CSE 251B PA3: Convolutional Neural Networks for Semantic Segmentation

Yun-Yi Lin
Computer Science and Engineering
UC San Diego
La Jolla, CA 92093
yul059@ucsd.edu

Zhuojun Chen
Computer Science and Engineering
UC San Diego
La Jolla, CA 92093
zhc111@ucsd.edu

Peiyi Li
Computer Science and Engineering
UC San Diego
La Jolla, CA 92093
pel049@ucsd.edu

Zi-Xiang Xia
Computer Science and Engineering
UC San Diego
La Jolla, CA 92093
zixia@ucsd.edu

Abstract

In this assignment, we use deep convolutional neural networks to experiment semantic segmentation task on India Driving Dataset. We use pixel accuracy and Intersection-Over-Union as the evaluation metrics and experiment different kinds of methods, including weighted loss, data augmentation, transfer learning, and U-Net architecture. Weighted loss method achieved 61.07% average IoU and data augmentation method with rotated and cropped images achieved 63.73% IoU. However, it largely increases the training time. Using the transfer learning VGG, we achieve 73.74% average pixel accuracy and 58.45% average IoU on the test set. In addition, we achieve 78.03% average test pixel accuracy and 64.04% average test IoU by the method of U-Net. Finally, we build a new model named Mobile-Deeplabv3+, and we achieved 83.56% average pixel accuracy and 71.35% average IoU on the test set.

1 Introduction

Image segmentation is a computer vision task in which we label different regions of image according to the class they belong to. Nowadays, it is important for the autonomous vehicle to understand the environment so the self-driving cars can safely running on the road. For normal deep convolution networks for the task of image classification, the task only cares about what the image contains. Therefore, convolution networks for image classification usually increase the number of feature maps to get more high-level features. To reduce the computation costs, we can periodically downsample our feature maps by pooling and striding. However, for image segmentation, we want to get the full resolution image instead. Hence, a popular method for image segmentation is to use an encoder followed by a decoder. First, we downsample the image to get lower-resolution feature maps. The model learns high-level features and distinguishes the difference between classes by lower-resolution feature maps, and then unsample the feature maps to get the full-resolution segmentation map. However, since encoder downsamples the resolution of input by a large number, the decoder will struggle to get well segmentation. To address the problem, researchers add "skip connections" from earlier layers. The skip connections could provide the details for the decoder to construct more accurate shapes for segmentation. To achieve better accuracy, we combine the

idea of a stronger encoder and learning multi-scale features. Therefore, we come up with a model named Mobile-Deeplabv3+, which use MobileNet as the backbone with astrous convolution to learn the different level of information. MobileNet can effectively extract visual features within bearable training time, and the usage of astrous convolution with decoder helps the model learn from both low and high level features. Therefore, Mobile-Deeplabv3+ has the best results among the all the techniques of we experimented: a basic deep CNN, an improved version, VGG and U-Net. More details about those methods are discussed in the next section.

2 Related Work

2.1 DeepLabv3+

Deeplabv3+[5] is an encoder-decoder structure which employs DeepLabv3[6] as its encoder module and a simple but effective decoder module. The rich semantic information is encoded in the output of DeepLabv3, which applies Atrous Spatial Pyramid Pooling (ASPP), i.e., several parallel atrous convolution with different rates. They also use the Xception model[7] for the segmentation task and apply depthwise separable convolution to both ASPP module and decoder module. The architecture is shown in Figure 1. They achieve state-of-art performance on PASCAL VOC 2012 and Cityscapes datasets.

Here, we briefly introduce the special techniques in this paper: atrous convolution[1] is a powerful convolution which explicitly control the resolution of features and adjust the perception field in order to capture multi-scale information. Atrous convolution is applied over the input feature map with atrous rate r which determines the stride with which we sample the input signal. In addition, depthwise separable convolution is used to reduces computation complexity. Compare to pointwise convolution which combines the output from the depthwise convolution, the depthwise convolution performs a spatial convolution independently for each input channel.

In our work, instead of using a heavy work like Deeplabv3 and xception, we use mobilenet[2] as our backbone to strike the balance between training time and accuracy.

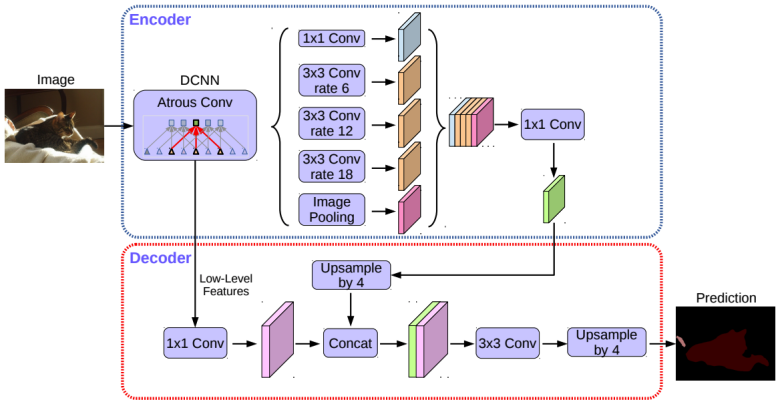


Figure 1: Visualization of the Deeplabv3+ Architecture. The encoder module is composed of a backbone network with atrous convolution. The encoder module is a convolution layer with the input of depthwise convoluted information from encoder and backbone model.

2.2 Transfer Learning - VGG

VGG was introduced in the paper Very Deep Convolutional Network for Large-Scale Image Recognition by K. Simonyan and A. Zisserman from the University of Oxford[3]. It provides a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolutional filters. In this paper, they addressed an important factor of ConvNet

architecture design – the depth. At the end, they achieved the state-of-art accuracy on The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) classification and localization tasks by increasing the depth of the ConvNet.

Figure 2 shows the architecture of the VGG with 16 layers. Note that all the blue layer represents for the convolutional layers with non-linear activation functions which is a ReLU unit. The red layers represent for the max-pooling layers so that there are 18 convolutional layers in total. Along with these, there are 3 fully connected green layers at the end. Therefore, it has 16 learnable parameters of which 13 is for convolutional layers and 3 is for fully connected layers. For each layer, the kernel size is 3×3. In the training procedure, we started with a fixed-size 224×224 image with very low initial channel 64, and then gradually increased by a factor of 2 after max-pooling layer until reaching 512.

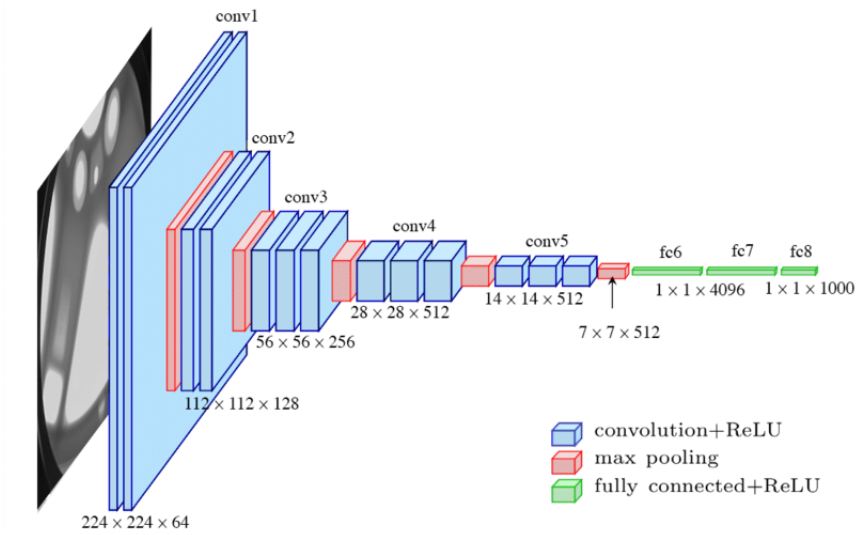


Figure 2: Visualization of the VGG Architecture

2.3 U-Net

U-Net was introduced in the paper U-Net: Convolutional Networks for Biomedical Image Segmentation[4] by Ronneberger et al. It was built to include localization for biomedical images, which have limited number of samples available. U-Net is a fully connected convolutional network consists of a contracting path and an expansive path as in Figure 3. The contracting path first downsizes the input images following the typical architecture of a convolutional network: it consists of the repeated application of two 3×3 convolutions, each followed by a ReLU and a 2×2 max pooling operation with stride 2, and the number of feature channels are doubled at each downsampling step. The expansive path consists of an unsampling of the feature map followed by a 2×2 convolution, a concentration with the correspondingly cropped feature map from the contracting path, and two 3×3 convolutions followed by a ReLU at each step. The final layer uses an 1×1 convolution to map each 64 component feature vector to the desired number of classes. U-Net works with very few training images and yields more precise segmentation, so it's worth experimenting on the India Driving Dataset.

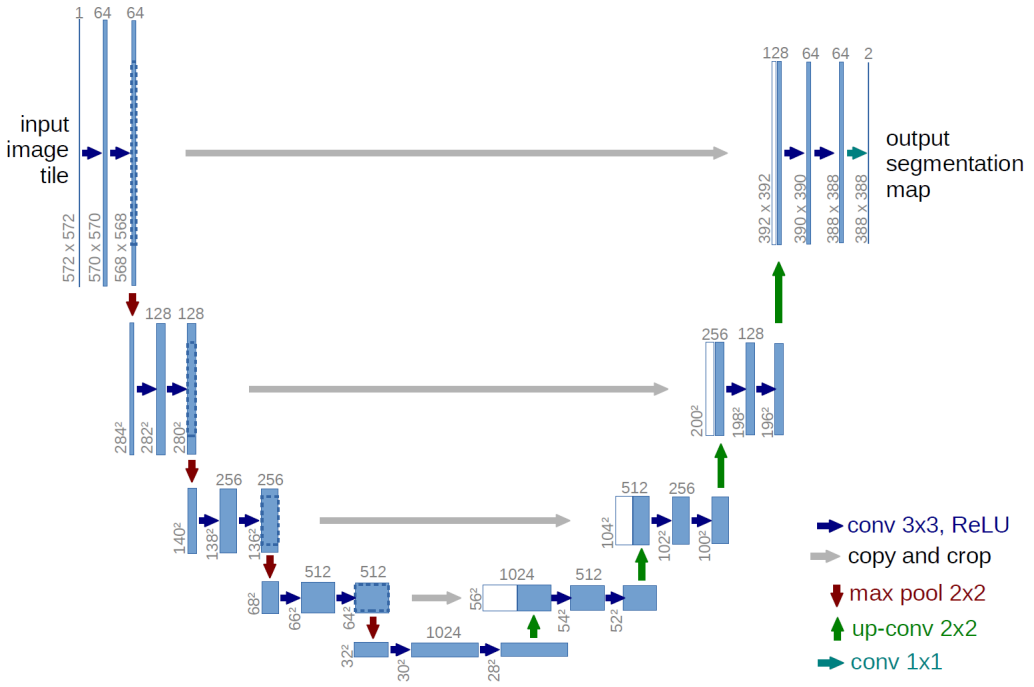


Figure 3: Visualization of the U-Net Architecture

3 Methods

3.1 Baseline

Our baseline architecture contains an encoder and a decoder structure. In our Dataloader, we first resize the images to 128 * 256 to reduce the training time. The last layer is the convolution with 32 input channels and 27 output channels followed by a softmax layer to predict the classes of each pixel. In baseline model, We use cross entropy loss, Adam optimizer, and learning rate 0.01.

In our improved baseline model, We try different loss criteria, including dice loss and weighted cross entropy loss, and data augmentation methods. We find that the weighted loss weighted by the reverse of the initial weights with square root works best. Weighted loss weights infrequent classes more to address data imbalance problem. We first sum up all the class of each pixel in all training set and normalize the each class by the number of all pixel. Then we divide 1 by weights of each class to get the reciprocals. Since the maximum of the weights is almost 10^4 times larger than the minimum, we square three times to get the final weights. We also try data augmentation methods by mirroring all the training set to double the training set. Although we get pretty good results, about 63% of IoU, it largely penalize the training time. Therefore, we use weight loss as our method to addressed the class-imbalance problem in the following experiments.

3.2 Experimentation

3.2.1 Mobile-Deeplabv3+

In this work, we revised the Deeplabv3+ model by using MobileNet as backbone. The model is composed of three modules: MobileNet, ASPP, and decoder. The detailed architectures are stated in Table 1, 2, 3, 4, 5. We choose Adam as our optimizer, and we use weighted crossentropy to deal with imbalanced data, which is discussed in Section 5.

Layer Index	Layer Dimension
MobbileNet	
Conv2d, BatchNorm2d, ReLU	in_channels=3, out_channels=32, kernel_size=3, stride=2, padding=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d	in_channels=32, hidden_channels=32 out_channels=16, kernel_size-1=3, stride-1=1, padding-1=0, dilation-1=1, kernel_size-2=1, stride-2=1, padding-2=0, dilation-2=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=16, hidden_channels=96 out_channels=24, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=2, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=24, hidden_channels=144 out_channels=24, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1

Layer Index	Layer Dimension
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=24, hidden_channels=144 out_channels=32, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=2, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=32, hidden_channels=192 out_channels=32, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=32, hidden_channels=192 out_channels=32, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1

Table 1: Mobile-Deeplabv3+ Architecture MobileNet-part1

Layer Index	**Layer Dimension**	**Layer Index**	**Layer Dimension**
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=32, hidden_channels=192 out_channels=64, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=2, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1	Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=64, hidden_channels=384 out_channels=96, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=64, hidden_channels=384 out_channels=64, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1	Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=96, hidden_channels=576 out_channels=96, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=64, hidden_channels=384 out_channels=64, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1	Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=96, hidden_channels=576 out_channels=96, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=64, hidden_channels=384 out_channels=64, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1	Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=96, hidden_channels=576 out_channels=160, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1

Table 2: Mobile-Deeplabv3+ Architecture MobileNet-part2

Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=160, hidden_channels=960 out_channels=160, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=160, hidden_channels=960 out_channels=160, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=1, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1
Conv2d-1, BatchNorm2d, ReLU, Conv2d-2, BatchNorm2d, ReLU, Conv2d-3, BatchNorm2d, ReLU	in_channels=160, hidden_channels=960 out_channels=320, kernel_size-1=1, stride-1=1, padding-1=0, dilation-1=1, kernel_size-=1, stride-2=1, padding-2=0, dilation-2=2, kernel_size-3=1, stride-3=1, padding-3=0, dilation-3=1

Table 3: Mobile-Deeplabv3+ Architecture MobileNet-part3

Conv2d, BatchNorm2d, ReLU	in_channels=320, out_channels=256, kernel_size=1, stride=1, padding=0, dilation=1
Conv2d, BatchNorm2d, ReLU	in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=12, dilation=12
Conv2d, BatchNorm2d, ReLU	in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=24, dilation=24
Conv2d, BatchNorm2d, ReLU	in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=36, dilation=36
Conv2d, BatchNorm2d, ReLU	in_channels=1280, (concat) out_channels=256, kernel_size=1, stride=1, padding=0, dilation=1

Table 4: Mobile-Deeplabv3+ Architecture ASPP

Conv2d, BatchNorm2d, ReLU	in_channels=48, (using interpolate to resize) out_channels=256, kernel_size=3, stride=1, padding=12, dilation=12
Conv2d, BatchNorm2d, ReLU	in_channels=304, (concat) out_channels=256, kernel_size=1, stride=1, padding=0, dilation=1
Conv2d, BatchNorm2d, ReLU	in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=24, dilation=24
Conv2d, BatchNorm2d, ReLU	in_channels=256, out_channels=27, kernel_size=3, stride=1, padding=36, dilation=36

Table 5: Mobile-Deeplabv3+ Architecture Decoder

3.2.2 VGG

We use the pre-trained VGG as encoder and use the same decoder as the baseline model. We also use Xavier weight initialization for the decoder. The detailed architectures are stated in Table 6. Again, we choose Adam as our optimizer, and we use weighted cross entropy to deal with imbalanced data, which is discussed in Section 5.

Layer Index	**Layer Dimension**
N/A	Pre-trained VGG
ConvTranspose2d, BatchNorm2d, ReLU	in_channels=encoder_out_channel, out_channels=512, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1
ConvTranspose2d, BatchNorm2d, ReLU	in_channels=512, out_channels=256, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1
ConvTranspose2d, BatchNorm2d, ReLU	in_channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1
ConvTranspose2d, BatchNorm2d, ReLU	in_channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1
ConvTranspose2d, BatchNorm2d, ReLU	in_channels=64, out_channels=32, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1
Conv2d	in_channels=32, out_channels=self.n_class, kernel_size=1

Table 6: VGG Architecture

3.2.3 UNet

We adopt the U-Net architecture as in Table 7. We use Xavier weight initialization for the encoder and decoder. Again, we choose Adam as our optimizer, and we use weighted cross entropy to deal with imbalanced data, which is discussed in Section 5.

Layer Index	**Layer Dimension**
Conv2d, ReLU, Conv2d, ReLU	in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1 in_channels=64 out_channels=64, kernel_size=3, stride=1, padding=1
MaxPool2d, Conv2d, ReLU, Conv2d, ReLU	in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1 in_channels=128 out_channels=128, kernel_size=3, stride=1, padding=1
MaxPool2d, Conv2d, ReLU, Conv2d, ReLU	in_channels=128, out_channels=256, kernel_size=3, stride=1, padding=1 in_channels=256 out_channels=256, kernel_size=3, stride=1, padding=1
MaxPool2d, Conv2d, ReLU, Conv2d, ReLU	in_channels=256, out_channels=512, kernel_size=3, stride=1, padding=1 in_channels=512 out_channels=512, kernel_size=3, stride=1, padding=1
MaxPool2d, Conv2d, ReLU, Conv2d, ReLU	in_channels=512, out_channels=1024, kernel_size=3, stride=1, padding=1 in_channels=1024 out_channels=1024, kernel_size=3, stride=1, padding=1

Layer Index	**Layer Dimension**
ConvTranspose2d, Conv2d, ReLU, Conv2d, ReLU	in=1024, out=512, kernel=2, stride=2, padding=0 in=1024, out=512, kernel=3, stride=1, padding=1 in=512, out=512, kernel=3, stride=1, padding=1
ConvTranspose2d, Conv2d, ReLU, Conv2d, ReLU	in=512, out=256, kernel=2, stride=2, padding=0 in=512, out=256, kernel=3, stride=1, padding=1 in=256, out=256, kernel=3, stride=1, padding=1
ConvTranspose2d, Conv2d, ReLU, Conv2d, ReLU	in=256, out=128, kernel=2, stride=2, padding=0 in=256, out=128, kernel=3 stride=1, padding=1 in=128, out=128, kernel=3, stride=1, padding=1
ConvTranspose2d, Conv2d, ReLU, Conv2d, ReLU	in=128, out=64, kernel=2, stride=2, padding=0 in=128, out=64, kernel=3, stride=1, padding=1 in=64, out=64, kernel=3, stride=1, padding=1
Conv2d	in=64, out=n_class, kernel=1, stride=1, padding=0

Table 7: U-Net Architecture

4 Results

In this section, we present the training, validation, and test results for the above models, Baseline, Improved Baseline, Mobile-Deeplabv3+, transfer learning - VGG, and UNet. Section 4.1 presents the training and validation loss curves for each model. Section 4.2 presents the validation set pixel accuracy, average IoU, and IoU of the class for road(0), sidewalk(2), car(9), billiboard(17), and sky(25). Section 4.3 shows the visualizations of the segmented output for the first image in the provided *test.csv* overlaid on the image.

4.1 Training and Validation Loss Curves

This section presents the training and validation loss for the baseline model, improved baseline model, Mobile-Deeplabv3+, transfer learning – VGG, and U-Net.

4.1.1 Baseline

Figure 4 shows the training and validation loss of baseline Model, we adopted early stopping with *early_stopping_epoch*=3 to avoid overfitting and the training procedure stops at epoch 16. The minimum validation loss is 0.594.

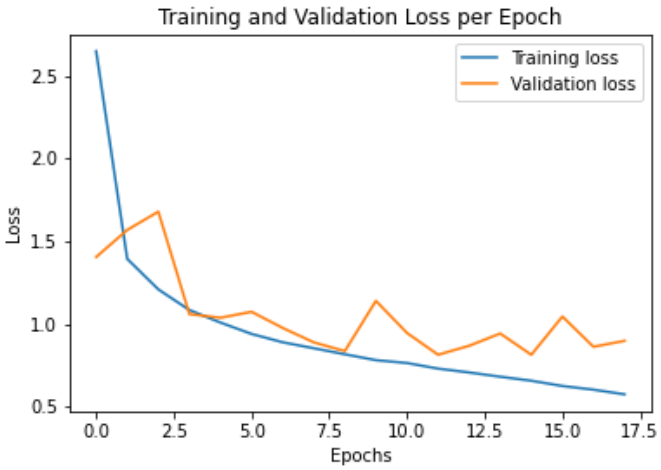
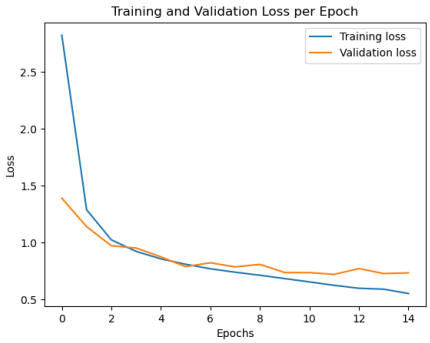


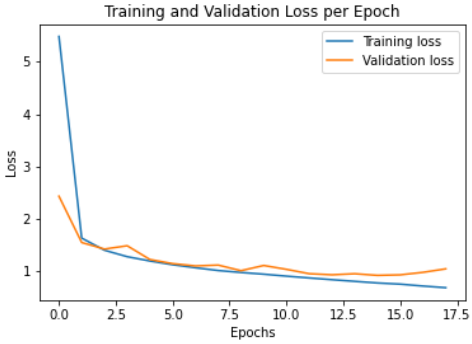
Figure 4: Training and Validation Loss of Baseline Model

4.1.2 Improved Baseline

Figure 5a shows the training and validation loss of improved baseline Model using data augmentation with rotate and crop, and Figure 5b shows the training and validation loss of weighted loss. We adopted early stopping with *early_stopping_epoch*=3 to avoid overfitting and the training procedure stops at epoch 16 and epoch 14. The minimum validation loss is 0.7352 and 0.9823s.



(a) Training and validation loss of baseline model with data augmentation



(b) Training and validation loss of baseline model with weighted loss

Figure 5: Loss for improved baseline model

4.1.3 Mobile-Deeplabv3+

Figure 6 shows the training and validation loss of Mobile-Deeplabv3+ model. We adopted early stopping with *early_stopping_epoch*=3 to avoid overfitting and the training procedure stops at epoch 20. The minimum validation loss is 0.6304.

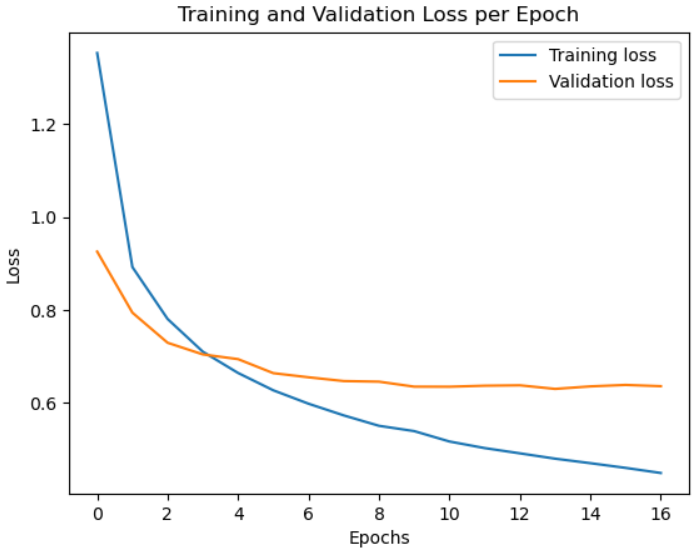


Figure 6: Training and Validation Loss of Mobile-Deeplabv3+

4.1.4 Transfer Learning - VGG

Figure 7 shows the training and validation loss of the transfer learning with VGG-16 as encoder. We adopted early stopping with *early_stopping_epoch*=3 to avoid overfitting and the training procedure stops at epoch 10. The minimum validation loss is 0.8627.

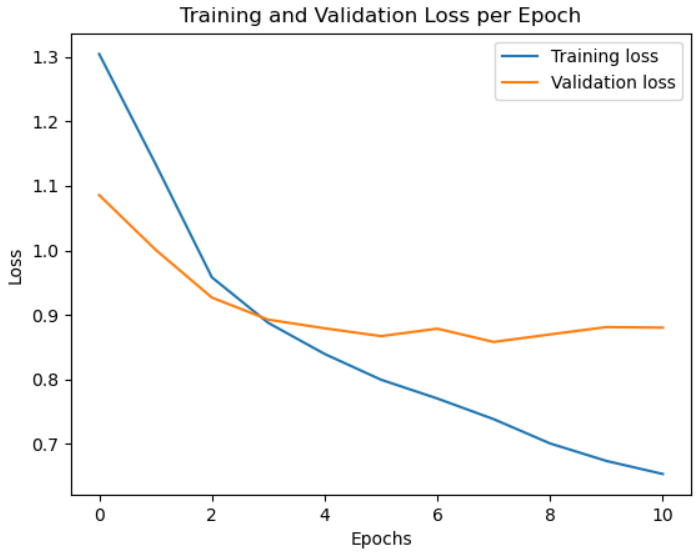


Figure 7: Training and Validation Loss of VGG

4.1.5 UNet

Figure 8 shows the training and validation loss of UNet model. We adopted early stopping with *early_stopping_epoch*=3 to avoid overfitting and the training procedure stops at epoch 14. The minimum validation loss is 0.8397.

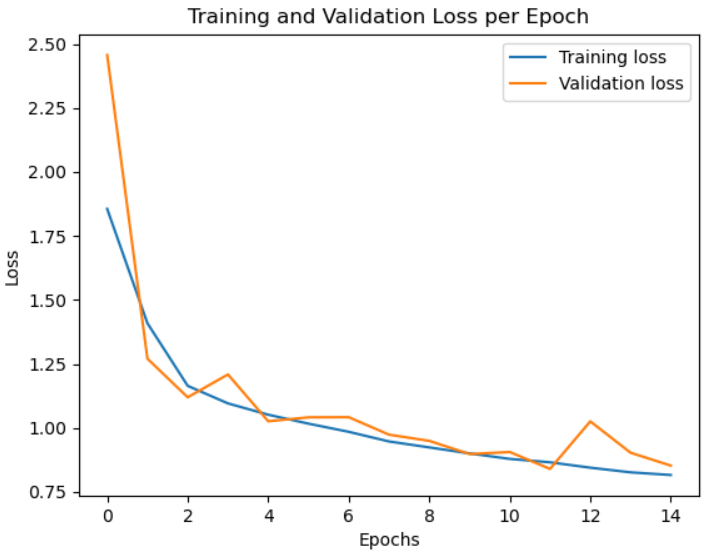


Figure 8: Training and Validation Loss of UNet

4.2 Validation Pixel Accuracy and IoU

In this section, we presented the pixel accuracy, average IoU, and IoU of the class for road(0), sidewalk(2), car(9), billiboard(17), and sky(25) of the validation set for each model we experimented on.

Model	Pixel Acc	Avg IoU	Road	Sidewalk	Car	Billboard	Sky
Baseline	0.7453	0.5949	0.8348	0.0892	0.4016	0.0085	0.9017
Improved(DiceLoss)	0.6444	0.4769	0.7114	0.0	0.0	0.0	0.8652
Improved(WeightedLoss1)	0.5938	0.423	0.7005	0.1010	0.3538	0.1185	0.8355
Improved(WeightedLoss2)	0.759	0.6137	0.8449	0.1143	0.4147	0.0683	0.9114
M-Deeplabv3+	0.8259	0.7039	0.9026	0.2590	0.5648	0.1601	0.9362
VGG	0.7296	0.5838	0.8009	0.0995	0.4244	0.1035	0.7891
UNet	0.7742	0.6323	0.8654	0.0026	0.3932	0.1078	0.9398

Table 8: Validation Average Pixel Accuracy, Average IoU, and IoU for 5 Classes

4.3 Visualizations of Segmented Output

In this section, we present the visualizations of the segmented output for the first image in the provided *test.csv* overlaid on the image for different models.

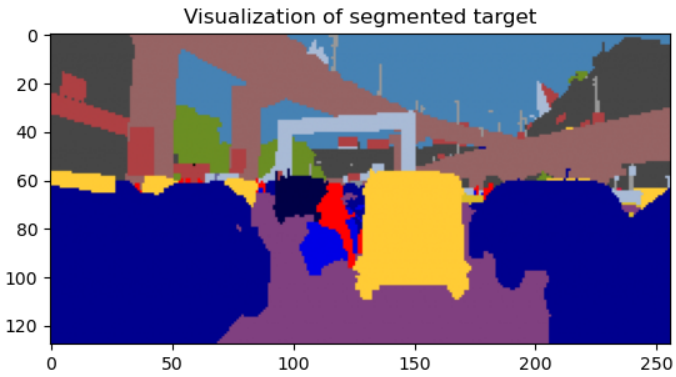
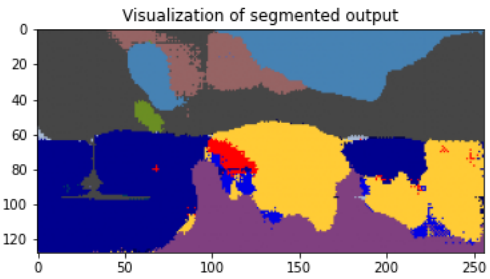
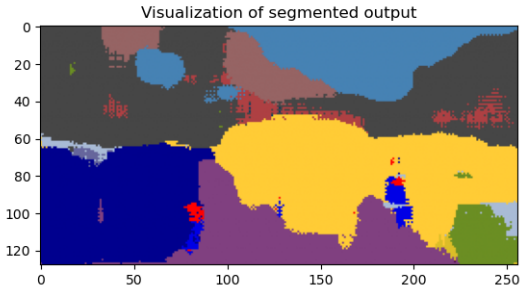


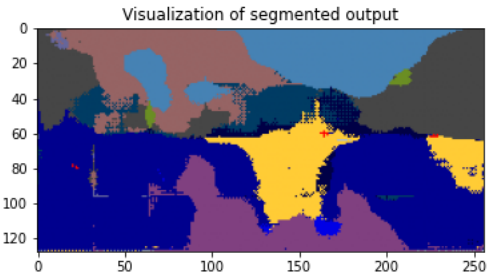
Figure 9: Visualizations of the Segmented Target



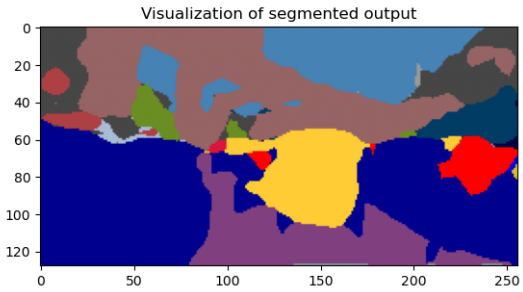
(a) Baseline



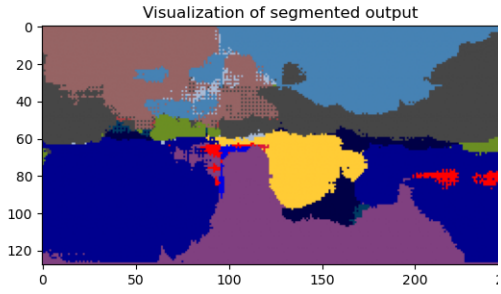
(b) Improved Baseline (Data Augmentation)



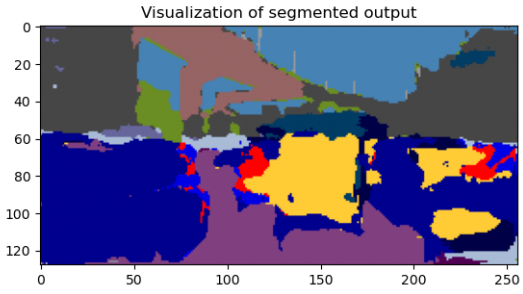
(c) Improved Baseline (Weighted Loss)



(d) Mobile-Deeplabv3+



(e) VGG



(f) U-Net

Figure 10: Visualizations of the Segmented Output

5 Discussion

In our baseline, we get the 59% IoU and 74% pixel accuracy. To improve our baseline model, we augment our dataset by applying mirror flip to double the training set. Since we double the training set with augmentation, we get better IoU with 63%. However, because the training set is larger, we get longer training time, which is almost three times than original one. Therefore, to reduce the training time to get more time to do the following experiment, we try weighted loss method. Weighted loss pressuring the network to learn infrequently seen classes. However, in this task, we find that using weights of reciprocal of initial weights will pressure the network too much for learning rarely classes and get worse average IoU result. We get worse IoU than the baseline results, however, the IoUs for low frequent classes works well when using only the reciprocal the initial weights, which is shown in Table 8 weightedLoss1. Therefore, we try to alleviate the pressure by squaring the weights. After the experiments, we find that square the weights three times get the best average IoU and pixel accuracy, which is shown Table 8 weightedLoss2. We have also conducted the experiment of dice loss for the loss criterion. Unfortunately, we do not get the well result as the baseline. Thereforem, we use the same setting of weightedLoss2 in the following experiments.

It is interesting to see that the Transfer Learning - VGG does not outperform the baseline model. It achieves 72.96% average pixel accuracy and 58.38% average IoU on the validation set. We used the backbone part obtained from the pre-trained model and change the decoder part. However, the pre-trained model may have been trained for different purposes, so in this case the performance is not the best since the model is not fully fitted on the Idd dataset. In other words, it suffered from the most common issue in the transfer learning – domain adaption. It happens when applying a model trained in one or more "source domains" to a different "target domain". Nevertheless, the advantage of the pre-trained model is that it is more efficient without using too much computer resources. As shown in Figure 7, it early stops at epoch 10, which is the much faster than all the other models we experimented.

UNet beats the baseline model: it achieves 77.42% average pixel accuracy and 63.23% average IoU on the validation set. Compared to the baseline model, it has larger number of channels in up sampling part. Note that baseline has 512 channels and UNet has 1024 channels, so it can capture more detailed features. Furthermore, there is a slight difference for the decoder method between UNet and baseline model. In the baseline model, the same level encoder feature map and decoder feature map are simply added to the next layer right away. On the other hand, UNet model concatenates the encoder feature map and decoder feature map to feed into the convolutional layers for additional processing. From the Figure 10f and 10a, we see that 10f has a clear contour of person and bridge.

Mobile-Deeplabv3+ outperforms both Unet and the transfer learning method. First, our encoder, MobileNet, is deeper than the encoder in Unet, which helps the model encode rich semantic information. The idea of learning multi-scale features in Unet is also leveraged in Mobile-Deeplabv3+ but better. Astrous convolution captures multi-scale information by adjusting filter's field-of-view, and the decoder combines the lower-level feature from encoder and higher-level feature after astrous convolution. We can also see this effectiveness of learning multi-scale feature when comparing the results with transfer learning. VGG and MobileNet are common encoders, and they have similar accuracy and the ability to extract visual feature, in some cases, VGG could have a better accuracy than MobileNet. With a similar behavior of encoding, we can now see the importance of learning multi-scale features in Mobile-Deeplabv3+.

6 Authors' Contributions and References

6.1 Individual Contributions to the Project

6.1.1 Zi-Xiang Xia

I am responsible for baseline model, and come up with a new architecture from 5a section in the write-up. We ran experiments, discussed results and wrote report together.

6.1.2 Yun-Yi Lin

I am responsible for baseline debugging and improved baseline coding, and report writing. We ran experiments, discussed results and wrote report together.

6.1.3 Zhuojun Chen

I implemented the backbone of transfer learning - VGG and UNet architecture. We ran experiments, discussed results and wrote report together.

6.1.4 Peiyi Li

I implemented the evaluation metrics. I also debugged and optimized VGG and UNet. We ran experiments, discussed results and wrote report together.

6.2 References

[1] Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI (2017)*

[2] . Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861 (2017)*

[3] Simonyan, K., Schnell, E. & Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR 2015*

[4] Ronneberger, O., Fischer, P. & Brox, T. (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation. *MICCAI 2015*

[5] L.C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *ECCV 2018*

[6] Chen, L.C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587 (2017)*

[7] Chollet, F.: Xception: Deep learning with depthwise separable convolutions. *CVPR 2017*