Semantic Segmentation
CSE 253 Deep Learning PA 3 Part II

Shujie Chen
A53303042
shchen@ucsd.edu

Felix Gabler
A53329004
fgabler@ucsd.edu

Yongchuang Huang
A53322973
yoh001@ucsd.edu

Jingpei Lu
A91033661
jil360@ucsd.edu

Yiran Xu
A53270264
y5xu@ucsd.edu

February 17, 2020

Abstract

In this assignment, we examined different approaches to semantic segmentation using deep Convolutional Neural Networks (CNN) architectures on the Cityscapes dataset. We consider semantic segmentation as a multinomial regression problem with respect to pixels. As a baseline model, we built a encoder-decoder CNN model which uses fully-convolutional network (FCN) architecture. In addition, we experimented with data augmentation and other architectures e.g. transfer learning with ResNet and U-net. Our results vary greatly with each approach tried but data augmentation and U-net were the greatest improvements over the baseline model, which achieves average IoU 0.2589 and 0.2631, respectively.

1 Introduction

The success of deep learning techniques in various high-level computer vision tasks such as image classification and object detection, motivated researchers to explore the capabilities of such networks for pixel-level labelling problems like semantic segmentation. Semantic segmentation is a task that is becoming increasingly more important in the realms of machine learning. Currently, one of its most popular applications is autonomous driving. In contrast to object detection, where we aim to locate and classify objects in images, the goal of semantic segmentation is to classify each and every pixel of the input.

Since CNN became popular, many researchers have started their explorations of using convolutional layers to deal with semantic segmentation. An early approach is FCN [1], which discards fully-connected layers and uses convolutional layers all the way. FCN also uses an encoder-decoder structure with multiple scales. The benefits of using fully convolutional layers is that FCN reduces the parameters needed to learn dramatically, and we can also obtain the output with the same image resolution as the input. Another popular encoder-decoder-based approach is U-Net [2]. This network not only upsamples to recover the original resolution, but also takes an advantages of using previous feature map with the same size.

A common dataset for semantic segmentation is CityScapes [3]. It includes 2,975 training images, 500 validation images and 1525 images test images. It involves 34 common classes in the urban driving scenarios. However, the dataset has an imbalanced data distribution. In this report, we also experimented with Dice Loss to mitigate the imbalanced data.

Also, Xavier initialization and Batch Normalization are considerable techniques for accelerating the training process. The details will be talked in Section 3.

2 Related Work

Currently, most of the successful state-of-the-art deep learning techniques for semantic segmentation stems from a common forerunner: the Fully Convolutional Network (FCN) by Long et al.[4]. Despite the power and flexibility of the FCN model, it still lacks various features which hinder its application to certain problems and situations: its inherent spatial invariance does not take into account useful global context information[5]. In order to overcome this problem, various encoding and decoding methods has been explored, including replacing the encoder with ResNet [6] or increasing the number of feature channels on the decoding process[2].

2.1 Fully Convolutional Network

The main idea of the FCN was to take advantage of existing CNNs as powerful visual models that are able to learn hierarchies of features. FCN transformed those existing and well-known classification models into fully convolutional ones by replacing the fully connected layers with convolutional layers to output spatial maps instead of classification scores. Those maps are upsampled using deconvolution operations to produce dense per-pixel labeled outputs. This work is considered a milestone since it showed how CNNs can be trained end-to-end for this problem, efficiently learning how to make dense predictions for semantic segmentation with inputs of arbitrary sizes and it achieves state-of-the-art segmentation accuracy on PASCAL VOC, NYUDv2, and SIFT Flow datasets.

2.2 ResNet

In order to combat the vanishing gradient problem in very deep neural networks, ResNet [6] employs shortcut connections that skip one or more layers. Therefore, the layers learn a residual mapping instead of the harder desired underlaying direct mapping.

2.3 U-Net

U-Net is a convolutional neural network proposed by Ronneberger et al. that was primary used for biomedical image segmentation[2]. It's an improvement and development of FCN. The main idea is to supplement a usual contracting network by successive layers, where pooling operations are replaced by upsampling or deconvolution operators. Using the modified architecture, U-Net achieves better accuracy on multiple image segmentation tasks.

3 Methods

3.1 General

We used Xavier weights initialization to start the training. Xavier initialization sets a layer's weights to values drawn from a uniform distribution $\mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}})$, where n_{in} is "fan-in" and n_{out} is "fan-out".

Between each activation layer and convolutional layer, Batch Normalization [7] is applied. In short, batch normalization will normalize the batch into a zero mean, unit standard deviation distribution as follows:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \,, \tag{1}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_B - \mu_B)^2 \,, \tag{2}$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \,, \tag{3}$$

$$y_i = \gamma\hat{x_i} + \beta \,, \tag{4}$$

where x_i is the data in the batch, m is the batch size, γ and β are learnable parameters that can recover the data.

For evaluation, we tested our model using pixel accuracy and average Intersection over Union (mIoU).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \,, \tag{5}$$

$$IoU = \frac{TP}{TP + FP + FN} \,, \tag{6}$$

where TP, TN, FP, FN stand for True Positive, True Negative, False Positive and False Negative, respectively. mIoU computes the mean value over interested classes.

3.2 Baseline Model

The baseline model is a fully convolutional network and using Cross-Entropy Loss function as the objective function. The detailed architecture is described in Table 1. Given the limited number of GPUs, in order to improve training speed at the cost of lossing some samples and lowering some accuracy, we resized images in training set from the size of 1024×2048 to 512×1024. The batch size is 4 and The training epochs are 35.

layer name	layer setting
conv1	$[3 \times 3, 32]$, pad=1 Batch Norm + ReLU
conv2	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
conv3	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
conv4	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
conv5	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
deconv6	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
deconv7	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
deconv8	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
deconv9	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
deconv10	$[3 \times 3, 32]$, pad=1 Batch Norm + ReLU
conv_final	$[1 \times 1, n\_classes]$

Table 1: Detailed architecture of Baseline Model

3.3 Data Augmentation

To generalize the model, we usually use image transforms to augment the dataset. In this project, random rotations for 10 degrees and horizontal flips are used with a probability $p = 0.2$. The model was trained with the same batch size and 30 epochs as the baseline model.

3.4 Adapted Model

To try one additional architecture, we added a sixth additional convolutional layer and changed all the activation functions in the convolutional layers to Parametric Rectified Linear Unit (PReLU). We also resized images in training set from the size of 1024×2048 to 512×1024 and trained with the batch size of 4 and 32 epochs.

3.5 Dice Coefficient Loss

To address the rare class or imbalanced class problem and pressure the network to categorize the infrequently seen classes, on the basis of Baseline Model, we changed Cross-Entropy Loss to Dice Coefficient Loss. The model was trained with the same batch size as the Baseline Model and 37 epochs . The Dice Coefficient Loss is

$$DCL = 1 - \frac{2 \times TP}{2 \times TP + FP + FN} \tag{7}$$

where TP, TN, FP, FN stand for True Positive, True Negative, False Positive and False Negative, respectively, in the predicted result compared with the ground truth.

3.6 Transfer Learning

As an additional architecture, we experimented with transfer learning by replacing the encoder part of the baseline model with a pretrained ResNet50 [6] model. For this, we removed the last fully-connected layer of the ResNet50 architecture and fed its output directly into our decoder. The decoder had to be slightly adapted to take larger inputs, as the output dimensions of the last convolutional layer of ResNet50 are larger. It is also important to note that for the main tests, we chose to freeze the weights in order to see if relying on good encoding results by ResNet50 and just learning the decoder weights would yield a good accuracy. Table 2 shows the architecture of the ResNet version used in our tests. Each convolutional block was also followed by ReLU to introduce nonlinearity.

layer name	layer dimensions
conv1	kernel-size 7×7, 64 out-channels, stride 2
conv2_x	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
avgpool	output-size 1×1

Table 2: Detailed architecture of ResNet50

3.7 U-Net

The implementation of the U-Net is adopted from the original paper [2]. The detailed architecture is described in Table 3. The layer dimension is described in the format $[kernel\_size, output\_channel]$. The padding and stride is not specified if it is default setting, and cat is the concatenation of the output feature maps.

layer name	layer setting
conv0_1	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
conv0_2	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
maxpool0_1	2×2, stride = 2
conv1_1	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
conv1_2	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
maxpool1_2	$[2 \times 2]$, stride = 2
conv2_1	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
conv2_2	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
maxpool2_3	$[2 \times 2]$, stride = 2
conv3_1	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
conv3_2	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
maxpool3_4	$[2 \times 2]$, stride = 2
conv4_1	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
conv4_2	$[3 \times 3, 512]$, pad=1 Batch Norm + ReLU
deconv4_3	$[2 \times 2, 512]$, stride=2 cat[conv3_2, deconv4_3]
conv5_1	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
conv5_2	$[3 \times 3, 256]$, pad=1 Batch Norm + ReLU
deconv5_2	$[2 \times 2, 256]$, stride=2 cat[conv2_2, conv5_2]
conv6_1	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
conv6_2	$[3 \times 3, 128]$, pad=1 Batch Norm + ReLU
deconv6_1	$[2 \times 2, 128]$, stride=2 cat[conv1_2, conv6_2]
conv7_1	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
conv7_2	$[3 \times 3, 64]$, pad=1 Batch Norm + ReLU
deconv7_0	$[2 \times 2, 64]$, stride=2 cat[conv0_2, conv7_2]
conv8_1	$[3 \times 3, 16]$, pad=1 Batch Norm + ReLU
conv8_2	$[3 \times 3, 16]$, pad=1 Batch Norm + ReLU
conv_final	$[1 \times 1, n\_classes]$

Table 3: Detailed architecture of U-Net

4 Results

We report pixel accuracy and IoU for each model. We only consider the classes whose $trainId \neq 255$. In particular, we report the class IoU for building, traffic sign, person, car and bicycle. The results are shown in Table 4.

Models	Pixel accuracy	mIoU	building	traffic sign	person	car	bicycle
Baseline	0.7527	0.2210	0.5819	0.1367	0.1514	0.4513	0.0785
Data Augmentation (resized)	0.7728	0.2460	0.6117	0.2180	0.1826	0.4985	0.1132
Adapted Model	0.7661	0.2243	0.6064	0.1062	0.1465	0.5278	0.0765
Dice Coefficient Loss	0.7401	0.2155	0.5757	0.2620	0.1859	0.4252	0
Transfer Learning	0.5380	0.0927	0.2566	0.0025	1.3824	0.1157	0
U-Net	0.7913	0.2631	0.6334	0.3192	0.2082	0.5267	0.1328
Data Augmentation (not resized)	0.8381	0.2589	0.6539	0.2315	0.1877	0.5300	0.1487

Table 4: Quantitative results on validation set: Pixel accuracy, average IoU (mIoU) and IoUs of specific classes.

4.1 Baseline Model

The batch size is 4 with 35 epochs and the initial learning rate is 0.005. We have explored the Baseline Model with resizing (from 1024×2048 to 512×1024).

The training loss and validation loss are shown in Figure 1. The quantitative result is listed in Table 4. The quantitative result on the first image of the testset in shown in Figure 2.

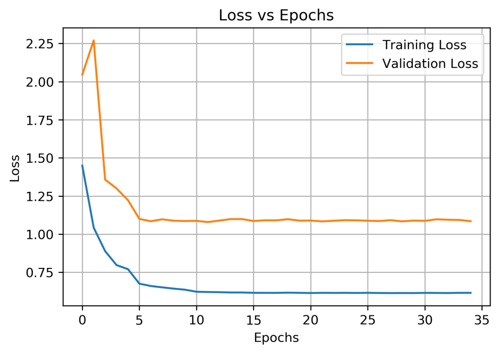


Figure 1: Training loss and validation loss with Baseline Model.

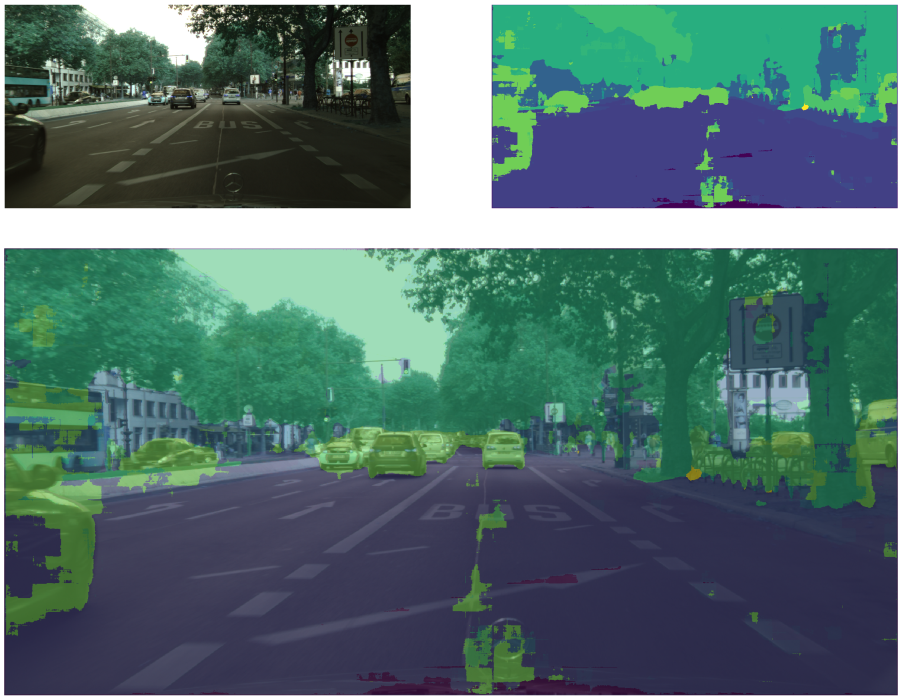
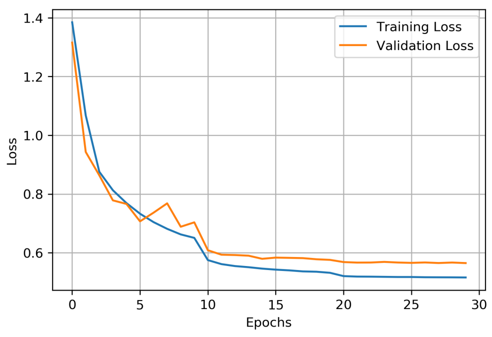


Figure 2: Qualitative result after training Baseline Model.

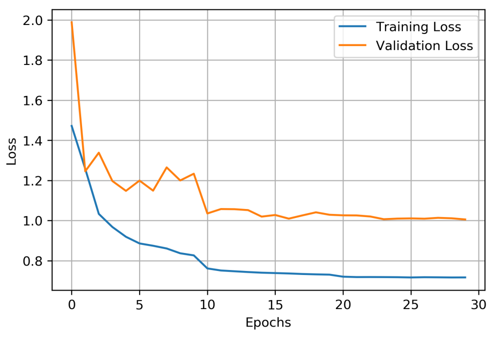
4.2 Data Augmentation

To augment the dataset, we implemented random horizontal flipping and random rotation. We randomly horizontally flipped and rotated the image for 10 degrees with a probability $p = 0.2$. The batch size is 2 with 30 epochs. The initial learning rate is 0.0001, and it will be divided by 10 in every 10 epochs. To avoid overfitting, we used ℓ_2-regularization with weight decay 0.0001. We also have explored the data augmentation with resizing (from 1024×2048 to 512×1024) and without resizing.

The training loss and validation loss are shown in Figure 3. The quantitative result is listed in Table 4. The qualitative result on the 1-st image of the testset is shown in Figure 4. We can see that data augmentation without resizing achieves better result than the baseline. Also, from Figure 3, the overfitting is mitigated compared to the baseline.



(a) Not resized



(b) Resized

Figure 3: Training loss and validation loss with data augmentation.

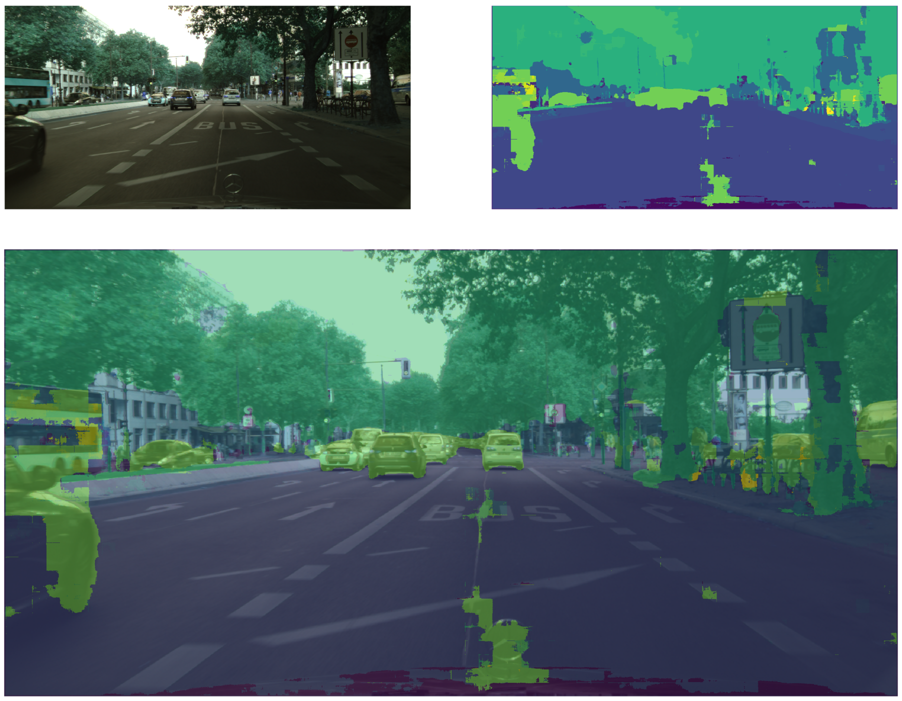


Figure 4: Qualitative result using data augmentation.

4.3 Adapted Model

We trained our adjusted model using a batch size of 4 for 32 epochs. Figure 5 shows the loss on training and validation sets. Figure 6 visualizes the segmented output of the trained model for the first image in the test set.

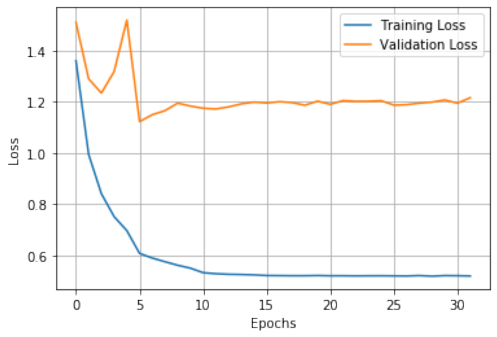


Figure 5: Training loss and validation loss with adapted model.

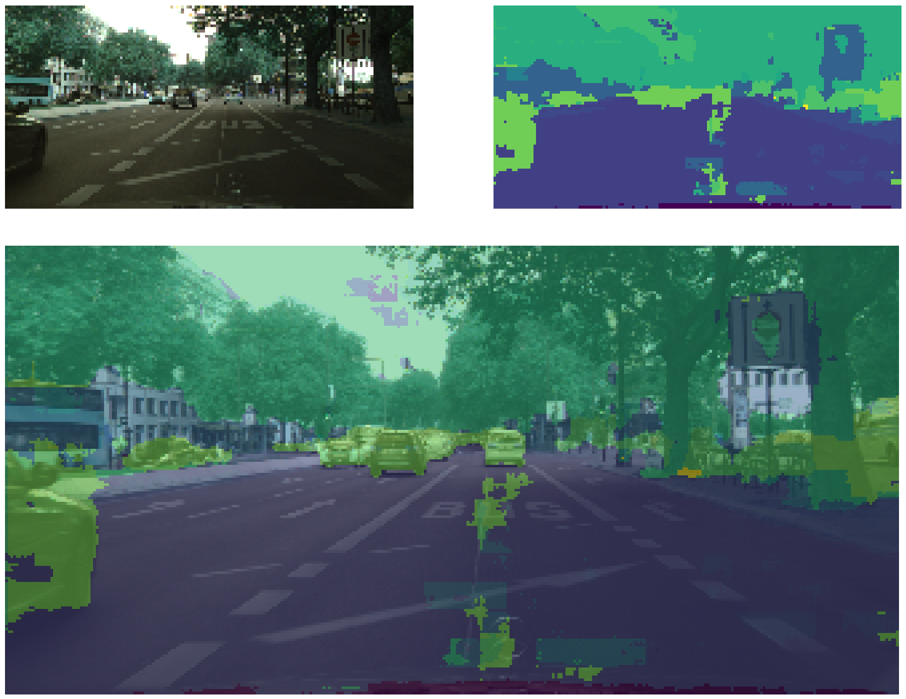


Figure 6: Qualitative result from adapted model.

4.4 Dice Coefficient Loss

We trained our adjusted model using a batch size of 4 for 37 epochs. Figure 7 shows the loss on training and validation sets. Figure 8 visualizes the segmented output of the trained model for the first image in the test set.

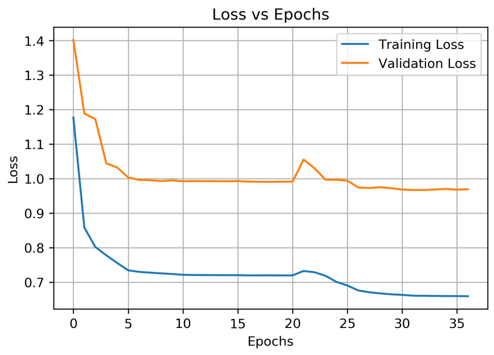


Figure 7: Training loss and validation loss using dice coefficient loss.

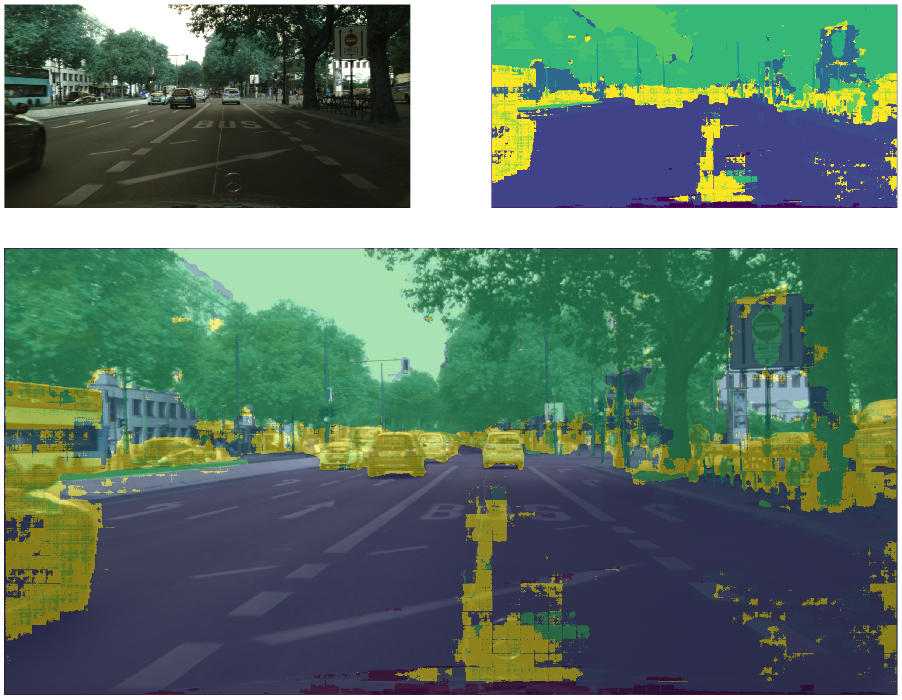


Figure 8: Qualitative result of Baseline Model with dice coefficient loss.

4.5 Transfer Learning

We trained the transfer learning model using a batch size of 2 for 30 epochs. Figure 9 shows the loss on training and validation sets. Figure 10 visualizes the segmented output of the trained model for the first image in the test set.

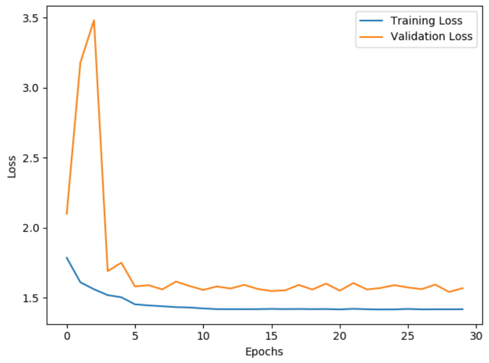


Figure 9: Training loss and validation loss with transfer learning (ResNet50).

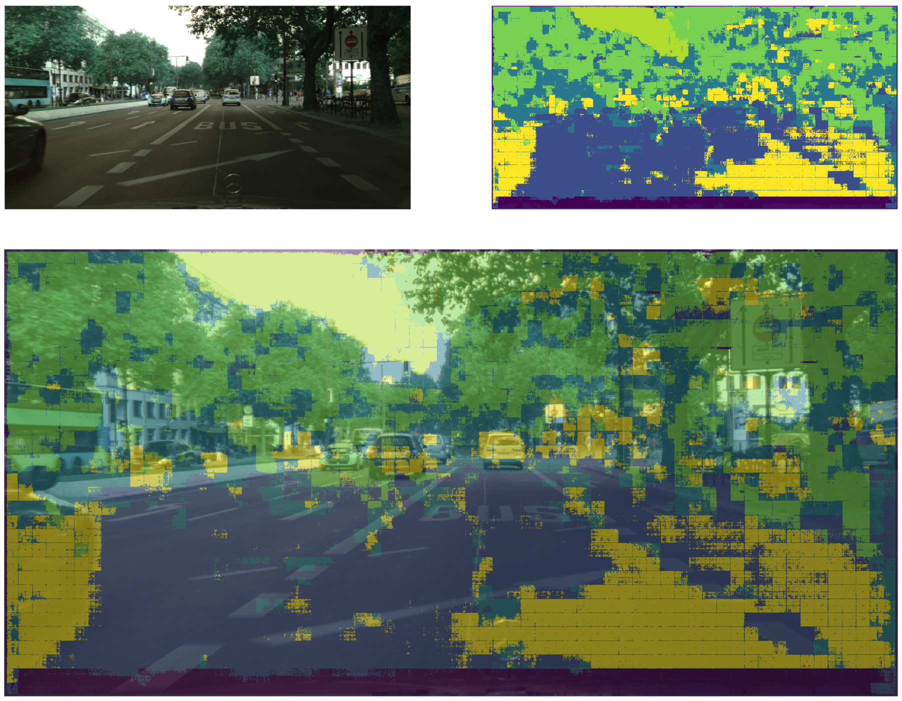


Figure 10: Qualitative result using transfer learning.

4.6 U-Net

The learning rate is initialized as 0.0001 and it will be be divided by 10 in every 10 epochs. We trained U-Net for 30 epochs with the batch size of 2. Training loss and validation loss are shown in Figure 11. The quantitative result is listed in Table 4 and qualitative result is shown in Figure 12.

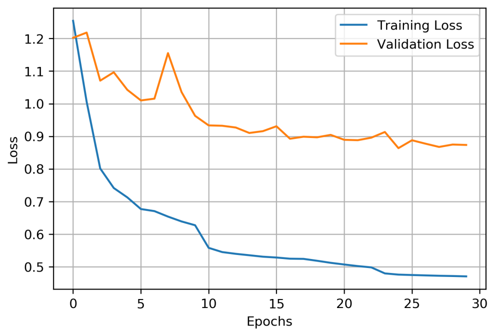


Figure 11: Training loss and validation loss using U-Net.

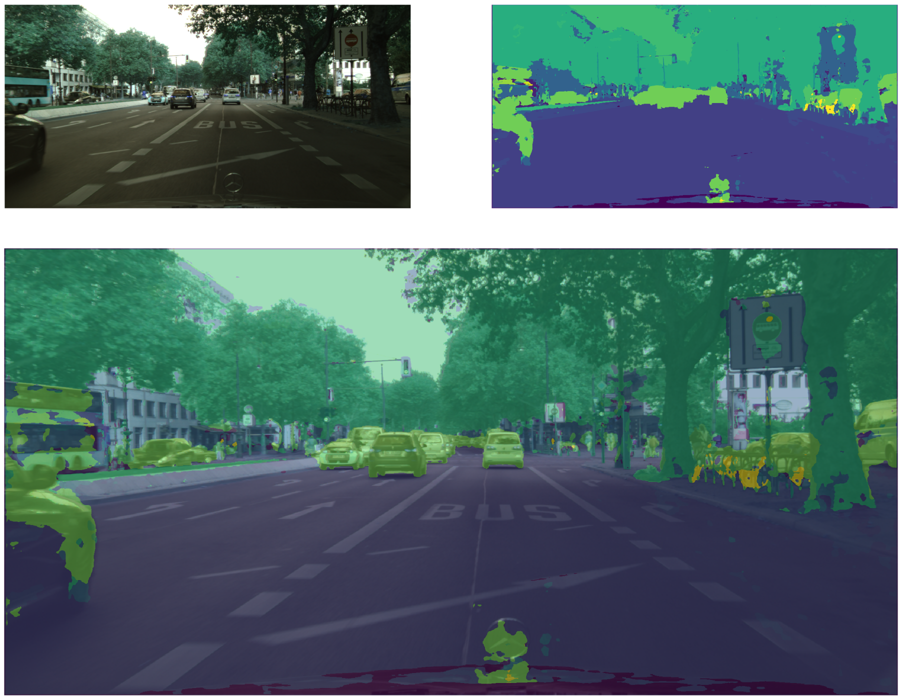


Figure 12: Qualitative result using U-Net.

5 Discussion

5.1 Baseline Model

Baseline achieves mIoU of 0.2210 and pixel accuracy of 0.7527. However, the baseline does not perform well for each class. For instance, it only achieves 0.0785 IoU on bicycle, since bicycle has a low fraction in the dataset (See Figure 13). Also, the baseline does not consider reuse information from previous layers.

5.2 Data Augmentation

The purpose of data augmentation is to enlarge the dataset. From Table 4, we can see that baseline + data augmentation outperforms the original baseline result. Furthermore, using data augmentation without any resizing can achieve higher accuracy and mIoU. This is probably because the model generalizes better given the same sizes of training data and validation data. Intuitively, for images without resizing, they are 4 times bigger than the resized images, which means provided more data points (pixels) during the training process. However, due to our limited conditions and time, we choose to use resized images.

5.3 Adapted Model

The ReLU activation function has been very popular since the last fews years ago but it has a problem of dying ReLU and then Leaky ReLU was introduced to solve this problem by putting a small negative slope for the negative values. Different from Leaky ReLU, PReLU allows the coefficient of the leakage to be a parameter that can be learned along with other neural network parameters. Therefore, we expected the adapted model to be better than the baseline model. Compared to baseline model, the pixel accuracy increased roughly 1.4%, the mean intersection over union increased 0.0033, which met our expectation.

5.4 Dice Coefficient Loss

Dice Coefficient Loss is used to address the rare class or imbalanced class problem. Through counting the number of each class's pixels from the training set, we found that among the 34 classes the class with the minimal pixels in the whole training set is $poleground$. We also found that among those whose $trainId \neq 255$ the rarest class is $motorcycle$. The result is shown in figure 13.

After changing Cross-Entropy Loss to Dice Coefficient Loss on Baseline Model, both pixel-accuracy and IoU decreased as shown in table 4. However, the IoUs of $trafficsign$ and $person$ obviously increased, which are the relative rare classes among those whose $trainId \neq 255$. The reason might be the majority classes are penalized more than they should be, which have more influence on the final pixel accuracy and IoU , although to some extend, using Dice Coefficient Loss improved the results of some rare classes.

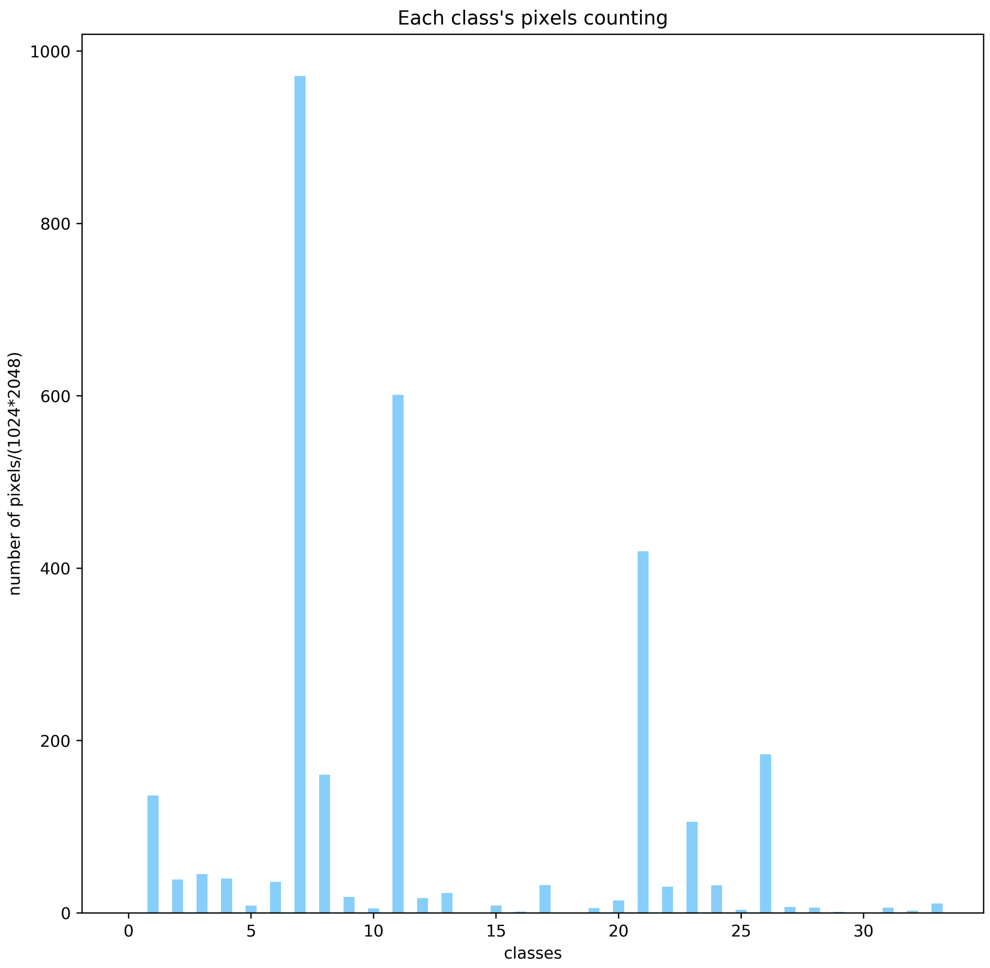


Figure 13: Each class's pixels counting. The ordinate represents the total number of pixels of each class on the whole training set, scaled by$1/(1024 \times 2048)$.

5.5 Transfer Learning

The benefit of this approach is that parts of the network are pretrained. Hence, they do not necessarily require retraining. However, as seen in our experiments, the performance at the end was very suboptimal when freezing the ResNet weights. It seems to react much more sensitive to certain objects (e.g. person vs. traffic sign) than the baseline model. This might be due to how it was pretrained. Overall, there is just so much the decoder is able to improve with the encoder being fixed. Therefore, both the training and the validation loss converge quickly to a much higher loss than the baseline model.

5.6 U-Net

We found that U-Net is about to achieve better segmentation accuracy compared to the baseline FCN model. U-Net uses a large number of feature channels in the upsampling part, which allow the network to propagate context information to higher resolution layers[2]. Also, by concatenating the low-level feature map to high-level feature map, the network is able to combine the information from the previous layers in order to get a more precise prediction.

6 Contribution

Shujie Chen Tried and assessed additional architectures

Felix Gabler Implemented and evaluated transfer learning architecture

Yongchuang Huang Completed the baseline model and experimented with solutions to the imbalanced class problem

Jingpei Lu Completed the U-Net.

Yiran Xu Implemented the metrics and transformations for data augmentation.

References

[1] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (2015), pp. 234–241. ISSN: 1611-3349.

[3] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[4] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (Apr. 2017), pp. 640–651. ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2572683. URL: http://dx.doi.org/10.1109/TPAMI.2016.2572683.

[5] Alberto Garcia-Garcia et al. *A Review on Deep Learning Techniques Applied to Semantic Segmentation*. 2017. arXiv: 1704.06857 [cs.CV].

[6] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv e-prints*, arXiv:1512.03385 (Dec. 2015), arXiv:1512.03385. arXiv: 1512.03385 [cs.CV].

[7] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).