Project 2: Road Segmentation

Nicolas Brandt, Leo Serena, Rodrigo Granja

I. Introduction

Since the dawn of mankind, humanity has always dreamed of being able to automatically segment roads. We chose to tackle this task for our second project, via the AIcrowd road segmentation challenge. It is a segmentation task, where we need to predict whether 16x16 patches of satellite images correspond to a road or not. We use a segmentation specific neural network, U-net, with the EfficientNet-B7 backbone, data augmentation, loss function selection, cyclical learning rates and snapshot ensembles, external data as well as ensembling to achieve a competition f1 score of 0.922, which is tied for first place as of this writing.

II. Data sets

Main data set: The data set consists of satellite images. The training set has 100 images of dimension 400x400 with 3 channels, with labels 400x400 binary masks showing where the roads are. The test set consists of 50 images of dimension 608x608 with 3 channels.

External data: In addition the to main data set, we used the Data Massachusetts Roads Dataset (https://www.cs.toronto.edu/~vmnih/data/). The data set consists of 1107 satellite images of dimension 400x400 with 3 channels and the corresponding binary masks labels for the roads.

III. Methods

A. Preprocessing

The data was preprocessed by scaling to [0,1]. Training images were rescaled to 384 for U-net.

B. U-net

The main neural network model we explore is U-net [1], which is designed specifically for segmentation, notably when data is scarce. The network consists of a contracting and an expanding path. The contracting path is fully convolutional, and consists of the usual convolutional neural network architecture, where the input size is progressively decreased and the number of channels increased. The expanding path then consists of upsampling layers, increasing the image size and reducing the number of channel, in order to predict a mask with the size of the original input. Known neural networks can be used to create a U-net in the following way: we take a known architecture (the "backbone"), such as ResNet34, replace its fully connected layers with convolutions layers, and add the upsampling layers. The main backbone we use are variants of EfficientNet [2]. These recent models have shown state-of-the-art results while using a lower number of parameters than typical large models, by introducing a method to efficiently scale networks parameters such width and depth. We additionally explore SEResNeXt ([3], [4]), ResNet34 [5] and VGG19 [6].

C. Loss functions

Different loss functions can be used for segmentation. The loss functions we explore are binary cross-entropy (BCE), Jaccard, DICE and focal:

- BCE: $L_{BCE} = -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$
- Jaccard: $L_{Jac} = 1 - \frac{y\hat{y}}{y+\hat{y}-y\hat{y}}$
- DICE: $L_{DICE} = 1 - \frac{2y\hat{y}}{\hat{y}+y}$
- $L_{Focal} = -\left(\alpha(1-\hat{y})^{\gamma} y\log(\hat{y}) + (1-\alpha)\hat{y}^{\gamma}(1-y)\log(1-\hat{y})\right)$

where y is the label and $\hat{y}$ is the prediction. We also explored combination of these losses.

D. Cyclical learning rate and snapshot ensembles

Algorithms used to train large neural networks are typically based on stochastic gradient descent (SGD). As such, they suffer from variance in the estimate of the gradient and the learning rate must be decayed. A typical way to do this is by reducing the learning rate by a certain factor at at certain number of epoch, for instance by a factor of 10 at epoch 150 and 225 such as in [5]. We explore another technique, based on cyclic learning rate scheduling and snapshot ensembling [7]. Cyclic learning rate scheduling works in the following way: an initial learning rate is chosen and the learning rate is decayed, then reset back to its original value and decayed again, until a specified number of cycles is reached. To decay the learning rate, we use cosine annealing as in [7]:

$$\gamma(t) = \frac{\gamma_0}{2}\left(\cos\left(\frac{\pi \bmod (t-1, \lceil\frac{T}{M}\rceil)}{\lceil\frac{T}{M}\rceil}\right) + 1\right), \qquad (1)$$

where γ_0 is the initial learning rate, T the total number of training steps and M the number of cycles. The rationale is the following: we allow the model to converge at a minimum, then "shoot" out when the learning rate is reset, and have it to converge at another minimum. Different minimum may have a similar loss, but can be expected to make different errors [7]. By saving the model at each of these minimum, ie before the learning rate is reset, we can achieve model diversity and use an ensemble with a single training pass.

E. Data augmentation

Data augmentation is a useful technique to help generalization when data is scarce, which consists of creating fake training examples to increase the size of the training set [8]. Its use makes the model more robust and helps it learn which transformations it should be invariant to [1]. The transformations to use is task specific; flipping can be problematic for classifying digits for instance, as a 6 could become a 9. The data augmentation techniques we explore are: horizontal/vertical flipping, cropping, shifting, rotating, scaling, blurring, change in contrast, change in gamma and distortions.

F. Test time augmentation

Test time augmentation (TTA) is a technique that involves augmenting the test set. Instead of predicting only on the test set images, we also predict transformed version of these images and merge the predictions [9]. The aim is to make predictions more robust.

G. Ensembling

Ensembling consists of combining different models to improve the overall prediction. We use soft voting ensembling, where the final prediction is determined by averageing the probability prediction of individual models.

IV. Experimental results

For local cross-validation, we split the full training set into a local training set and validation set, using a 50/50 split. We did not opt to use k-fold cross-validation, as we wished to keep the same cross-validation method throughout and the k-fold cross-validation runtime was prohibitively long for higher number of iterations. Unless otherwise specified, the f1 values for the local training set are referred to as "train f1" and the f1 values for local validation set are referred to as "val f1". In part IV-A competition scores are with the model trained on the local training set only unless specified otherwise. In part IV-B, competition scores are obtained with the model trained on the entire training set. To determine if a 16x16 patch was a road or not for the competition test set, we did the following: we chose an initial cutoff of 0.25, and if the average model prediction for pixels in this patch was above the cutoff value, we assigned to the patch a road label.

A. Baseline results: CNN, U-net, backbones, losses, optimization, snapshot ensemble, TTA

Convolutional neural network (CNN): We very first used the proposed approach from the provided Tensorflow tutorial: splitting the training images in patches of 16, with label for the patch being an average of the mask for this patch, and training a CNN on this new data set. This approach showed however very poor performance (competition f1 score of under 0.4). A reason for this is obvious: with input only being 16x16 patches, the model is very limited in the information it is provided; it is not possible to distinguish for instance a grey roof from a road. As this approach was ineffective, we did not investigate it further and quickly moved on to U-net.

U-net baseline with different backbones: We first experiment with different backbones for U-net. All backbones were pretrained on imagenet. We trained for 50 epochs, using Adam with the default learning rate of 0.001 and a batch size of 2. For loss function, we used BCE. Results are shown in Table I. EfficientNet-B7 shows the clear best performance, both in terms of validation and competition f1 score. We can also note that the older models, ResNet and VGG, show weak performance compared to newer networks.

Backbone	EfficientNet-B7	EfficientNet-B5	SEResNeXt101
Train f1	0.9321	0.9306	0.9328
Val f1	**0.7917**	0.7884	0.7786
Comp. f1	**0.867**	0.862	0.844

Backbone	SEResNeXt50	ResNet34	VGG19
Train f1	0.9348	0.9181	0.8276
Val f1	0.7809	0.7668	0.6838
Comp. f1	0.850	0.834	0.764

Table I
U-NET BASELINE WITH DIFFERENT BACKBONES

Loss functions: We now explore the performance of different loss functions, using EfficientNet-B7 and the same training scheme as before. We also explore combination of loss functions, where the loss functions are summed. Selected results are shown in Table II. We can see that the Jaccard loss outperforms the other losses in all training, validation and competition score.

Loss	BCE	Jac.	BCE+Jac.	BCE+DICE	Foc.+Jac.
Train f1	0.9321	0.9478	0.9408	0.9410	0.9394
Val f1	0.7917	**0.8141**	0.8001	0.7937	0.7903
Comp. f1	0.867	**0.872**	0.854	0.854	0.853

Table II
LOSS FUNCTIONS

Cyclical learning rate and snapshot ensembles: We use the best settings from the previous sections, but increase the number of epochs to 300 and use 6 cycles, as used in [7]. For the ensemble, we use all 6 snapshots. Results are shown in Table III, We can note that the final model after 6 cycles shows better generalization than without learning rate decay, and that ensembling further improved the competition f1.

Data augmentation: In this section we show the impact of different data augmentation schemes. We train for 300 epochs and use cyclic learning rate and show the results at the end of training, without ensembling. We showcase three different data augmentations. First: flips, rotations and blurring (**simple**); then, we add shifts, scaling, change in

Decay	no decay	6 cycles	6 cycles, ensembled
Train f1	0.9598	0.9618	-
Val f1	0.8078	**0.8176**	-
Comp. f1	0.861	0.868	**0.869**

Table III
LEARNING RATE DECAY

contrast and gamma (**lum.**); then add distortions (**dist.**). Results are in Table IV. We can note that data augmentation indeed helps reducing overfitting: with more augmentation, our training score decreases, and both the validation and competition score increase; except for **dist.**, where all f1 scores decrease. This can be expected; distortions are likely to not be an appropriate data transformation, as roads are usually straight. More details on the values used for the **lum.** augmentation scheme can be found in the *script.py* script.

Augmentation	no aug.	simple	lum.	dist.
Train f1	0.9618	0.9482	0.9427	0.9149
Val f1	0.8176	0.8222	**0.8269**	0.8195
Comp. f1	0.868	0.892	**0.900**	0.893

Table IV
DATA AUGMENTATION

Momentum and tuning the learning rate: Although Adam is a fast optimizer, it may be inferior to SGD (including with momentum) for generalization [10]. Using SGD instead of Adam (both with well tuned learning rates) resulted in a consistent improvement on the generalization scores, which we illustrate from Table V. We show results for different learning rates for SGD at the end of training with cyclical learning rates, without ensembling, with no data augmentation, and default momentum value of 0.9. We can note that 0.1 is the best learning rate, and that despite having similar training f1 score than Adam ("6 cycles" column in Table III; learning rate tuning for Adam not shown, but the default value of 0.001 was best), SGD has both a higher validation and competition f1 score.

Initial learning rate	1	0.1	0.01
Train f1	0.9607	0.9620	0.9594
Val f1	0.8272	**0.8340**	0.8029
Comp. f1	-	**0.878**	-

Table V
LEARNING RATES

Current best result and TTA: In this section we show in Table VI our current best result using the previous techniques (EfficientNet-B7, Jaccard loss, SGD with momentum and learning rate of 0.1, ensemble of 6 snapshots and **lum.** augmentation scheme). We train on the entire training set to help generalization, and show that we can improve further the score using TTA. For TTA, we use horizontal flip, as well as horizontal and vertical flips. As horizontal TTA is most common, and we did not find benefit from adding vertical flips, we kept only horizontal for the next sections.

Decay	no TTA	horizontal	horizontal + vertical
Comp. f1	0.908	**0.909**	**0.909**

Table VI
TEST TIME AUGMENTATION

B. Going further: external data, cropping, model ensembling

We explore how to improve the competition f1 score further using external data, cropping and higher batch sizes, model ensembling and tuning the cutoff. For local cross-validation, we looked at the f1 score at the end of the first training cycle. Competition f1 scores where determined by running the model for 6 cycles on the entire training set and using snapshot ensembling. The snapshot ensembles used all the snapshots. We used the best settings as obtained from the previous sections.

Additional data: In this section we explore the benefit from external data. As the size of the external data set is large compared to the training set, we need to determine what is the appropriate number of images to add. We test a number corresponding to half, same and twice the size of the set used for training (50 for local cross-validation, 100 for competition). Results are shown in Table VII and Table VIII.

Added images	0	25	50	100
Train f1	0	0.7215	0.7144	0.7072
Val f1	0	**0.8504**	0.8044	0.6502

Table VII
EXTERNAL DATA FOR CROSS-VALIDATION

The validation shows that the number of additional images should stay low, which is confirmed on the competition score, as in both cases the performance quickly deteriorates when too many images are added. The optimal amount of images to add however differed for local cross-validation and competition. We were not able to generate a local cross-validation that completely accurately correspond to the competition score.

Added images	0	50	100	200
Train f1	0	0.7733	0.7166	0.7419
Comp. f1	0	0.908	**0.912**	0.858

Table VIII
EXTERNAL DATA FOR COMPETITION

Cropping and larger batch size: We then explore cropping and larger batch size, as cropping allows to reduce the size of the image. The batch size we used for a given crop size was the highest possible. We did not use external data. To further explore batch sizes, we also used a smaller network (EfficientNet-B5). As the number of iteration per epoch decreases for larger batch sizes, we increased number of iterations per cycle for larger batch sizes compared to the usual 50 epochs/cycles. Results are shown in Table IX. We can note that a too small crop dimension (96) hurts performance, and that the best result is obtained by using the best network (B7) with batch size of 20 and dimension of crops of 128. We can note that using a better network (B7) was more important than larger batch sizes. Interestingly, using both cropping and data augmentation resulted in lower performance (not shown); we believe this may be due to the external data often having certain "patches" of the images missing (ie only white).

Crop dim.	**96**	**128**	**192**	**256**
Batch size	**40**	**20**	**10**	**4**
Net. model	**b7**	**b7**	**b7**	**b7**
Epoch/cycle	**400**	**400**	**200**	**100**
Train f1	0.8656	0.8885	0.9001	0.9023
Val f1	0.8274	0.8367	**0.8418**	0.8384
Comp. f1	0.908	**0.913**	0.909	0.909

Crop dim.	**192**	**256**	**128**
Batch size	**18**	**10**	**40**
Net. model	**b5**	**b5**	**b5**
Epoch/cycle	**400**	**200**	**400**
Train f1	0.9123	0.9109	0.8937
Val f1	0.8233	0.8375	0.8449
Comp. f1	0.909	0.908	0.907

Table IX
CROPPING AND BATCH SIZE

Ensemble: We experiment with combining the prediction of different models. We use a soft-voting scheme, where the predicted value $[0, 1]$ of the different models is averaged and the cutoff then applied. Results are shown in Table X. The models we tried to combined are: the best (in terms of competition f1) from the data augmentation part (model 1), the best from the cropping part (model 2), the best b5 network from the cropping part (model 3). We can note that more models is not better, and that our best result was averaging only for model 1 and model 2. This can also illustrate the importance of model diversity for ensembling, as the two models where trained on different data, with different data augmentation and using a different optimization (batch size, cycle iterations).

Tuning the cutoff: In this final section we tune the cutoff. Results are shown in table XI We can note that a slightly higher cutoff improves the result on the competition score, with best results with cutoff 0.28-0.30. We can infer that with the initial cutoff we were predicting too many non-road patches as road (ie too many false positives), and that our f1 score was improved by increasing our precision.

Models	**1+2**	**1+2+3**	**100**	**200**
Comp. f1	**0.918**	0	0	0

Table X
MODEL ENSEMBLING

Cutoff value	**0.25**	**0.27**	**0.28**	**0.29**	**0.30**	**0.32**
Comp. f1	0.918	0.921	**0.922**	**0.922**	**0.922**	0.921

Table XI
TUNING THE CUTOFF

V. DISCUSSION

We obtained a competition f1 score of 0.922. We noted among other: the importance of choosing the best performing model, an appropriate loss function, the right optimizer and learning rate schedule, of choosing the appropriate data augmentation, the particular benefits from appropriate cropping and the importance of model ensembling (and that more models is not necessarily better), as well as the difficulty of finding an effective local cross-validation method. We described the main techniques we used, but others were explored, although not as in depth, such as:

- Less than 6 cycles (4), which always proved worse when tried
- More cycles (10), which was also worse; we did not search for finer (ie 7,8) cycle number or finer cycle length, notably due to the high runtime
- Using different numbers of snapshots for the ensemble, using all snapshots was consistently better

Other techniques not explored include: 1) different preprocessing than scale to $[0, 1]$ (we briefly tried scaling to $[-1, 1]$, which showed worse performance); 2) using road segmentation specific additional features or preprocessing steps; 3) using traditional learning rate scheduling; 4) architectures other than U-net; or 5) finer selection of which additional images to add (for instance, not adding images that had missing parts). Although we obtained a strong public score of 0.922, our final steps required us to tune using the public f1 score. This is in part due to the difficulty of detecting small improvements on the local cross-validation and in part for runtime reasons, as it requires to train the model twice (one time on the local training set, one time on the entire training set). As a result, there may be overfitting on the competition data set used for the public score.

REFERENCES

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[2] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[3] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[4] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[9] N. Moshkov, B. Mathe, A. Kertesz-Farkas, R. Hollandi, and P. Horvath, "Test-time augmentation for deep learning-based cell segmentation on microscopy images," *bioRxiv*, p. 814962, 2019.

[10] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.