Foundations of Deep Learning
Kaggle Competition Report

Master in Data Sciences and Business Analytics
ESSEC and CentraleSupélec

Xinyu HU
Miao WANG
Sauraj VERMA
Yixin ZHAO

2021.01.17

Abstract

Our task in this Kaggle competition is to apply deep learning techniques to segment aerial images of residential neighborhoods into 25 segments at pixel level. We first preprocessed the images with resizing, normalization, and data augmentation. Multiple models, including U-Net, Attention U-Net, U-Net++, Mobile-U-Net, and Efficient-U-Net were implemented and evaluated with a validation set. U-Net structure from scratch with specific adjustment dominated other networks after several initial trials. With further experiments of hyperparameter tuning, this architecture finally achieved a training loss of 2.91 and the best public score of 0.70925 (with 20% test data).

Introduction

Context

Neural network models have been proved very successful in image classification tasks. As a further task in computer vision, image semantic segmentation takes image classification to a new level by trying not only to recognize the objects present in an image but also to identify the exact boundary of all objects, in other words, image classification at the pixel level. The image segmentation techniques are widely used in areas such as medical image analysis, autonomous vehicles, and augmented reality. In the context of this Kaggle competition, our task is to segment images used to assess the damages after Hurricane Harvey in Houston, Texas. There are in total 25 target segments, including roof, road, and vehicles.

Related work

In the area of semantic segmentation, a major amount of work has been done on DL-based image segmentation models. Long et.al[8] proposed a Fully Convolutional Network that allows for semantic segmentation of images, only containing convolution layers. Due to the structure of the FCN, encoder-decoder based models were further enhanced to produce segmentation maps, with SegNet[7], fully convolutional encoder-decoder architecture, with the novelty that the decoder upsampling the images in a nonlinear fashion. Another class of model that is highly popular for medical image segmentation is the U-Net[10]. The popularity of U-Net furthered the research in image segmentation for pyramid based network models that could extract key points in the images through localization. Further architectures that were built used Residual networks in order to capture patterns within the convolution layers. Lastly, one of the most popular methods to perform image segmentation is the Faster R-CNN network[11], developed by Facebook. The Faster R-CNN network uses a Region Proposal Network that proposes bounding box candidates, and then the network further detects the regions of interest to appropriately capture semantic segments of the different objects in the image.

Methodology

Model

U-Net

For the proposed architecture upon which we performed our Kaggle task, we implemented the U-Net. The U-Net is easy to use and build due to it sharing a similar network structure with classical encoder-decoder models, with some new additions of skip-connections and asymmetric up-convolutions.

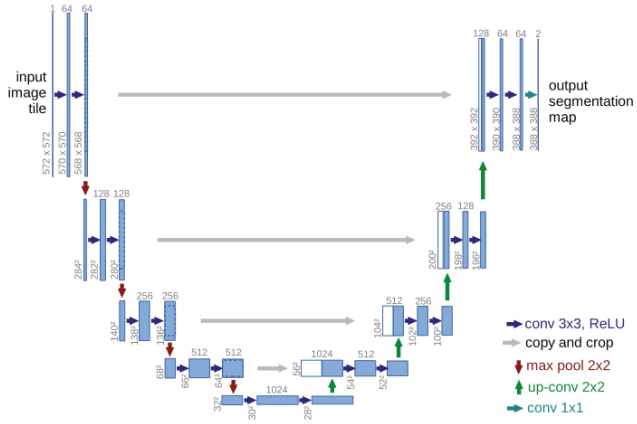

Fig 1. Structure of a typical U-Net

As the figure above shows, the U-Net architecture resembles a U-shape structure, with the network consisting of a downsampling layer and an upsampling layer. Both the layers have equal amounts of convolution and pooling layers, allowing the network to symmetrically capture the features within the image. The essential part of the network comes from its skip connections, which allows the network to propagate context information to higher resolution layers, making the decoder part symmetric to the encoder part. The benefit of having skip connections is that it allows the preserving of gradients during the backpropagation flow and alleviates the problem of vanishing gradients in deep neural networks.

Using U-Net was our first initial choice due to its simplicity and ease of training, which allowed us to better optimize and debug the network during fault occurrences.

Other models

Besides U-Net, we have tried several other models. Although we finally decided to proceed with U-Net and abandon them given their less performant results, we summarized these models here.

Attention U-Net

Another architecture we tried to implement in order to boost our prediction accuracy is the Attention U-Net[9]. The idea of Attention U-Net is similar to that of the classic U-Net, with the addition of an attention mechanism that allows localizing pattern recognition in places where the network believes to

find significant patterns. By using the Ψ-layers, the attention layers capture precise locations of the pixels that aid in the segmentation of images. Our motivation for using this architecture was to classify minority points that were more prevalent around the borders and edges of the images, aiding in boosting the class accuracy.

MobileNet

After building the U-Net model from scratch, we would like to experiment with its variants by the virtue of smp[4], a Python library with Neural Networks for Image Segmentation based on PyTorch.

Firstly, we tried with MobileNet-V2. Unlike the general trend in computer vision aiming for deeper and more complicated networks, MobileNets are featured for its light weight and consequently lower latency, which is crucial in some real world applications like autonomous driving, the recognition tasks need to be carried out in a timely fashion on a computationally limited platform.

For the goal of light computation, depthwise separable convolutions constitute the core layers of MobileNet, which is a form of factorized convolutional which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution [6].

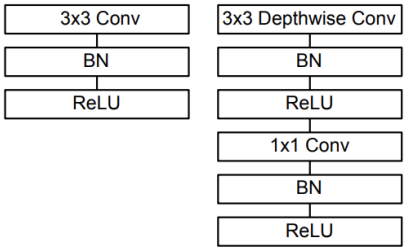

Fig. 2 A comparison between standard convolutional layer with batchnorm and RELU and a depthwise separable convolution.

By specifying the encoder_name= 'mobilenet_v2' and encoder_weights='imagenet', we defined our model and trained it with 15 epochs.

EfficientNet

Another architecture we tried is U-Net with pre-trained EfficientNet as the encoder. This model won 1st place in the Severstal: Steel Defect Detection Kaggle Competition[1] in 2019. EfficientNet[5] was proposed by Google Research in 2019 to efficiently scale up Convolutional Neural Networks. dimensions of depth/width/resolution using a compound coefficient. While the process of scaling up remains quite arbitrary, the EfficientNet balances the dimension of width, depth, and resolution with a set of constant scaling coefficients. The intuition behind the idea of EfficientNet is that larger input images will need larger receptive fields and more channels to capture more detailed patterns. We initially experimented with EfficientNet3.0, which was used in the 1st place solution mentioned above. Despite a training loss of 2.50, this model only achieved a score of 0.63415 on the test set, which implied overfitting. To simplify the network, we then tried EfficientNet3.0, EfficientNet2.0, and EfficientNet1.0, which have fewer channels, and added batch normalization. The best score of EfficientNet was 0.66943 with EfficientNet2.0.

Table 1 Comparison of different models

model	Validation Loss (Cross-Entropy)	Test set accuracy (Kaggle score, Dice efficient)
Attention U-Net	1.48	0.62
MobileNet-V2	2.62	0.68
EfficientNet	2.79	0.67
U-Net Plus Plus	2.10	0.66

Implementation details

Data preprocessing

The input images are 3000 pixels by 4000 pixels, with an average size of 4-5MB. Given the computation constraints, we firstly reduced images to 512*512. To scale all features to the same range and to help convergence, we then normalized the data set by subtracting the mean and dividing by the standard variance. Since the training set has in total 261 items and the testing set has 112 images, if we train solely on the training set, our models are very likely to overfit. To enrich the training samples, we augmented the training images and their masks by transformations such as flipping, random rotation, sharpening, and random brightness control. The implementation was realized with the Albumentations[2] library. It is noteworthy that we opted for online data augmentation. Compared to offline data augmentation, which saves all synthetic images on the disk and thus results in an explosive increase in size, online augmentation performs all transformations on each mini-batch during the training process and never actually saves the generated images. Another advantage of online augmentation is that it can be accelerated by GPU.

Optimization strategy

In the optimization strategy, the Adam algorithm with good performance was used as the optimizer. In addition, a discriminative learning rate was used to make the shallow layer of the model have a smaller learning rate, and the deeper layer of the model has a higher learning rate. This is because the shallow layer of the model often extracts coarser information such as contour information, which determines the shape and position of the target, so a smaller learning rate is required to maintain the stability of the information. In addition, whether dropout plays a role and how much the probability of an element to be zeroed depends on what model is used and the complexity of the model. Complex models and deep layer networks should work together with a high dropout rate, meanwhile low dropout probability or even no dropout should be implemented in simple neural networks(with fewer parameters).

Loss function

As the most popular choice for classification problems, CrossEntropy loss is used here. Plus, one compelling reason for using cross-entropy over dice-coefficient or the similar IoU metric is that the gradients are nicer to compute and conducive to faster convergence.

Model fitting

Except the network structure itself, hyperparameter tuning is also essential for model performance to avoid both underfitting and overfitting by the choice of parameters. The ideal way to tune hyperparameters is actually grid search and tune all parameters together to find the optimal parameter set, but due to the long network training time, it is not realistic to implement grid search to network with our local machine. Therefore, parameters are tuned by **background research, previous experience and some necessary experiments**.

The choice of the number of epochs and learning rate should be decided together, in general, we set a high learning rate with a low number of epochs or vice versa. Based on experience, the learning rate is always set around 0.001 while an adapted learning rate method can be implemented as an improvement of the constant learning rate, and decay weight is set as 1e-4. The plot below shows the training loss and validation loss with different epochs.

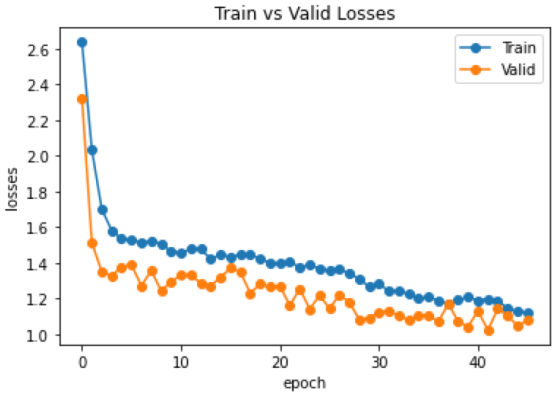

Fig. 3 Learning curve with different epochs.

We can see from above that the training loss is decreasing all the time while the validation loss becomes oscillating after 20 epochs. **We drew a conclusion that the optimal number of epochs should be around 20-35.** We experimented with several epoch choices between 20 and 35, and it turns out that epoch 20 gave us the best performance given on the learning rate decided before.

Experiments

Our trials for improvement

1. **Reduce the output channels of convolution layers**
   Based on simple U-Net, we adjusted the output channels of each convolution layer to align with our specific problem. This adjustment surely improved model accuracy quite a lot by avoiding overfitting.
2. **Handling class imbalance**
   Among the 25 segments, we noticed a significant imbalance. ‘Grass’, ‘Trees / Shrubs’ and ‘Flooded’ labels take up more than 50%. To take care of this imbalance, we specified the `weight` argument in `nn.CrossEntropyLoss()` by applying 1/log(percentage of occurrences).

Basically, for classes with a small number of training images,we give it more weight so that the network will be punished more if it makes mistakes predicting the label of these classes. For classes with large numbers of images, you give it a small weight. However, this step didn't really help with the improvement.

3. **Applying adaptive learning rate**
   This was important to ensure that the learning rate changes as we progress further in the training scheme, reducing the learning rate if the validation accuracy keeps improving over epochs.
4. **Applying softmax layer to the output**
   We noticed that most architectures would only produce the segmentation map from the final convolution layer, but the output probabilities were not given as the networks were following the proposed architecture of their paper. Hence we added a softmax function in the final layer and it significantly improved our performance.
5. **Reducing the image size by half:**
   Our initial image size post-processing was 512x512, but we further reduced it to 256x256 as U-Net architecture tends to perform better for smaller images, and the size reduction improved the validation score and the leaderboard position.
6. **Implemented U-Net++**
   We further tried U-Net++, a sister network of U-Net with deeper deconvolution layers. The idea here was that perhaps having deeper encoder-decoder layers would improve the performance of the segmentation.

The optimal model

Since the divergence between validation loss and public Kaggle score observed during previous experiments, we assumed the model is overfitted, thus we tried several strategies to avoid overfitting. After several experiments above, our model performance has improved from around 0.60 to more than 0.70 in the leaderboard.

1. **Network**
   The optimal model is built on the U-Net from scratch with some changes. For double convolution layers, we keep the U-Net structure with 5 double convolution layers, meanwhile cutting the output channel of the first double convolution from 64 to 16. Furthermore, the batch normalization layer is applied after every convolution layer in the encoder stage.

   The reason why we still need batch normalization with normalized input is that the weights of the network change at each iteration, this would be extremely costly. That is why batch normalization tries to estimate it from the batch at hand and using some historical information sometimes. Furthermore, the dropout layers are also implemented after every max pooling layer with a quite small rate 0.1. And a softmax normalization layer is implemented in the output layer for multi-classification problems.

2. **Hyper-parameters**
   In the end, we implemented 25 epochs with learning rate 0.001, and the batch size is set as 15.

The original remote sensing picture and corresponding predicted mask by best performance model are shown below.

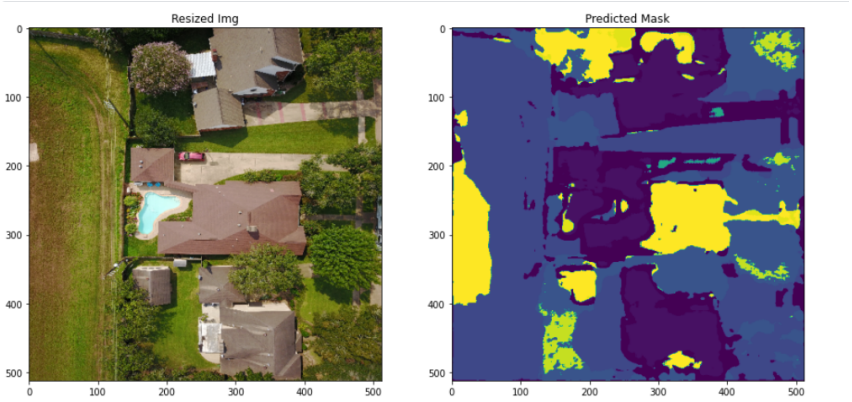

Fig. 5 Predicted mask

Further steps

As for potential further steps, ensemble deep learning can be an interesting direction. Overfitting was one of the major challenges we encountered during the competition, hence ensemble strategies combining several models together to achieve better generalization performance are worth further investigation. In [3], the authors designed a Fully Convolutional Network (FCN) to perform semantic segmentation on high-resolution aerial images of urban areas and showed that an ensemble of several networks achieved excellent results using only the raw data as input.

Bibliography

[1]
R. Guo, “1st Place Solution | Severstal: Steel Defect Detection,” *kaggle.com*. https://www.kaggle.com/c/severstal-steel-defect-detection/discussion/114254#latest-675874.
[2]
A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, “Albumentations: Fast and Flexible Image Augmentations,” *Information*, vol. 11, no. 2, p. 125, Feb. 2020, doi: 10.3390/info11020125.
[3]
D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla, “SEMANTIC SEGMENTATION OF AERIAL IMAGES WITH AN ENSEMBLE OF CNNS,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III–3, pp. 473–480, Jun. 2016, doi: 10.5194/isprs-annals-iii-3-473-2016.
[4]
P. Iakubovskii, “Segmentation Models Pytorch,” *GitHub*, 2020. https://github.com/qubvel/segmentation_models.pytorch (accessed Jan. 17, 2022).
[5]
M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *arXiv.org*, 2019. https://arxiv.org/abs/1905.11946.
[6]
A. G. Howard, M. Zhu, B. Chen, and D. Kalenichenko, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, [Online]. Available: https://arxiv.org/abs/1704.04861v1.
[7]
V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017, doi: 10.1109/tpami.2016.2644615.
[8]
E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 640–651, 2017, doi: 10.1109/TPAMI.2016.2572683.
[9]
O. Oktay *et al.*, “Attention U-Net: Learning Where to Look for the Pancreas,” *arXiv.org*, 2018. https://arxiv.org/abs/1804.03999.
[10]
O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science*, pp. 234–241, 2015, doi: 10.1007/978-3-319-24574-4_28.
[11]
S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, doi: 10.1109/tpami.2016.2577031.