Pattern Classification and Machine Learning Road Segmentation

Justinien Bouron, Nicolas Casademont and Nicolas Roussel
Team : "Here come dat boi"
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—As part of the CS-433 course given at EPFL, we tackle the problem of road segmentation. To achieve road detection we use a convolutional neural network. After setting up a basic network we try different methods to improve our score. We start by the usual pre-processing methods, we add borders of context to our batches and we apply even a second convolutional neural network on the output of the first one to add even more context and straighten the discovered roads. We evaluate each of these methods and others that didn't work.**

I. Introduction

As part of the CS-433 "Pattern classification and machine learning" course given at EPFL by Martin Jaggi and Rudiger Urbanke, we tackle the problem of labeling roads on satellite images. More precisely, the goal of the project is to label patches of 16 by 16 pixels on satellite images as either roads or backgroung. The evaluation metric for this task is the F1 score.

II. Existing Works

The challenge we have at hand here, can directly be related to the more general problem of image labeling. Also, if we consider the individual labeling of each patch, the underlying task is a simple binary object classification problem. Given the time-frame of this project, we performed a non-exhaustive literature review of state of the art solutions to these problems which can be related to our challenge.

After some research we found a lot of papers and courses related to the subject at hand. Our work is mostly based on a paper and a website which help us better grasp the problem at hand.

Volodymyr Mnih in his paper "Machine Learning for Aerial Image Labeling" [1] takes on the following two tasks: road segmentation and building detection. His approach is to first use a convolutional neural network to get a prediction about the road labeling. He then performs a postprocessing operation on this predicition, by using either a convolutional neural network again or deep conditional random fields. His goal with the postprocessing step is to learn and apply the strucutres which roads and buildings usually have.

Stanford's online class CS231 [2] gives an overview of state of the art approaches for visual recognition with convolutional neural networks. It covers hyperparameter tuning as well as common architectures.

Convolutional network are the state of the art technique used in all papers we have read so far, in particular with image recognition. Their success is due to the limited amount of weights needed per layer for the result they produce. They work well with images as it uses their underlying structure instead of all pixels.

III. Validation & Comparison Metrics

Given the average training time of a neural network on the limited computational resources at our disposition, it was unreasonable to use k-fold cross validation. Hence we use a train, validation, test split of the data which is common in image recognition methods. The first one is used to train our model. The second is used for confirming that we are not over-fitting our model on the training set. Finally, the test set is our local check of the behavior of our model on unseen data, it also serves as a check that we are not over-fitting on the validation set. We will use the testing to validate our hyper-parameters as it is the most objective representation of the tested model.

As our objective is to have the best F1 score we are using it as our primary comparison metric. It is computed as follows :

$$2 * \frac{precision * recall}{precision + recall}$$

We can see that this metric relies on a weighted average between the precision and recall. Hence, in order to maximize its value, there might be a trade-off which can be used : have less false positives for more true negatives with only small changes to false / true positives. In the case of convolutional neural networks, this metric cannot be used as a loss function, because of its non-convex nature. However, the output of such a model is a probability distribution over the possible classes, in our case two : road and background. The actual prediction is then based on a threshold value; i.e., if the probability of the input being a road is above some theta, we declare it as a road. It is therefore possible to maximize the F1 score by finding the best theta which performs the best trade-off instead of just using 0.5

In order to check for over-fitting, and to know when the model has finished learning, we need to measure our score on both the training and validation set every so often, e.g after every epoch, every x iterations, etc. Given that the F1-score needs a tuning in parameters before giving its maximal value we decided to validate our score during training with accuracy.

IV. Architecture

For our neural network architecture we used [2] as an inspiration. In particular the architecture which gave the best results is inspired from VGGNet which comprises a succession of convolutional layers, max pooling steps and ends with a series of Fully Connected (FC) layers. To better visualize these concepts we use a simple representation : [l_1, l_2, ...] where l_i belongs to $\mathbb{N}$. Each number in the array represents a series of l_i convolutional layers with a fixed depth (32, 64, etc.) and with no pooling. Between all l_is, we do a simple 2 by 2 pooling to reduce the size of the input weights for the following series of layers. We then finish with a series of FC layers for which we try different sizes : one FC of 512 weights, and two FC with 1024 for both, the latter one gave worse results therefore we did not bother looking for larger FC architectures. We first start to test on a [1,1] architecture and evolve to a [2, 4, 4, 4] architecture which gives the best results for the lowest loss and best testing set accuracy as depicted in Fig. 1.

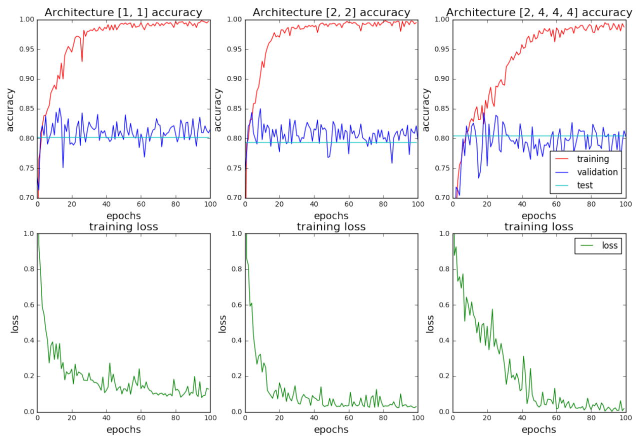

Fig. 1. Comparison of different architectures.

As we can see the improvement is minimal, in the following test we use the [1,1] architecture as it greatly speeds up our computations. There are other parameters that we fixed early on to easily test more significant improvements. Hence we will fine tune them later. The first one is the optimizer. We use the Adam optimizer as it was easy to implement and is known to converge faster than a basic momentum [2]. Moreover it does not require too much fine tuning in terms of learning rate as it adapts it itself. We also use a base 3 by 3 filter as we read it is best for image recognition type problems in most cases [2]. We try to fine tune this parameter in section IX. We also stick to a 0.8 dropout as it is widely accepted to be the most reliable value for this parameter. Again, this will be fine tuned in section IX. For our pooling layers we use only a 2 by 2 max pooling, as it was mentioned multiple times in our lectures and in papers we read that modifying it wouldnt have an impact on accuracy or f1-score.

V. Pre-processing

The input fed to our neural network can be pre-processed. Usually, for image classification no additional features are engineered and added as input - the job to discover these usefull features is left to the network itself. However the usual standardization can still be helpful. We apply a simple standardizazion on a channel-basis, meaning that all three of the green, red and blue channels are standardized independently of each other. The operation consists of removing the mean and dividing by the standard deviation.

Given the nature of the RGB color space, an interesting approach is to convert the input data from the RGB to the HSV space. The goal of this conversion is to overcome the non-uniform distance nature of the RGB color space. In short, this means that the euclidian distance between two RGB vectors is not representative of the actual perceived color distance. Contrary to the RGB space, HSV has uniform distances and might therefore produce better results. The basis of this supposition, is that the inputted data to the neural network will be more accurate, as the changes in color will better represent reality. Of course standardization can still be applied here on a per-channel basis. After some testing on different architectures we found out that HSV transformation does not improve the performances of our model.

The data is very unbalanced - about 70% of background. When given such unbalanced data, the network will not be able to compensate for it, it might simply always predict the most likely class. To overcome this effect, we balance our data by sampling it. The neural network will therefore have to entirely rely on the features of the data and not on the likelihood of seeing a class. Of course this is flawed, since there might be benefit in knowing that a class is highly unlikely. As explained in chapter III, by changing the threshold we can compensate for this.

VI. Initial Results

Using only the architecture [2, 4, 4, 4] and pre-processing steps defined in the previous chapter we are able to achieve an accuracy of the order of 0.78 on the testing set. Looking at the resulting predictions we can notice two major problems of this approach.

The first problem is the under-fitting of diagonal roads. Intuitively this can be explained by the fact that our neural network learns to recognize roads that are usually not diagonal because of the small number of such roads in the training set compared to the testing set.

The second issue is the assumption that patches are independent from each others, making the problem a pure classification of patches. This approach usually predicts wrongly when the color of a road changes due to lighting conditions like shadows of nearby building, or when an object covers a portion of the road, ie. trees or cars. The reason behind this is that our neural network cannot differentiate between a road with shadows or objects occluding it and another patch that have nothing to do with a road. While results are not terrible, improvements are possible by dropping this assumption. It is intuitive that in order to predict the class of a given patch we may use some information from its neighboring patches. As an example if we are given a single patch in which we recognize the top of a tree, we cannot distinguish if this tree is above a road or not. If we are able to see that its neighboring patches are classified as roads, we can safely assume that this patch is just a road occluded by a tree. The opposite case is analogous. Figure 1 gives examples of the two issues described above.

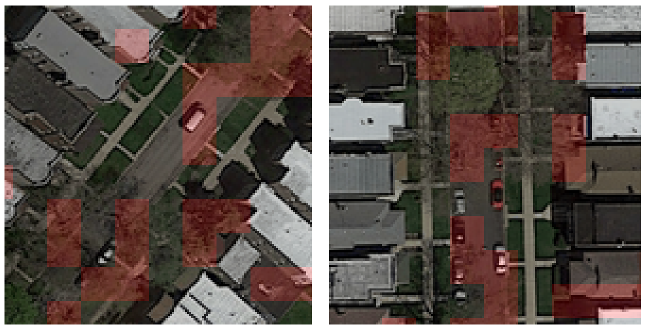
Fig. 2. On the left: example on how the NN performs on diagonal roads. On the right: example of predictions on occluded roads.

For now we will not look into fine tuning of hyper-parameters to solve those issues, but rather using techniques on the data and the computation graph.

VII. Dealing with Diagonal Roads

For the under-fitting on diagonal roads we use data augmentation on the data set we were given. This consists in adding randomly rotated images (the border pixels are dealt with reflections to avoid partial patches) to the training set, effectively balancing the amounts of diagonal and non-diagonal roads in the training set. The amount of such images added is limited to avoid having the same problem as above but with non-diagonal roads.

With this solution we are able to get an improvement of the order of 0.01 ± 0.002 on the testing set accuracy.

VIII. Patch Independence

For the patch independence, we simply changed the input of our neural network. With independence of patches, our neural network took only the pixels of a patch as input to determine its class. With dependence of patches, it takes not only the pixel of the patch in question, but also some from its neighboring patches (with reflection for border patches as in VII), creating a context for the classification, and use all those "border" pixels for the classification task. Simply put, compared to before we now decide the class of a patch by also considering the pixels around this patch and not only the ones restricted to this patch. From now on we refers to this as context enhancement.

With this addition, we achieve a big leap in performance in cases of road occlusion, we also note that we get less isolated (ie. none of its neighbors are roads) patches predicted as a road. The quantitative effect of the context enhancement, with different sizes, can be seen on Fig. 3.

IX. Fine tuning of Parameters

All hyperparameters can be fine tuned, these comprise of the filter size, dropout rate and regularization terms. All tests are ran with random rotations and context enhancement of

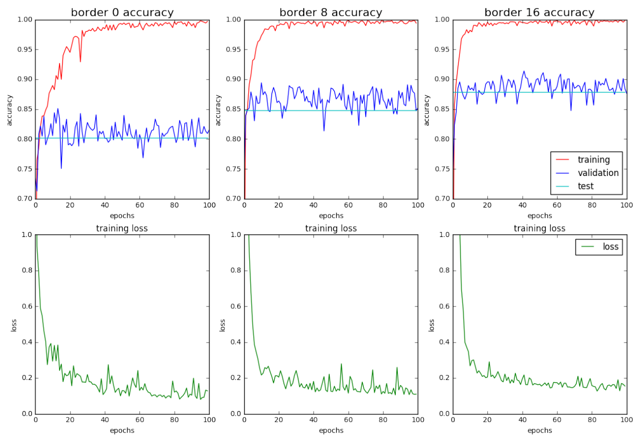

Fig. 3. Effect of adding the local neighborhood to the patches on the different validation, with different sizes.

16 pixels. Some tests are ran on multiple architectures when the effect of a parameter is ambiguous on simple architecture. Furthermore , we assume that if an improvement is seen on a simple architecture, it will also be seen on a larger one.

For the filters size, we tried 6 different sizes : 2x2, 3x3, 4x4, 5x5, 7x7 and 12x12, the testing has been done on the simple architecture [1,1]. The results are gathered in Table I.

Filter size	2x2	3x3	4x4	5x5	7x7	12x12
Test set accuracy	0.767	0.782	0.772	0.757	0.745	0.698

TABLE I
Test set accuracy for different filter size.

The explanation is rather simple, as the filter gets bigger, there are more parameters in the model, the risk of over-fitting gets higher, hence the validation score decreases. A filter size of 3 seems to be optimal.

To understand the effect of the dropout rate on the predictions, we tested to change the keep probabilities from 0.5 to 0.8. After running on all architectures, the measured differences between test accuracies are reported in Table II.

Architecture	[1,1]	[2,2]	[2,4,4]
Delta on test set accuracy	+0.06	+0.09	+0.14

TABLE II
Test set accuracy change for a dropout rate from 0.5 to 0.8 on different architectures.

Thus we decide to take a dropout of 0.8, as commonly used in practice.

The effect on changing the regularization term is negligible. The difference on test set accuracy is negligible when comparing for λ between 0 and 5.10^{-4} on a logistic scale, the biggest absolute difference on the test set accuracy is under 1%.

As seen in section VIII, image border size improve drastically the prediction performances. The only limit is the computation power as the memory required is exponential to the image border size.

The remaining parameters were left as their original values, for instance the stride size is 1 as we cannot get anything better that that. The batch size has nothing to do with the prediction performance but is rather a trade-off between convergence time and computation power.

X. Post-processing

On the outputted results of the previous chapter some obvious flaws of our model still appear. The problems explained in chapter VI, where an object covers the road still produces errors and additionally some patches are predicted as roads even thought they have no other road-patch in their neighbourhood. This can again be explained by the lack of context, the border used to solve this problem is still not large enough. Given the computational limits of our resources we cannot continue to further increase the border. Our intuition is to use a salt and pepper filter, but the parameters for such a filter are very unclear, and might therefore even worsen our results. Therefore, similarly to [1] we use a second convolutional neural network, whose purpose it to learn the structure of roads and will therefore fill in holes in our predictions and remove structures which typically are not roads. The input of this network are the labels of our predictions. Given their binary nature we can compress the original 400 by 400 labeled image into a 25 by 25 image. This solves the computational issue we originally had, since such a low resolution image allows us to easily have a large neighborhood.

The results of this step vary widely depending on the initial predictions - very good initial predictions are not improved as much by post-processing as decent initial predictions. Our best initial prediction was improved by 0.01 ± 0.003.

An example on the post-processing step can be seen in Fig. 4.

XI. Conclusion

In conclusion we achieved the score of approximately 0.91 for the F1 score on the test set. From our results context enhancement was a very important part. Additionally the post-processing step improve the results even further by repairing the most obvious faults of our predictions.In future work it would be interesting to maybe improve post-processing work as there are still obvious faults. And see what a greater computation power could achieve.

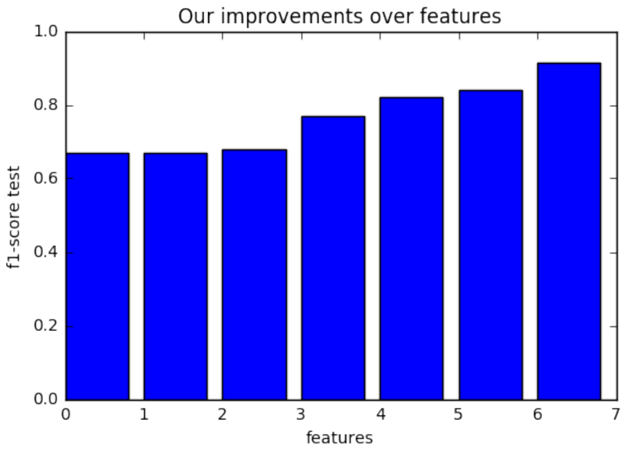

Fig. 5. Our F1-scores on the testing set through our different improvements. From left to right : [1,1], [2,2], [2,4,4,4], border 8, border 16, rotations, post processing

XII. References

[1] Machine Learning for Aerial Image Labeling by Volodymyr Mnih 2013

[2] http://cs231n.github.io/convolutional-networks/

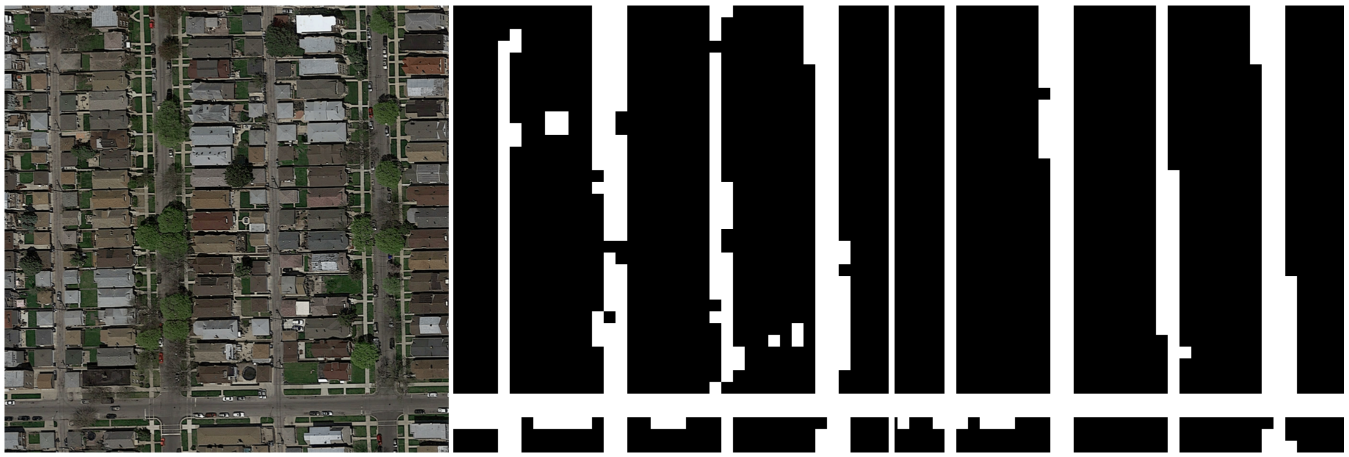

Fig. 4. Example on a recovery from the post-processing step. The first image on the left is the original satellite image, the one on the center is the prediction from our first neural network, the image on the right is the "corrected" version of this prediction by the post-processing neural network.