Deep U-Net Ensemble for Road Segmentation

Group: The Predictors
Department of Computer Science
ETH Zurich, Switzerland

Marialena Kadoglou
ETH Zurich
mkadoglou@ethz.ch

Younis Khalil
ETH Zurich
khalily@student.ethz.ch

Neville Walo
ETH Zurich
walon@student.ethz.ch

Laura Wülfroth
ETH Zurich
wlaura@student.ethz.ch

***Abstract*—ToDo write abstract... Done in the end.**

I. Introduction

Image segmentation is one of the most studied problems in computer vision. The main goal in image segmentation is to classify each pixel into a specific class. It has many use cases in medical images analysis and in object detection used for example by self-driving cars. In this work, we address the problem of road segmentation in aerial images.

There are a large number of satellites orbiting the Earth every day, so there is a huge dataset of up-to-date aerial images of the world. All over the world, thousands of kilometers of new roads are built every day, but manually adding these roads to a road network can be a tedious and time-consuming process. Direct classification of roads from satellite imagery can speed up the process of creating and maintaining up-to-date road maps.

In this paper, we present a novel approach on how well-known convolutional neural networks (CNNs) can be used in a U-Net ensemble to achieve state-of-the-art road segmentation performance. We present our approach to the Kaggle competition *ETHZ CIL Road Segmentation 2021* of the course 263-0008-00L Computational Intelligence Lab at ETH Zurich.

II. Related Work

TODO:Talk about previous attempts to road segmentation and CNN.

A. Standard Neural Networks

- AlexNet [9]
- VGG-16 [20]
- GoogleLeNet or Inception [16]
- Xception [4]
- EfficientNet [17]

B. U-Net Architectures

- U-Net CNN [13]
- GC-DCNN [10]
- U-Net CNN with standard NNs from above (EfficientNet + U-Net [1], Inception + Encoder/Decoder [2])

III. Dataset

This section presents the datasets used to train and evaluate our models. We trained and evaluated our models using 2 different datasets: Kaggle Dataset and Google Dataset.

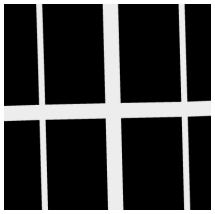
A. Kaggle Dataset

This dataset is the dataset originally provided by the Kaggle competition. In the rest of the document, we will refer to it as the *Kaggle* data.

The training dataset of the Kaggle dataset consists of 100 RGB images with a resolution of 400×400 pixels. The training dataset also contains the corresponding 400×400 pixel groundtruth mask of each aerial image, where pixels that are a road are assigned a value of 1 and pixels that are not a road are assigned a value of 0. Fig. 1 shows an example of the training dataset of the Kaggle data.

The test dataset of the Kaggle data consists of 94 RGB images with resolution 608×608 pixels.

(a) Training Image



(b) Training Mask

Fig. 1. Training image example from the Kaggle dataset.

B. Google Dataset

Since the Kaggle training dataset contains only 100 images, we created another dataset for which 1156 images and their corresponding mask were collected. In the rest of the document, we refer to this dataset as *Google* data.

For this dataset, we followed the same specification as for the Kaggle dataset, so it can be used as a drop-in replacement for training. The images and mask in the Google dataset were gathered using the Google Maps API. The images are from different cities in the US and were selected to have the same zoom level and structure as the Kaggle data.

IV. Data Processing

This section introduces the data processing used in our pipeline. The data pipeline used to feed our models with data and then make predictions consists of several stages. During training, we first perform general preprocessing and some image augmentation. To create a submission, some postprocessing must be performed.

A. Preprocessing

In a first preprocessing step, the satellite images are resized to match the input dimensions of the model and the masks are resized to match the output dimensions of the model. For both resize operations, bilinear interpolation with antialiasing is used. If not performed by the model itself, the input image and mask are divided by 255 so that all inputs and masks are in the interval $[0, 1]$.

B. Augmentation

To increase the diversity of the training images, the following image transformations are performed to each batch.

1) Rotation: The image is rotated by either $0°, 90°, 180°$ or $270°$. Each angle has the same probability.

2) Flipping: The image is flipped horizontally with probability 0.5 and also flipped vertically with probability 0.5.

3) Brightness: The brightness of the image is adjusted by uniformly choosing a factor in the interval $[-0.2, 0.2)$.

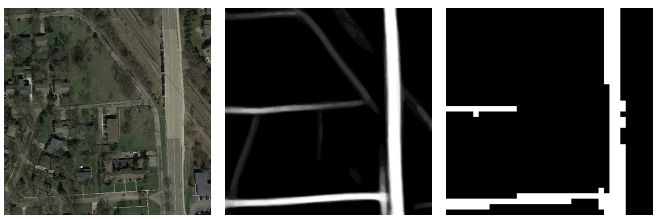
4) Contrast: The contrast of the image is adjusted by uniformly choosing a factor in the interval $[0.0, 0.5)$.

5) Hue: The hue of the image is adjusted by uniformly choosing a factor in the interval $[-0.2, 0.2)$.

6) Saturation: The saturation of the image is adjusted by uniformly choosing a factor in the interval $[0.0, 0.5)$.

C. Postprocessing

To create a submission, the output of the model was resized to 608×608 pixels using bilinear interpolation with antialiasing. The output was multiplied by 255 so that an actual image can be reconstructed. In the Kaggle competition, rather than submitting the entire mask as a prediction, we are asked to predict whether each 16×16 pixel patch is road or not. The submissions are created with a foreground threshold of 0.25 (default value in Kaggle competition), meaning that if more than 25% of the pixels in a 16×16 patch are road, the entire patch would be classified as a road. Fig. 2 shows an example of creating a submission from a test image.



(a) Test Image (b) Prediction (c) Submission

Fig. 2. Creating a submission from a test image.

V. Model

This section presents the models we trained and the baselines we compare against. We first describe the U-Net architecture, which is a basic building block for the models we created. Then we describe the modifications in detail, followed by the baselines.

A. U-Net architecture

A typical U-net consists of a downsampling step (encoder) and an upsampling step (decoder), see Fig. 3. The encoder follows the standard idea known from convolutional neural networks: The input image is downsampled using convolutional layers, pooling layers and activation functions, which reduces the spatial dimension but increases the depth of the feature map. On the other hand, the decoder then increases the spatial dimension but decreases the depth of the feature maps by using upconvolutions (also sometimes called transpose convolutions or deconvolutions). To facilitate the flow of information in the network, the various stages in the encoder and decoder are interconnected, giving the network its characteristic U-shape. The connection from the encoder to the decoder can be understood by imagining long skip connections, see [4]. However, the big difference to skip connections is that the connections are not added, but concatenated. As a final layer, a *softmax* or *sigmoid* activation is used to predict class probabilities for each pixel.

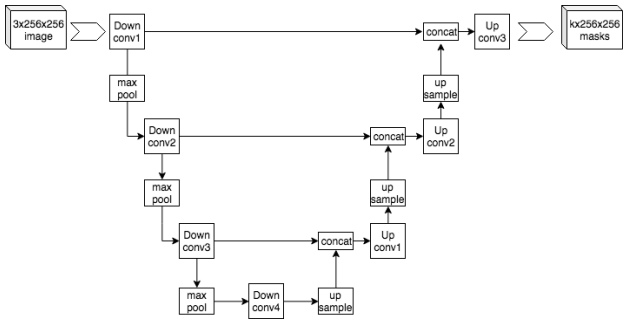


Fig. 3. Example architecture of a U-Net. Source: [18].

B. Xception models

In these models, we used the popular Xception [4] CNN as an encoder in a U-Net architecture. As a decoder, we used the standard approach of the default U-Net, but enriched it with the common features used to improve deep neural networks: Batch Normalization [7], Skip Connections [4] and Dropout [14]. As an activation function in the decoder, the *LeakyReLu* [11] is used.

We created several versions of this model with different input and output sizes. We also created a version using the pre-trained weights from the imagenet competition. Table I shows all the models we have created using the Xcpetion [4] CNN as an encoder. In the rest of the document, we will refer to these as the *Xception* models.

TABLE I
THE MODELS WE CREATED USING THE XCPETION [4] CNN AS AN ENCODER IN AN U-NET ARCHITECTURE.

Model	Input Size	Trainable Parameters
Xception 128	128	$38,370,009$
Xception 256	256	$38,370,009$
Xception 400	400	$38,370,009$
Xception Pretrained	299	$23,059,377$

C. EfficientNet models

In these models, we have taken the same approach as in the Xception models, but instead we use the EfficientNetB4 [17] CNN as the encoder. The decoder architecture is the same as for the Xception models. We took the EfficientNetB4 over its siblings because it is the CNN in the EfficientNet family whose input is closest to 400 (the size of our images).

Similarly to the Xception models, we created several versions of this model, see Table II. In the rest of the document, we will refer to these as the *EfficientNet* models.

TABLE II
THE MODELS WE CREATED USING THE EFFICIENTNETB4 [17] CNN AS AN ENCODER IN AN U-NET ARCHITECTURE.

Model	Input Size	Trainable Parameters
EfficientNet 128	128	$25,924,673$
EfficientNet 256	256	$25,924,673$
EfficientNet 400	400	$25,924,673$
EfficientNet Pretrained	380	$22,133,169$

D. Ensemble

To further improve the predictive performance of our models, we created several ensembles from the best models we trained. Creating an ensemble lowers variance and averages out bias, which usually leads to better performance. This was achieved by saving the output probabilities of each model and averaging them. Fig.2 shows the output of on an ensemble.

E. Baselines

We compare our approach against 2 baselines.

1) CNN on Patches: In this baseline, 16×16 patches are fed into a small CNN with 3 convolutional layers followed by 2 linear layers. The network relies on Max Pooling to grow the receptive field and on Batch Normalization [7] and Dropout [14] to stabilize the training. The final linear layer outputs a scalar through a sigmoid activation. Fig 4 shows the architecture of this baseline.

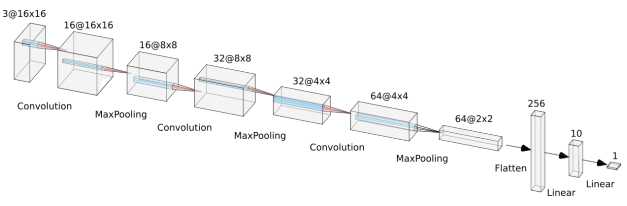


Fig. 4. Architecture of the baseline CNN.

2) U-Net: This baseline represent a default U-Net architecture similar to the one presented in [13]. This implementation differs from the original by introducing Batch Normalization [7] layers and padded convolutions to avoid cropping features. The input and output size is 384×384 pixels.

VI. EXPERIMENT

This section presents how the models were trained, tuned, and evaluated.

A. Evaluation

To evaluate the performance of our models, we submitted the predictions to the Kaggle leaderboard, which calculated the public F1 score of 49% of the test data.

We chose not to use a validation dataset because we already have limited training data and a small validation dataset would not be very informative.

B. Training

The Xception and EfficientNet models were implemented using Tensorlflow [12] and Keras [3]. The image pipeline, as described in Sec. IV, is also fully implemented in Tensorlflow [12]. To keep track of all our experiments and easily compare and organize them, we use Comet [5].

Due to the unbalanced nature of the problem, the dice loss [15] was used to learn the best model weights. The dice loss is an adjusted version of the F1-score which can be used in segmentation tasks. We used a generalized version of the dice loss with the smoothing coefficient 1, which can be calculated as follows:

$$1 - \frac{2 \cdot |A \cap B| + S}{|A| + |B| + S}$$

where S is the smoothing coefficient.

We used the Adam [8] optimizer with its default parameters ($\beta_0 = 0.9, \beta_1 = 0.999, \epsilon = 1e-7$) to perform gradient descent.

Since only 8 submissions per day are allowed on the Kaggle leaderboard, we were limited in the amount of hyperparameter tuning we could perform. The only tuning we conducted was on the number of epochs and the learning rate of the Adam [8] optimizer.

VII. RESULTS

In this section, we present and comment on our results.

A. Prediction Performance

Table III shows the public scores of the models and baselines. We report the max, mean and standard deviation of the public score of each model over all the experiments we conducted. Fig. 5 shows the maximum achieved public score of the models and baselines.

We can see that the baselines already perform reasonably well when fed with enough training data. The U-net approach seems to be the right way to tackle image segmentation tasks since it is able to capture global dependencies, which a simple CNN on patches cannot. Overall, adding more training data really improves performance, even when using pretrained

encoders. Training on higher resolution images also improves performance, but the advantage gets smaller as resolution increases. The EfficientNet models slightly outperform the Xception models in the best case, but the Xception models perform better on average and have less variance. The best performance is achieved by combining the best 20 models into one ensemble.

TABLE III
PUBLIC SCORES OF THE MODELS AND BASELINES.

Model	Dataset	Max	Mean	Std. Dev.
CNN on patches	Kaggle	0.79537		
CNN on patches	Google	0.84150		
U-Net	Kaggle	0.85905		
U-Net	Google	0.90998		
Xception 128	Google	0.91015	0.90362	0.00739
Xception 256	Google	0.92097	0.91707	0.00343
Xception 400	Google	0.92593	0.92252	0.00168
Xception 400	Kaggle	0.87608	0.86877	0.01331
Xception Pretrained	Google	0.91722	0.90732	0.00792
Xception Pretrained	Kaggle	0.85746	0.82555	0.03562
EfficientNet 128	Google	0.90288	0.88515	0.01576
EfficientNet 256	Google	0.92282	0.90046	0.02756
EfficientNet 400	Google	0.92671	0.91968	0.00372
EfficientNet 400	Kaggle	0.86044	0.84797	0.01248
EfficientNet Pretrained	Google	0.92039	0.91734	0.00234
EfficientNet Pretrained	Kaggle	0.87522	0.85011	0.02503
Top 3 Xception		0.92805		
Top 3 EfficientNet		0.92853		
Top 3 Ensemble		0.92897		
Top 5 Ensemble		0.92972		
Top 10 Ensemble		0.93002		
Top 20 Ensemble		**0.93094**		

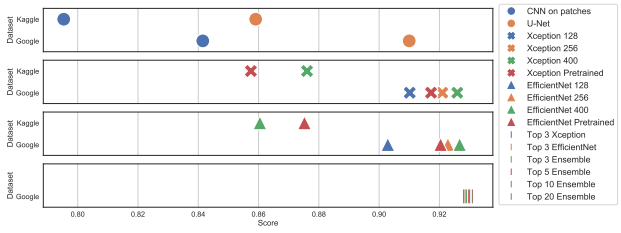


Fig. 5. Maximum public score of the models and baselines.

B. Convergence

Fig. 6 shows the convergence behaviour the Xception and EfficientNet models. We can see that the EfficientNet models require many more epochs to converge and are generally more difficult to train. The Xception models always converge very quickly.

We found that the Xception models performed best at a learning rate of 0.0001 with approximately 90 epochs. The EfficientNet models performed the best at a learning rate of 0.001 and about 160 epochs.

VIII. CONCLUSION

In this work, we presented improvements to the basic structure of a U-Net to successfully solve the road segmentation task. We applied data augmentation and collected additional training data to further improve the result. We have shown that well-known CNNs can be used as encoders in a U-Net and perform well. Finally, we obtained our best model by combining our top 20 models in an ensemble to achieve a score of 0.93094 on the public leaderboard.

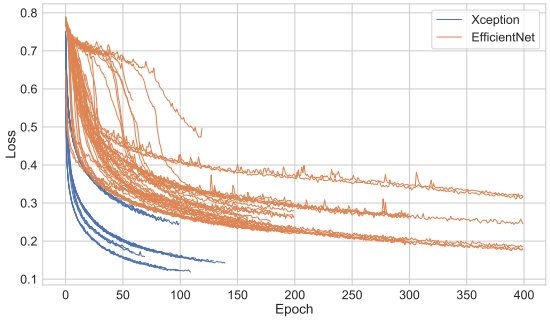


Fig. 6. Convergence behavior of Xception and EfficientNet models.

IX. FUTURE WORK

There has been a lot of research in the area of image segmentation, which has led to new, more powerful variants of the U-Net, such as U-Net++ [19] or UNet 3+ [6]. In this work, we have only focused on the original U-Net architecture, but it would definitely be worth investigating how these newer methods perform in road segmentation. It would also be worth investigating how other well-known CNNs perform in a U-net and how they behave with different input sizes and different training data.

REFERENCES

[1] Bhakti Baheti et al. "Eff-UNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 1473–1481. DOI: 10.1109/CVPRW50498.2020.00187.

[2] Liang-Chieh Chen et al. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv: 1802.02611 [cs.CV].

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

[4] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. arXiv: 1610.02357 [cs.CV].

[5] Comet.ML. *Comet.ML home page*. 2021. URL: https://www.comet.ml/ (visited on 06/10/2021).

[6] Huimin Huang et al. *UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation*. 2020. arXiv: 2004.08790 [eess.IV].

[7] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada, 2012, pp. 1097–1105.

[10] Meng Lan et al. "Global context based automatic road segmentation via dilated convolutional neural network". In: *Information Sciences* 535 (2020), pp. 156–171. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.05.062. URL: https://www.sciencedirect.com/science/article/pii/S0020025520304862.

[11] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[12] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[14] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[15] Carole H. Sudre et al. "Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations". In: *Lecture Notes in Computer Science* (2017), pp. 240–248. ISSN: 1611-3349. DOI: 10.1007/978-3-319-67558-9_28. URL: http://dx.doi.org/10.1007/978-3-319-67558-9_28.

[16] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[17] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[18] Wikipedia. *U-Net*. 2021. URL: https://en.wikipedia.org/wiki/U-Net (visited on 06/10/2021).

[19] Zongwei Zhou et al. *UNet++: A Nested U-Net Architecture for Medical Image Segmentation*. 2018. arXiv: 1807.10165 [cs.CV].

[20] Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014).