.25 and 0.5 at every fully connected layer were quite effective in our application, whose plots we omit for brevity.

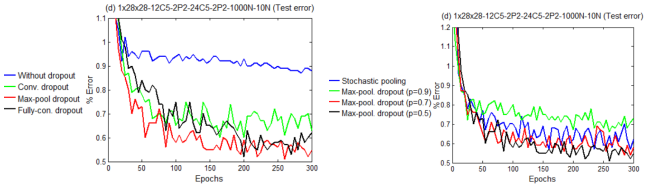

Fig. 8: Effects of the choices of the placements of dropout layers and their probabilities of keeping (a neuron) [2]

D. Fully Connected Layers

1) Number of layers and neurons

As the size of the neural network is limited by computational cost (and underlying training time), there is a trade-off between the number of convolutional layers and the number of dense layers. For a task of detecting local features in a set of images, convolutional layers are the most

important. Additionally, fully connected layers tend to increase the number of parameters a lot, as for each neuron in a layer there exists as many parameters as there are neurons in the previous layer. We resolve this dilemma by implementing at most one fully connected layer with no more than 128 neurons, for keeping training time reasonable while not compromising greatly our model's performance.

E. Activation functions

It is standard practice to use rectified linear units (ReLU) in neural networks in general. However, some applications have reported successes with Leaky ReLU (LReLU) activation functions to address the *dying ReLU* problem. It works by implementing a small negative slope (instead of 0) for $x < 0$ which we can control with the hyper-parameter α [3]. In this project, we use $\alpha = 0.1$ which has proven to be effective. In general, the use of LReLU leads to slightly faster convergence and higher average validation F-scores which is evident both in Figure 9 and Table II.

Fig. 9: Comparison of ReLU and LReLU on model 1 from Section IV

TABLE II: Comparison of ReLU and LReLU

Activation	Cross Validation F-scores
ReLU	0.664 ± 0.023
LReLU	0.677 ± 0.018

V. NETWORK ARCHITECTURE AND FINAL MODEL

Synthesizing the experimental results in the previous section, we present three architectures that are comparable in performance. They each have subtle variations to account for the fact that it is simply impossible to come up with a single optimal model with CNNs. Details of their architectures are presented in Table III. Network 3 is more experimental in nature, making use of a set of two consecutive convolutional layers and using dropout only near the end.

TABLE III: Complete list of layers in three comparable networks

Type	Network 1	Network 2	Network 3
Input	48x48x3	64x64x3	48x48x3
Convolution filters	64 (5x5)	16 (4x4)	128 (3x3)
Max Pooling	2x2 *same*	2x2 *valid*	-
Dropout	p = 0.25	p = 0.25	-
Convolution filters	128 (3x3)	32 (4x4)	128 (3x3)
Max Pooling	2x2 *same*	2x2 *valid*	2x2 *same*
Dropout	p = 0.25	p = 0.25	-
Convolution filters	256 (3x3)	64 (4x4)	128 (3x3)
Max Pooling	2x2 *same*	2x2 *valid*	2x2 *same*
Dropout	p = 0.25	p = 0.25	-
Convolution filters	256 (3x3)	128 (4x4)	-
Max Pooling	2x2 *same*	2x2 *valid*	-
Dropout	p = 0.25	p = 0.25	-
Fully Connected	128 neurons	-	64 neurons
Dropout	p = 0.5	-	p = 0.5
Output	2 neurons	1 neuron	1 neuron
Activation	Softmax	Sigmoid	Sigmoid

A. Ensemble Model

An *ensemble model* of several CNNs is formed by consolidating individual predictions from each of the networks by means of majority voting rule. In other words, given three models such as in our case, if two or more of the models predict 1 for a patch image, then the final consolidated prediction for that patch will be 1, but 0 otherwise. In neural networks, ensemble models are particularly desirable for the compelling reason that selection of weights is an optimization problem with many local minima [4]. By combining the three networks presented in Table III in this way, we were able to create a superior model which also serves as our final model.

VI. MODEL TRAINING

A. Loss Function

We use the cross entropy loss function, which is widely used in neural networks. For networks 2 and 3 as described in Section V, where the output layer has one neuron, we use the binary cross entropy function. For network 1, where the output layer has two neurons, we use categorical cross entropy function for numerical stability.

B. Type of Optimizer

Optimal optimizer can vary for different applications. On a relatively light network, we experimented with different choices of optimizers and their effect on validation F-score, whose results are shown in Figure 10. Although we observe that either RMSprop or Adadelta are the fastest and the most effective optimizers, they did not scale well with the size of the network as evident in Figure 11, demonstrating their lack of robustness in our application. For instance, RMSprop on a more complex network was numerically unstable while Adadelta on the more complex network showed signs of overfitting. Though overfitting can be avoided by introducing more data and/or stricter regularization, this is expected to be much more computational costly since it will require longer training time. Stochastic gradient descent (SGD) is very slow, which can also be very costly with complex networks. Therefore we used Adam, another popular optimizer which has great results.

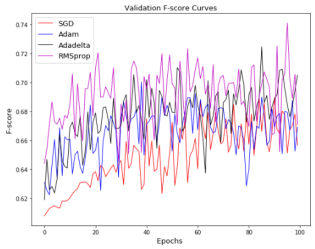

Fig. 10: Comparison of validation F-scores by optimizer. RMSprop and Adadelta are seemingly best, followed by Adam and then SGD.

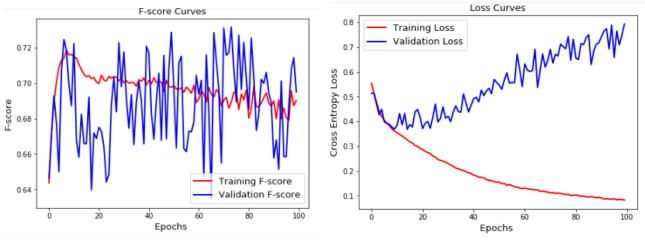

Fig. 11: Lack of robustness in RMSprop (left) and Adadelta (right)

VII. RESULTS AND DISCUSSION

Figure 12 summarizes the 10-fold cross validation F-scores of the models as described in Table III. By default, the individual CNN models are trained with image padding. To assess the effectiveness of image padding, we also present the result of the ensemble model created from CNN models trained with no padding. Finally, to assess the effectiveness of CNNs in general, we present our benchmark model which randomly generates 1 with a probability equal to the proportion of road image patches in the validation dataset.

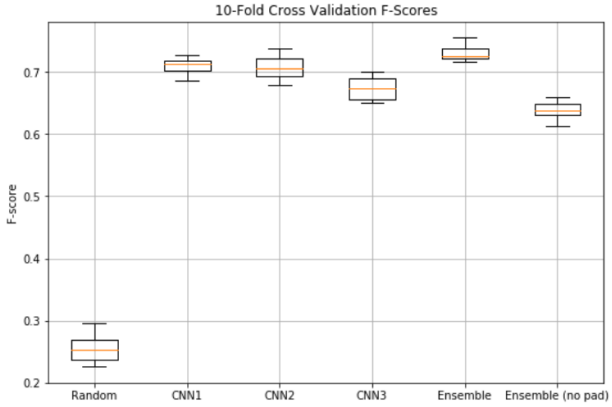

Fig. 12: Cross validation F-scores of the random model, models in Table III, and the final ensemble models

TABLE IV: Cross validation F-scores of the random model, models in Table III, and the ensemble models

Model	Cross Validation F-score $\pm \sigma$
Random	0.255 $\pm$ 0.022
CNN1	0.711 $\pm$ 0.012
CNN2	0.707 $\pm$ 0.019
CNN3	0.673 $\pm$ 0.018
Ensemble	**0.730 $\pm$ 0.011**
Ensemble (no padding)	0.640 $\pm$ 0.014

CNNs are very powerful tools that outperform the random model significantly. The use of image padding is clearly beneficial as the ensemble model developed from models trained without image padding performs significantly worse than any of the individual models trained with image padding. Furthermore, the ensemble model outperforms any of its individual constituents in terms of both F-score and standard deviation, as evident in Table IV. This suggests that consolidating several networks leads to a superior model in both performance and robustness.

Finally, we visualize our predictions based on our final ensemble model for example test images in Figure 13. We occasionally see that the predictions fail to identify patch images with trees as roads when they should be. We also observe that the predictions sometimes identify parts of the parking lot or house roofs to be roads. Nevertheless, we can confirm that our model performs reasonably well.

VIII. LIMITATIONS

As we trained our model, we encountered the classic limitations of neural networks, namely training time and memory needs. The model we described in this paper, as well as its variants, is a complex model. On typical laptop CPUs, the training process would take up to a few days, which turned our attention on using the cloud computing service *Amazon Web Services*. With this solution, it would only take a few hours to train, but this was a financially costly option, and we still encountered memory allocation errors which significantly reduced our ability to scale the model even larger or to amplify our training data size with greater padding.

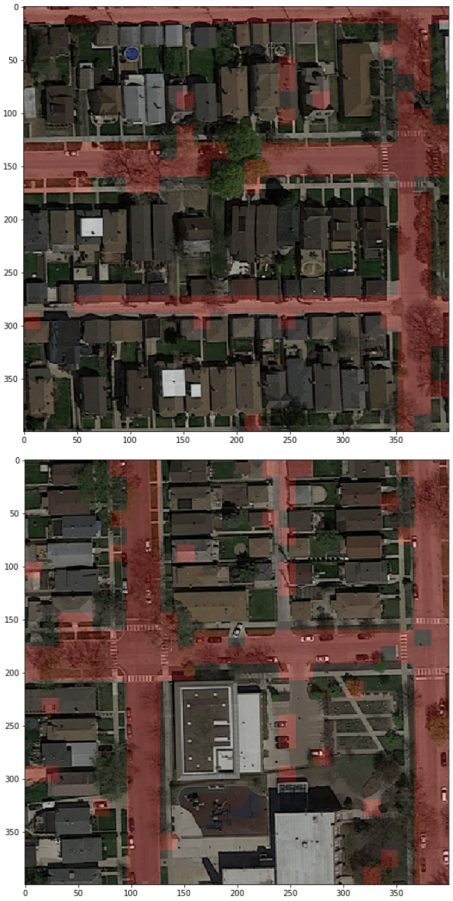

Fig. 13: Predictions (red) overlaid on example test images

IX. CONCLUSION

Our model is far from perfect. However in this report, we have developed our understanding of convolutional neural networks via first-hand experiments, with a variety of hyperparameters, which contributed to the improvement of our final model. We have experimented with different network structures, activation functions, dropout probabilities, type of optimizers, data processing methods, and ensemble models. However, our ability to grow the network and the frequency of our experiments were both significantly limited, despite using *Amazon Web Services*. We conclude by conjecturing that using a powerful GPU to develop more complex models, as well as using real-time image augmentation, may produce a higher F-score and improve the results of road segmentation from satellite images.

REFERENCES

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[2] H. Wu and X. Gu, Towards Dropout Training for Convolutional Neural Networks, Neural Networks, vol. 71, pp. 110, Nov. 2015.

[3] Karpathy, A. (2018). CS231n Convolutional Neural Networks for Visual Recognition. [online] Cs231n.github.io. Available at: http://cs231n.github.io/neural-networks-1/ [Accessed 14 Dec. 2018].

[4] Hansen, L. and Salamon, P., "Neural network ensembles," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, issue 10, pp. 2, Oct. 1990.