PA3 Report: Convolutional Neural Network Implementation for Semantic Segmentation on the Cityscapes Dataset

Hao Xiang
Computer Science & Engineering
UC San Diego
La Jolla, CA 92093
haxiang@ucsd.edu

Huimin Zeng
Computer Science & Engineering
UC San Diego
La Jolla, CA 92093
zenghuimin@yahoo.com

Xingyi Yang
Computer Science & Engineering
UC San Diego
La Jolla, CA 92093
x3yang@ucsd.edu

Zhenrui Yue
Computer Science & Engineering
UC San Diego
La Jolla, CA 92093
yuezrhb@gmail.com

Abstract

This document is the report of the first programming assignment (PA3) in CSE 253 Neural Networks/Pattern Recognition at UC San Diego in Winter 2020 by Hao Xiang, Huimin Zeng, Xingyi Yang and Zhenrui Yue. The report is based on the requirements of the assignment and contains sections for the programming problems as well as the individual contributions of each team member in this group.

In this report, inspired by Deeplab v3 [1], we implemented Astrous Spatial Pyramid Pooling (ASPP) based neural network and we implemented ASPP with various backbones (including ResNet and U-Net). Also, we implemented many other neural networks for semantic segmentation including FCN, UNet and ResNet+Decoder architectures. To increase our accuracy, we utilized various techniques like dice loss, weighted loss, and data augmentation, skip connection, atrous convolution, spatial pyramid pooling etc. Our baseline model achieved XXX mIoU while our ASPP model with ResNet 50 achieved the overall XXX accuracy and XXX mIoU, which beats FCN, and UNet and ResNet.

1 Introduction

Cityscapes Dataset is an urban visual understanding dataset based on over 50 cities in Europe [2]. With over 20,000 images of urban traffic and corresponding pixel labelling, each label giving one of the 30 classes of possible objects like road and vehicle. With this dataset, we would like to create a convolutional neural network to perform the task of semantic segmentation, which requires the model to recognize the image in pixel level, giving labels to all pixels and hence tell all possible different objects in the image.

For this purpose, it is necessary to train convolutional neural networks (CNN) with Xavier weight initialization and batch normalization, CNNs are a certain type of neural networks commonly used for image processing and computer vision, typical structures of CNN usually contain convolutional layers, pooling layers and fully connected layers. Convolutional layers are usually used for incrementing receptive fields and extracting visual features, pooling layers could be either increasing sizes of receptive fields and strengthen the model invariance against translation and resizing, while fully connected networks would process the

information from previous layers and perform classification task using softmax function.

The training process of a CNN model would require weights initialization and input normalization for better generalization, in our case, we adopted Xavier weight initialization and batch normalization. Xavier weight initialization is a specific method of weight initialization, this method ensures optimal initial weights so that vanishing gradients and saturated neurons could be avoided. The variance of each layer is expressed as:

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

This considers the input number n_{in} and the output number n_{out} so that the variances of each weight matrix are comparable, this could significantly speed up the training period. Batch normalization is an input preprocessing method, it normalizes the input in every batch with the following equation:

$$x_i' = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

Where μ represents the batch mean, σ the standard deviation of the batch and ε a very small value in case of 0 batch standard deviation in denominator. Applying batch normalization to each batch, the normalized input could be transformed into data with zero mean and unit variance. With normalized data, the network weights no more depend on the range of the input data, and the model training process become less reliable on the initialization of the network weights, with the model itself more generalized and capable of predicting on equally preprocessed data.

2 Related Work

In this section, we present a few related models and published work in the field of computer vision and semantic segmentation, based on which we have created modified models or reproduced the identical models for our task. These models and implementation methods will be introduced in the following subsections.

2.1 Very Deep Convolutional Network (VGG)

The very deep convolutional network is a deep CNN network proposed by Simonyan et al. at the Visual Geometry Group from University of Oxford [3]. The paper proposed two CNN models with respectively 16 and 19 layers, designed for the task of large-scale image recognition. Unlike AlexNet by Krizhevsky et al. [4], the major difference of VGG structure is the adoption of a much smaller convolutional kernel (3x3 size) and a much deeper network structure, compared to 5x5 and 7x7 sized kernels and 8 total layers in AlexNet. This change reduces the total number of parameter and retains the most information from the input, as the layer goes deeper, the receptive fields grow larger and extracts features from the output of the previous layer. Therefore, VGG could extract features from features of the image and learns to recognize such patterns, which significantly improves its performance on most visual recognition tasks.

The structure of VGG with 16 layers and 19 layers could be found in the following image. As shown in the image, the VGG networks have mostly convolutional layers of different sizes in the front, which gradually increase the channels of the input. After a max-pooling layer, the network passes the input to fully connected layers to learn these extracted internal features and utilize softmax function to perform the image recognition task. The advantages of VGG are the simple structure, increased depth and smaller convolution kernels which enhances the capability to capture features, but it also requires large computational power in the training process due to the fully connected layers.

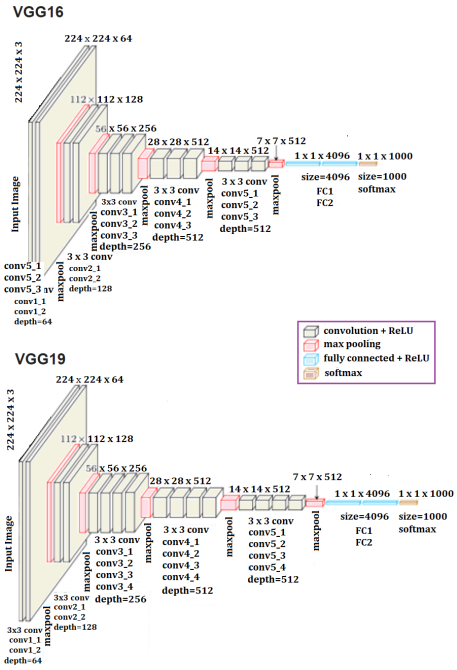


Figure 3: VGG-16 and VGG-19 Structure [4]

2.2 Deep Residual Network (ResNet)

In this part, we introduced a residual learning convolutional neural network by He et al. [5], this specific residual network (ResNet) and the shortcut connection residual block is specifically designed for very deep neural models, as the value passing among network layers could cause information loss, which results in lower model performance compared to similar models with fewer layers. One important contribution of this work is the introduction of residual building blocks to the layers of the convolution network. The purpose of this building block is to retain the original input of the network and prevent information loss among the different layers. Note that this requires the dimensions between layer input and output to be identical, if not so, a projection of the input x must be made to match to output dimension, this is shown in the following formula, where $\mathcal{F}$ represents the residual layer and W_s the projection matrix in case of dimension change.

$$y = \mathcal{F}(x, \{W_i\}) + x$$

$$y = \mathcal{F}(x, \{W_i\}) + W_s x$$

In the paper, the authors implemented two plain and two residual networks, each of 18 and 34 layers. The networks structures are shown below, the difference between plain and residual networks is the shortcut connection between layers, please refer to the image below for the plain and residual network structures. These models were trained and tested on ImageNet, in evaluation section, the plain networks perform worse than residual networks in error rate for most iterations. See below for evaluation results, the 18-layer plain network performs better than 34-layer without shortcut connection between layers, however, residual networks could significantly improve the model performance for deeper models, enabling the 34-layer ResNet to decrease its error rate to circa 0.28, reversing the results between the comparison of plain networks [5].

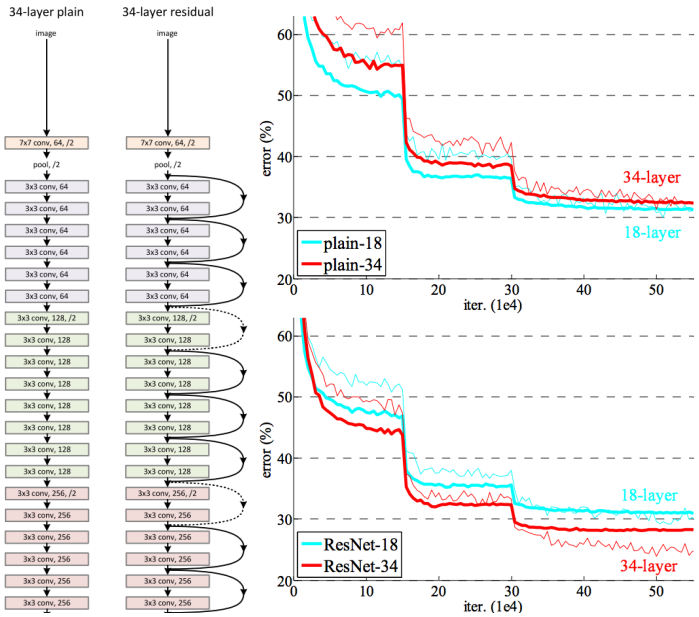


Figure 1: Model Structure and Evaluation Results of ResNets [5]

2.3 Convolutional Networks for Segmentation (U-Net)

This part aims to introduce a specific convolutional network for biomedical image segmentation by Ronneberger et al, which was also implemented in our assignment for semantic segmentation on the CityScapes dataset. The proposed model is a fully convolutional network with the main idea of first downsizing the input image with multiple convolutional layers with max-pooling operators, following by upsizing convolutional layers which that could learn to generate a more precise output. Additionally, this network adopted a few shortcut connections between the contracting and expansive layers, using the cropped input out each layer and concatenate the input to the corresponding upsamlping layers, giving the network both low-level and high-level features of the input image [6].

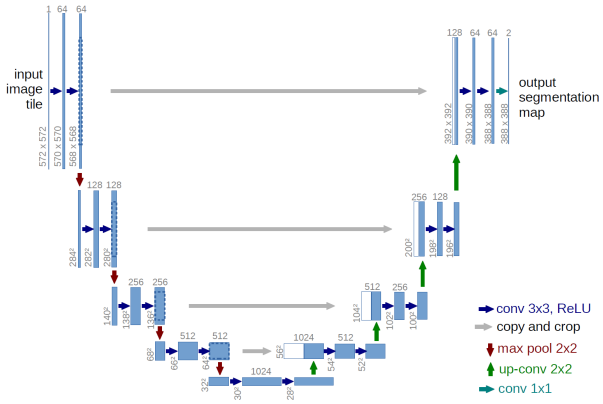


Figure 2: U-Net Structure [6]

The U-Net model structure first increases the input channels using downscaling convolutional layers, extracting multiple features from the corresponding input image and double the channel number with each further down step. After reaching the bottom of the network, the network starts to upsamling the image using a 2x2 convolution kernel, this process would half the input feature channels but produce output of greater size, meanwhile, the cropped input of the same size would be concatenated here, which would output an overall smaller image as the model output. This network structure is particularly well-performed for segmentation tasks in biomedical applications, it doesn't need large

amounts of labelled training data and has a reasonable training time. Therefore, we implemented and modified the U-Net in out assignment, then applied the model to test its performance for semantic segmentation task.

2.4 Atrous Convolution, and Fully Connected CRFs (DeepLab)

DeepLabv3 is a deep convolutional neural network (DCNN) proposed by Chen et al. [1] in the year of 2017, before DeepLabv3, there were also DeepLabv1 [7] and DeepLabv2 [8] from. Major contributions of DeepLabv1 and DeepLabv2 include the use of Atrous Convolution and conditional random field (CRF), the previous method is visually represented in the image below, which performs the convolution step in one of every few (e.g. two) pixels, which leads to an enlarged receptive fields and feature map of higher resolution, whereas conditional random field enhances the model utilization of global image details and features. Besides, the Atrous Spatial Pyramid Pooling (ASPP) method was also introduced in DeepLabv2, the reason for ASPP is a larger sampling rate and less valid weights for image processing, which is computationally more effective and broadens the range of input image resolution.

Overall, the DeepLab DCNN model achieves a higher efficiency and better classification performance with a relatively simple model structure for image semantic segmentation with improved Atrous Convolution, Atrous Spatial Pyramid Pooling and the combination of DCNN and CRF, which would retain the global input information along the neural network and process the image efficiently.

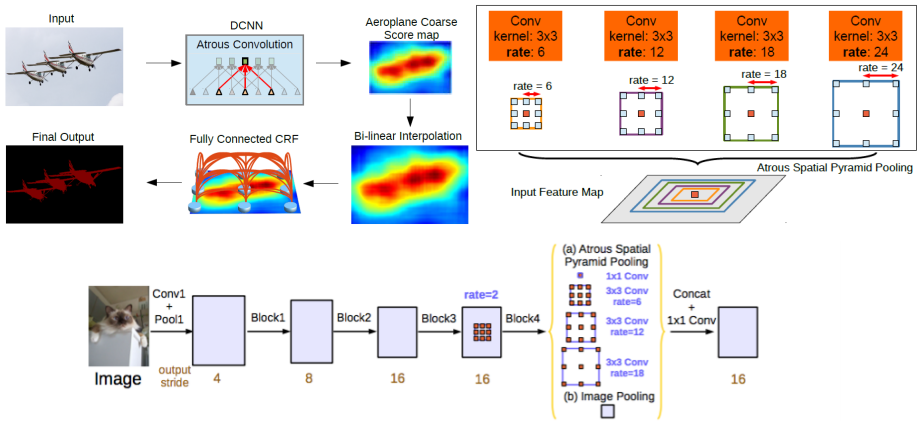


Figure 4: Atrous Convolution, Atrous Spatial Pyramid Pooling and DeepLabv3 [1][7][8]

3 Methods

A few convolutional neural network models for semantic segmentation have been created and evaluated on the CityScapes dataset, these model structures and their implementation will be introduced in the following parts. Please note that in all of our models, we utilized the cross-entropy objective function, Adam optimization method, initialized the weights with Xavier initialization.

3.1 Baseline Method (Fully Convolutional Network)

For this assignment, a baseline CNN model and some code for the training process was provided, based on the baseline model, we completed the first CNN network with five convolutional layers and five deconvolutional layers. Each convolutional layer consists of a convolution step (in most layers, the channels are doubled but the image size shrinked), followed by a batch norm step and a ReLU activation function. After four convolutional layers, the input data goes through five deconvolutional layers which upsizes the image and reduces the channel number, which has a transposed convolution step that halves the image channel (except first deconvolutional layer), then the identical batch norm and ReLU

activation function. Finally, the output would be passed to a last convolutional layer that performs similarly as a linear classifier and outputs the results with the same size of total class number.

No.	Elements	Configuration
1	*Conv2d, BatchNorm2d, ReLU*	*in_channels=3, out_channels=32, kernel_size=3, stride=2, padding=1, dilation=1*
2	*Conv2d, BatchNorm2d, ReLU*	*in_channels=32, out_channels=64, kernel_size=3, stride=2, padding=1, dilation=1*
3	*Conv2d, BatchNorm2d, ReLU*	*in_channels=64, out_channels=128, kernel_size=3, stride=2, padding=1, dilation=1*
4	*Conv2d, BatchNorm2d, ReLU*	*in_channels=128, out_channels=256, kernel_size=3, stride=2, padding=1, dilation=1*
5	*Conv2d, BatchNorm2d, ReLU*	*in_channels=256, out_channels=512, kernel_size=3, stride=2, padding=1, dilation=1*
6	*ConvTranspose2d, BatchNorm2d, ReLU*	*in_channels=512, out_channels=512, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
7	*ConvTranspose2d, BatchNorm2d, ReLU*	*in_channels=512, out_channels=256, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
8	*ConvTranspose2d, BatchNorm2d, ReLU*	*in_channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
9	*ConvTranspose2d, BatchNorm2d, ReLU*	*in_channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
10	*ConvTranspose2d, BatchNorm2d, ReLU*	*in_channels=64, out_channels=32, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
11	*Conv2d*	*in_channels=32, out_channels=self.n_class, kernel_size=1*

Table 1: Structure of the modified U-Net

3.2 Modified U-Net

In the last section, we introduced the U-Net model that performs well on specific segmentation tasks such as biomedical images. In this part, the modified structure of U-Net specifically designed for the CityScapes segmentation task would be described.

The modified version of U-Net has a few convolutional and upsampling layers, similar to the original U-Net structure. The first layers would utilize 2d convolution functions to downsize the image, usually processed by batch normalization and a ReLU activation afterwards, meanwhile max pooling elements are being added between these layers to enlarge the receptive fields. The rear part (deconv layers) of the network contains two parallel pipelines and applied the upsampling function to increase the output size while decreasing the image channels using 2d convolutions, the convolution layers (6, 7, 8, 9) would cut down the image channels to the same size and their outcome would be concatenated with the deconvolutional layers (deconv1, deconv2, deconv3, deconv4), note that batch normalization and ReLU are also adopted between these layers.

No.	Elements	Configuration
1	*Conv2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1), Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)*
2	*MaxPool2d, Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*MaxPool2d(kernel_size=2), Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1), Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)*
3	*MaxPool2d, Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*MaxPool2d(kernel_size=2), Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1), Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)*
4	*MaxPool2d, Conv2d,*	*MaxPool2d(kernel_size=2), Conv2d(in_channels=256,*

	BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU	*out_channels=512, kernel_size=3, padding=1), Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)*
5	*MaxPool2d, Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*MaxPool2d(kernel_size=2), Conv2d(in_channels=512, out_channels=1024, kernel_size=3, padding=1), Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, padding=1)*
deconv1	*Upsample, Conv2d, BatchNorm2d, ReLU*	*Upsample(scale_factor=2, mode='bilinear', align_corners=True), Conv2d(in_channels=1024, out_channels=512, kernel_size=3, padding=1)*
6	*Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*Conv2d(in_channels=1024, out_channels=512, kernel_size=3, padding=1), Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)*
deconv2	*Upsample, Conv2d, BatchNorm2d, ReLU*	*Upsample(scale_factor=2, mode='bilinear', align_corners=True), Conv2d(in_channels=512, out_channels=256, kernel_size=3, padding=1)*
7	*Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*Conv2d(in_channels=512, out_channels=256, kernel_size=3, padding=1), Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)*
deconv3	*Upsample, Conv2d, BatchNorm2d, ReLU*	*Upsample(scale_factor=2, mode='bilinear', align_corners=True), Conv2d(in_channels=256, out_channels=128, kernel_size=3, padding=1)*
8	*Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*Conv2d(in_channels=256, out_channels=128, kernel_size=3, padding=1), Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)*
deconv4	*Upsample, Conv2d, BatchNorm2d, ReLU*	*Upsample(scale_factor=2, mode='bilinear', align_corners=True), Conv2d(in_channels=128, out_channels=64, kernel_size=3, padding=1)*
9	*Conv2d, BatchNorm2d, ReLU, Conv2d, BatchNorm2d, ReLU*	*Conv2d(in_channels=128, out_channels=64, kernel_size=3, padding=1), Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)*
clf	*Conv2d*	*Conv2d(in_channels=64, out_channels=self.n_class, kernel_size=1)*

Table 2: Structure of the modified U-Net

3.3 Modified ResNet

To adapt the ResNet to our semantic segmentation task, we made some tiny modifications to the ResNet with 50 layers (ResNet-50). We first load the pre-trained ResNet-50 model from PyTorch, after that, we created a few extra deconvolutional layers and append these layers to the pre-trained model so that the ResNet could be adapted optimally for segmentation on our dataset.

Five deconvolutional layers of the identical structure but different parameters were added to the end of the original model, each consisting of transposed convolution, batch normalization and ReLU activation. These layers would reduce the total channels of the input, the channels were reduced to 32 and passed to a 2d convolutional layer, which then assign the results of all pixels for different object classes.

No.	**Elements**	**Configuration**
N/A	*N/A*	*Pre-trained ResNet-50*
deconv1	*ConvTranspose2d, BatchNorm2d, ReLU*	*channels=encoder_out_chnnel, out_channels=512, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
deconv2	*ConvTranspose2d, BatchNorm2d, ReLU*	*channels=512, out_channels=256, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
deconv3	*ConvTranspose2d, BatchNorm2d, ReLU*	*channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
deconv4	*ConvTranspose2d, BatchNorm2d, ReLU*	*channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*
deconv5	*ConvTranspose2d, BatchNorm2d, ReLU*	*channels=64, out_channels=32, kernel_size=3, stride=2, padding=1, dilation=1, output_padding=1*

clf	*Conv2d*	*in_channels=32, out_channels=self.n_class, kernel_size=1*

Table 3: Structure of the modified ResNet-50

3.4 ResNet + ASPP / U-Net + ASPP Model

Other than the previous models, we also created a neural network model for the segmentation assignment based on the ResNet and DeepLab model. Here, the major improvement is the use of Atrous Spatial Pyramid Pooling (ASPP), the input image is first passed through a convolutional layer, then seperately through a series of few ASPP steps of different rates consisting of an ASPP convolutional layer, batch normalization, ReLU activation and a dropout step (optional) to avoid overfitting, note that the output should keep the size of input image.

Parallelly, an average pooling layer was added to the model, with an average pooling element that will pool the input image, then a 2d convolution and ReLU activation, whose output will be bilinear interpolated to scale it to the same size of input. The next step is to concatenate the outputs from the processing series of different rates and the pooled image, before the result is finally fed to an encoder layer. The encoder layer is also made up of 2d convolution, batch normalization and ReLU. The concatenation of previous branches is processed here as the input, and the layer will compute the corresponding values for each class and give out the classification results.

No.	Elements	Configuration
1	*Conv2d*	*in_channel, out_channel=128, kernel_size=(1,1), stride=1, padding=0, dilation=1*
2	*Conv2d, BatchNorm2d, ReLU*	*in_channel, out_channel=128, kernel_size=(3,3), stride=1, padding=6, dilation=6*
3	*Conv2d, BatchNorm2d, ReLU*	*in_channel, out_channel=128, kernel_size=(3,3), stride=1, padding=12, dilation=12*
4	*Conv2d, BatchNorm2d, ReLU*	*in_channel, out_channel=128, kernel_size=(3,3), stride=1, padding=18, dilation=18*
5	*AdaptiveAvgPool2d, Conv2d, BatchNorm2d, ReLU*	*AdaptiveAvgPool2d(output_size=(1,1)), Conv2d(in_channel, out_channel=128, kernel_size=(1,1), stride=1, padding=0, dilation=1)*
6	*interpolate*	*interpolate(self.pooling(layer5), size=input.shape[2:], mode='bilinear', align_corners=True)*
7	*concatenate*	*concatenate(layer1, layer2, layer3, layer4, layer6)*
8	*Conv2d, BatchNorm2d, ReLU*	*in_channel=128*5, out_channel, kernel_size=(1,1), stride=1, padding=0, dilation=1)*

Table 4: Structure of the ASPP Model

Based on this ASPP model, we created the ResNet + ASPP model, in which the pre-trained ResNet-101 model was first imported, then an ASPP element was appended to ResNet, which forms the ResNet + ASPP in two of our own models. Then, we also applied ASPP to U-Net to improve the semantic segmentation performance of the original model on the CityScapes dataset, the U-Net + ASPP model is very similar to the modified U-Net structure mentioned in the previous sections. However, the deconv4 layer and the final classifier in the modified U-Net structure was replaced by two ASPP elements, U-Net + ASPP is the second of our own CNN models.

3.5 Modified Loss function

To address the problem of imbalanced data, we also try different loss functions: weighted cross-entropy and dice loss. Weighted loss assigns a weight term for each of the class during the computation of cross entropy, the weight term could be used to force the model learning certain underrepresented patterns and classes, as weight term would increase the original cross-entropy loss for less frequent classes in the training dataset and decrease the loss for

frequent ones, resulting in better performance on classes with small quantity. The formula for weighted cross-entropy loss could be found below:

$$\mathrm{L} = -\sum_{n}^{N}\sum_{\mathrm{k}}^{\mathrm{c}} w_k t_k^n \log\left(y_k^n\right)$$

The dice loss was also introduced in the assignment to tackle the unbalanced data issue, dice loss first computes the intersection of true positive data over the union of true labelled data and predicted data, the numerator and denominator are biased with a σ term, and the loss could be acquired using one subtracted by this result, dice loss is similar to weighted loss that utilizes IoU values of all image classes, so that for less frequent labels would also be considered, this performs particularly well with unbalanced image dataset, see formula:

$$\mathrm{L} = 1 - \frac{\sum_k^c |A_k \cap B_k| + \sigma}{\sum_k^c |A_k| + |B_k| + \sigma}$$

4 Results

This part included the training, validation and test results of the models we mentioned in the last section. For each mode, we present a single plot of the training process with training and validation loss curves, validation pixel accuracy, average IoU and IoUs of five classes: building (11), traffic sign (20), person (24), car (26) and bicycle (33). Please see below for average IoU values and for each of the five classes. The lower part of table 5 shows the result comparison of different loss functions (dice loss and weighted loss) and the last for augmented images modified from the baseline fully convolutional networks.

Method	**build.**	**sign**	**person**	**car**	**bike**	**mIOU**
Baseline	63.1	20.1	30.2	50.8	12.2	25.4
Resnet50	73.2	20.2	32.5	71.9	17.2	29.1
UNet	77.1	55.9	46.2	78	48.7	42.7
Deeplab	80.2	51.2	56.7	76.4	53.4	49.9
ResNet + ASPP	78.7	49.1	55.3	82.2	55.3	50.0
Baseline	63.1	20.1	30.2	50.8	12.2	**35.28**
Dice Loss	71	28	34.3	62.6	12.7	**41.72**
Weight. Loss	66.5	35.1	33.7	52.7	30.1	**43.62**
Augmented	66.5	35.1	33.7	52.7	30.1	**43.62**

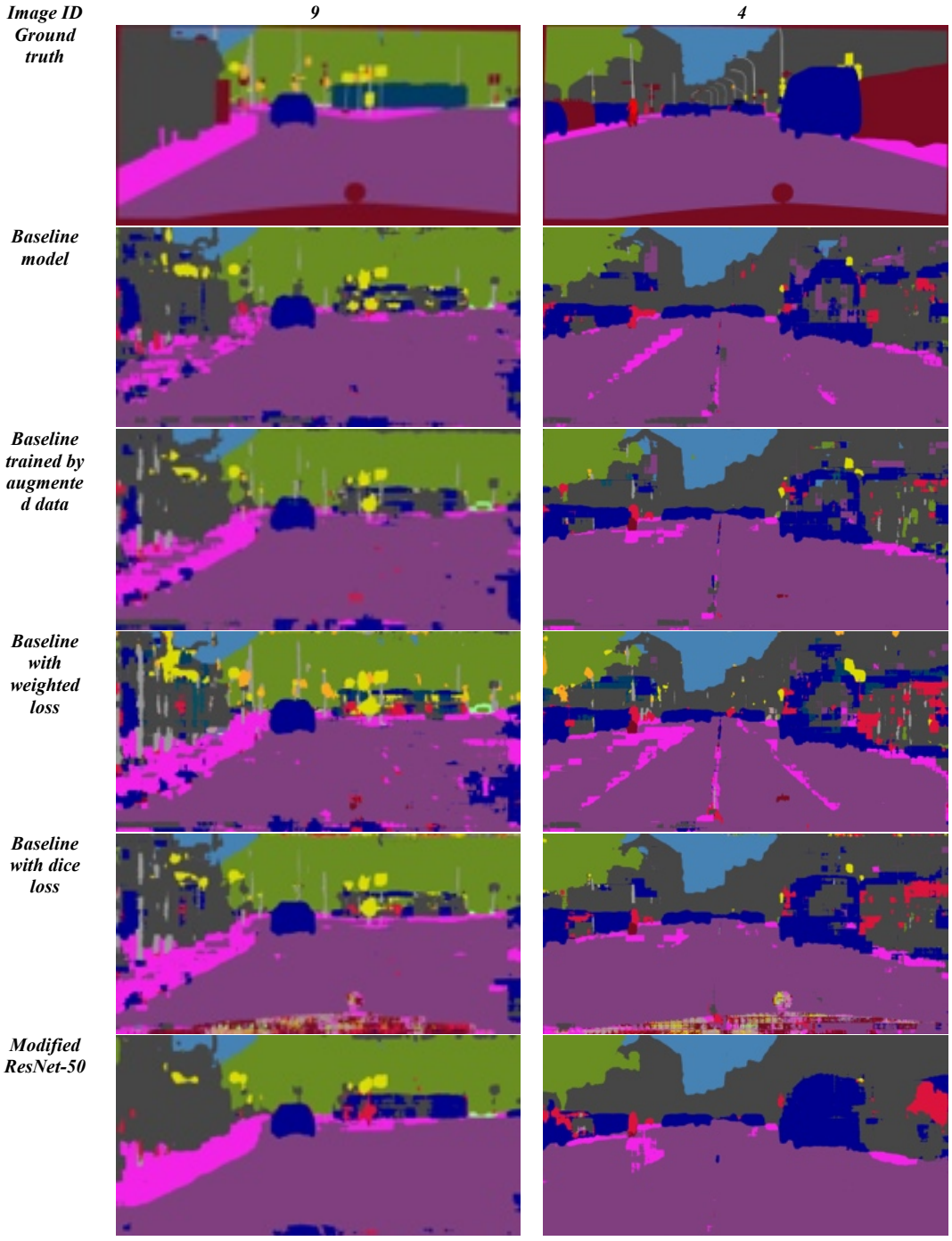
Table 5: IoU Results of Top-5 Classes

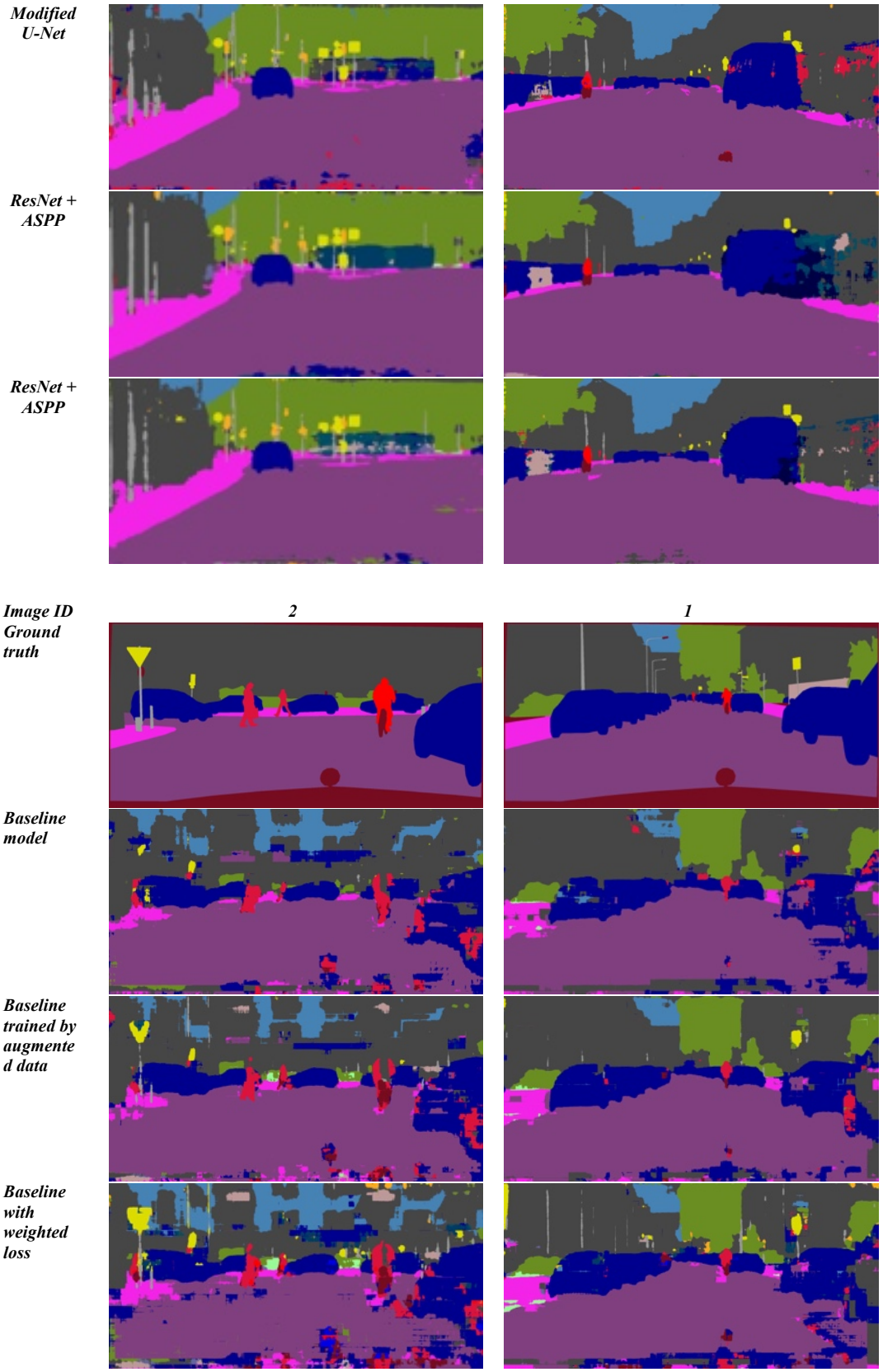
Method	**Pixacc**
Baseline	83.1
Resnet50	88.0
UNet	91.1
Deeplab	92.1
ResNet + ASPP	92.0
Baseline	83.1
Dice	87.1
Weight. Loss	83.6
Augmented	86.0

Table 6: Pixel Accuracy of Different Models

Table 6 shows the overall pixel accuracy of the five methods, each representing one model and the lower part of the table shows results for specific tasks. Another important result for these models is the training process, here we post the training loss curve

The segmented output of four (ID: 1, 2, 4, 9) images in the test set of all different models compared to the visualized ground truth is also attached for comparison of the models, please see table 6 for the segmented images.




Modified U-Net
ResNet + ASPP
ResNet + ASPP
Image ID
Ground truth
2
1
Baseline model
Baseline trained by augmented data
Baseline with weighted loss

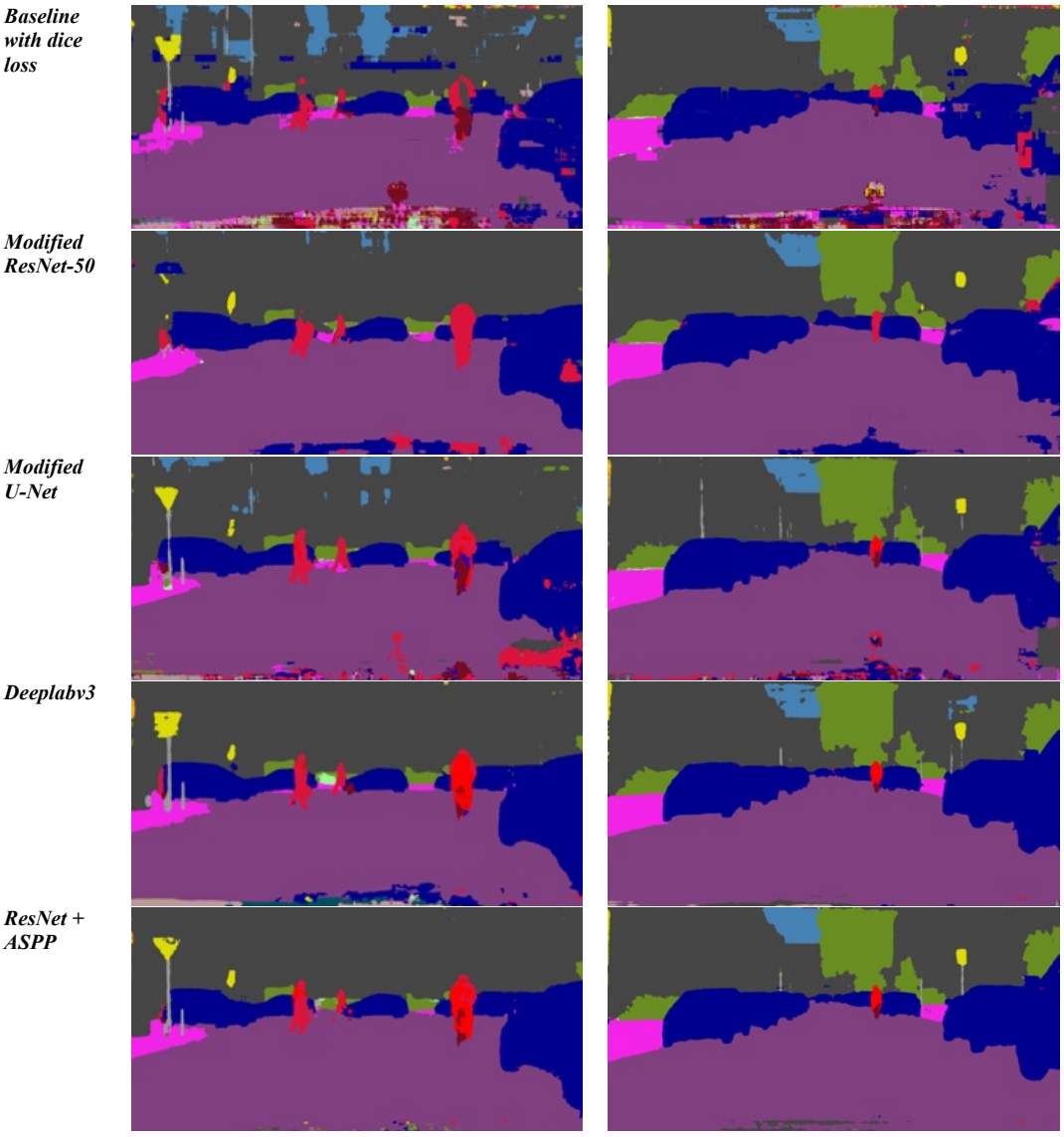


Table 7: Visualized Segmented Output of Different Models

4.1 Baseline Model

We exploit to adopt Fully Convolutional Network on Cityscape dataset as our baseline model. It achieves poor IoU performance with mIoU=25.41%. This experiment illustrates that plain encoder-decoder architecture is insufficient to tackle semantic segmentation task with large environment variation and unbalanced data.

4.2 Weighted Loss and Dice Loss

One of the common ways to tackle the imbalanced data problem is to modify the loss function. By setting reasonable class weight parameter, baseline with weighted loss surpasses the original model by 7% percent on mIoU. Misclassification errors of the less frequent classes can be up-weighted in the cross-entropy loss. This result in higher IOU of “traffic sign” and “bicycle”. Dice loss has the same weighting mechanism for segmentation. It treats the evaluation metric directly as the optimization target, which result in higher evaluation performance. It also marginally improves the baseline, reaching 29.6% mIoU and 87.1% pixel accuracy.

4.3 Transfer Learning

To explain the importance of different network topology and its impact on semantic segmentation, we also switch the encoder part of the baseline to the pretrained ResNet50 model. With a more sophisticated network, the ResNet-FCN yield a mIoU of 29.1%. It demonstrates that, deep network and skip-connection is beneficial for the segmentation task.

4.4 U-Net

U-net architecture is even powerful for the validation set of Cityscape challenge. We can see that the skip connection design dramatically increases the mIoU by approximately 15% compared with the baseline. The fusion of multilayer feature map further strengthens the generalization capability of CNN, which is even comparable with more sophisticated model with ASPP layer.

4.5 ResNet + ASPP

We implemented Astrous Spatial Pyramid Pooling (ASPP) and add it to the decoder part of the U-Net and ResNet. And the best IoU we can get is around 0.5 while the pixel accuracy is 92%, beating all the other models in our experiments. After experimenting with several combination of backbones including U-Net, ResNet50, ResNet101, VGG etc. we find that the best combination for ASPP is to use ResNet50 as the backbone. The training/validation loss is shown in the figure []. Also, the visualization of the output of the network is shown in the table 7. And we can see that our model can successfully segment many objects with various shapes like the humans, the traffic signs, cars etc. Also, our model can generate more smooth prediction. This is largely due to the architecture of skip connection, astrous convolution and spatial pyramid pooling architecture. And we will further discuss this in the next discussion section.

5 Discussion

Concerning a semantic segmentation task, we only implemented Fully Convolutional Networks to extract the visual features from the images. The classification is directly conducted using an extra convolutional layer. That is, no fully connected layers were used in any of our models. Moreover, given the knowledge that the performance of neural networks largely depends on the capacity of themselves. Therefore, we built several neural networks with different architectures and different depths. For instance, the skip connection architectures were realized via Residual Learning (ResNet) structure and U-Net structure. In addition, for a better comparison, we choose ResNet50 and ResNet101 as backbone. We observed that the performance could be greatly improved with the growth of the depth of the model.

5.1 Depth of the Model

For further improvement on the segmentation task, we could draw the conclusion that a model would achieve better performances with a deeper architecture, this could be proved by the 16- or 19-layer VGG model, which was first introduced for an image classification task and achieved outstanding result. This was also reflected in our assignment, regardless of the minor modifications, the best-performing model in this assignment is ResNet + ASPP, which was based on a 50-layer ResNet with an additional ASPP element and achieved the best semantic results among all the models we experimented for the task.

5.2 Combination of High- and Low-Level Features

Another important factor is to retain the image information and low-level features when the input image is passed forward through layers. For very deep neural network models, the input information is filtered multiple times by its layers that often contain non-linear operations such as max-pooling and ReLU activation, which could often lead to loss of low-level information. As the input was processed further, certain features are left behind in

the network and only the filtered high-level information are being fed to a final classifier. Another important reason for retaining low-level information is to prevent vanishing gradients as the network goes deeper, since the multiple layers of activation functions (such as sigmoid and tanh function) could possibly reduce or even set the values of gradients to zero, this would cause very slow learning behavior or even stops the model from learning anything.

In this sense, different low-level feature retaining methods are introduced here, in this assignment we adopted skip connection in ResNet and U-Net model. In ResNet the original input vector would be passed further using skip connection, so that the original input is passed forward and certain low-level features would be kept for further processing. Similarly, the U-Net also passed the different input values from convolutional layers to the upsampling (deconvolutional) layers. The gradient vanishing problem is avoided in this way that the gradients could be concatenated with greater magnitude of the input vector. In this way, deep neural networks could conclude better results with residual blocks (like skip connections), as proved in the previous sections. In our experiment, our models ResNet + ASPP and U-Net + ASPP

5.3 Extraction of Global Features

I report.

5.2 XXXXXXXXXX

I report.

6 Individual Contributions

This part included the individual contributions of the two team members Hao Xiang, Huimin Zeng, Xingyi Yang and Zhenrui Yue in the programming assignment.

6.1 Hao Xiang

Hao Xiang implemented models including U-Net, ResNet + ASPP, U-Net + ASPP and build the pipeline for train.py as well as the evaluation metrics including IOU calculation, accuracy calculation, as well as contribute to the documentation of the code and README file for the repository. Also, Hao contributes to the discussion, results and methods section in the assignment report.

6.2 Huimin Zeng

Huimin Zeng implemented the first version of the baseline model and the function loading other pretrained models, data loader, visualizer, and plot function for saving the training curves and other utility functions. I also ran some experiments to fine tune the hyperparameters.

6.3 Xingyi Yang

Xingyi Yang completes all the experimental framework. He also trains and conducts the comparison analysis for different loss function and various network topology. He manages all model training and evaluate their performance. In addition, Xingyi finishes the data augmentation method like flip, rotation, random crop.

6.4 Zhenrui Yue

XXXX

References

[1] Liang-Chieh Chen et al. “Rethinking atrous convolution for semantic image segmentation”. In:arXiv preprint arXiv:1706.05587(2017).

[2] Marius Cordts et al. “The cityscapes dataset for semantic urban scene understanding”. In:Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 3213–3223.

[3] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scaleimage recognition”. In:arXiv preprint arXiv:1409.1556(2014).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deepconvolutional neural networks”. In: (2012), pp. 1097–1105.

[5] Kaiming He et al. “Deep residual learning for image recognition”. In: (2016), pp. 770–778.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks forbiomedical image segmentation”. In: (2015), pp. 234–241.

[7] Liang-Chieh Chen et al. “Semantic image segmentation with deep convolutional nets and fullyconnected crfs”. In:arXiv preprint arXiv:1412.7062(2014).

[8] Liang-Chieh Chen et al. “Deeplab: Semantic image segmentation with deep convolutional nets,atrous convolution, and fully connected crfs”. In:IEEE transactions on pattern analysis andmachine intelligence40.4 (2017), pp. 834–84.