Segmentation Using UNet Based Structures

Zhuonan Lin, A53218121

June 13, 2019

A UNet for Image Segmentation

In this section, we will first introduce the three different UNet style structures for the image segmentation, there are basic UNet inspired by [2], UNet with dilated convolution from [3] and UNet with dilation and pyramid pooling [**?**]. Next, we will introduce some important training details, including loss function, optimizer, learning rate, bathc size, etc. Subsequently, we will show our training results such as the accuracy and loss curves, and test phase results including IoU evaluation and prediction examples.

A.1 UNet Structures for Semantic Segmentation

A.1.1 Basic UNet

The basic UNet model we use in this assignment, is directly inspired by the structure proposed in [2]. It includes an encoder-decoder structure where the decoder concatenates the feature map from the encoder and upsamples to the initial image size. The basic UNet structure is given in the *model.py* and we show the structure in Figure 1.

Here, the change of the height and weight of feature map are calculated based on the convolutional and max pooling layer input/output size formula. The upsample size used bilinear can can be directly set to arbitrary size. We can see that the whole structure are very similar to the original UNet structure. The encoder will have a similar structure as ResNet18. In the encoder, a ***ResBlock1*** contains 2 conv. layers both have stride 1, which will keep the output size same as input, while a ***ResBlock2*** has a stride 2 convolutional layer followed by a stried 1 conv. layer, and as a result, the output size will be approximately half of the input, so that the block is served as downsample use. The code of Basic UNet (including *encoder* and *decoder*) is given in *model.py*.

A.1.2 Dilated UNet

The dialted Unet will use dilation convolution in the last 4 Resnet blocks, similar to [3], as shown in Figure 2.

Here, the in the last 4 blocks in encoder, we change ResBlock to DilatedResBlock. In DilatedResBlock, all the convolutional layers have stride 1, and the chance of size by dilation

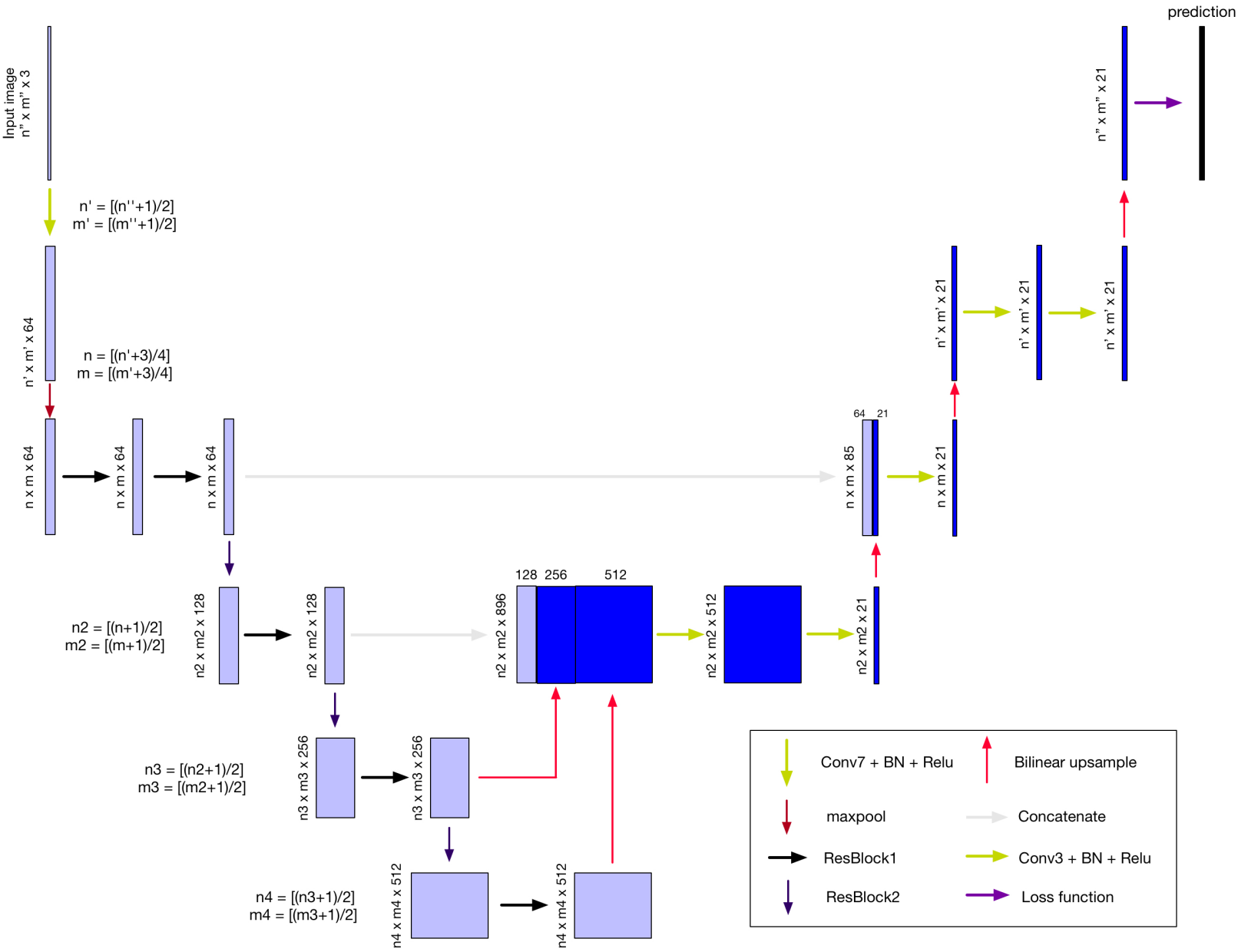

Figure 1: Basic UNet stucture.

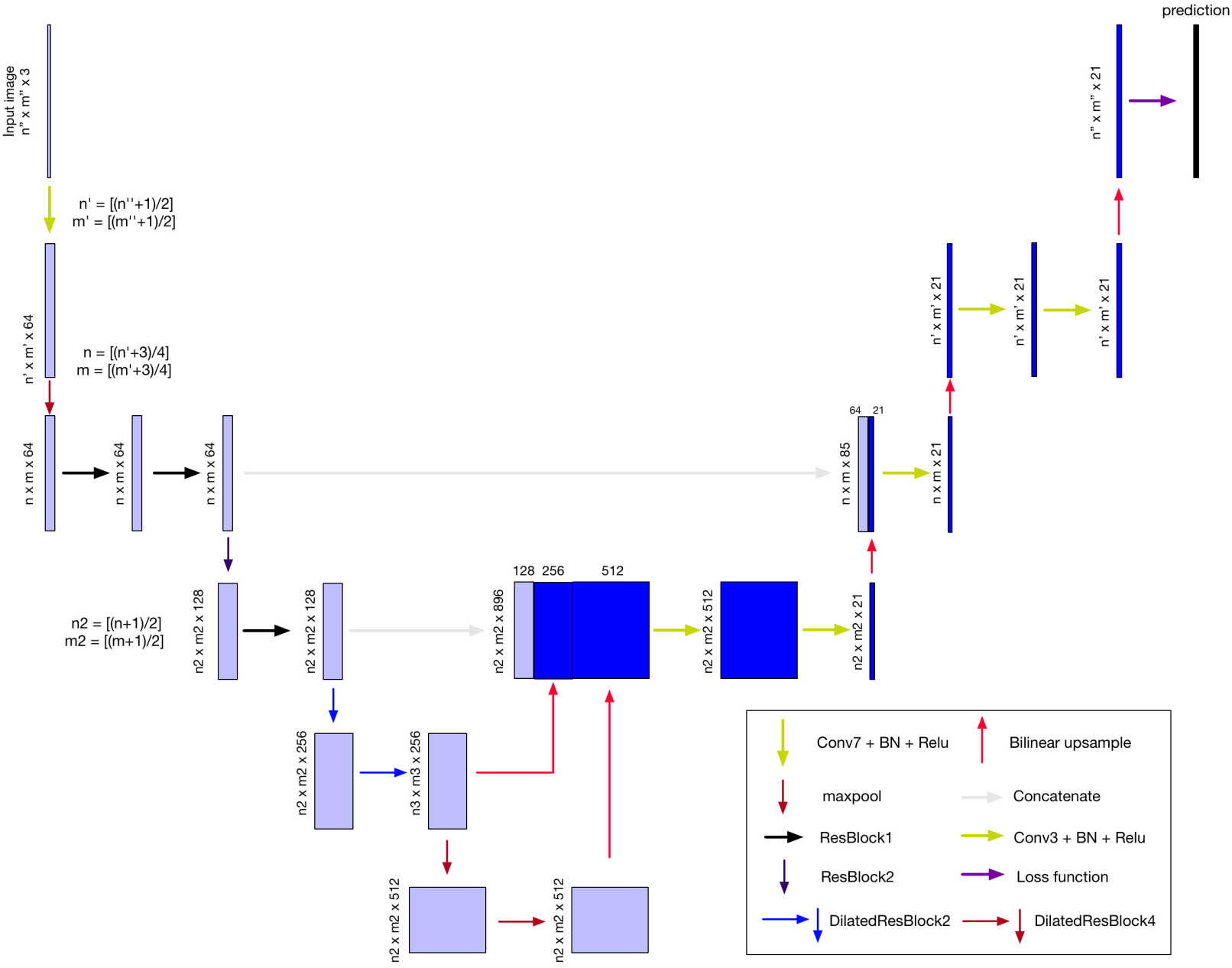

Figure 2: Dilated UNet stucture.

can be canceled by the padding we set, so that every DilatedResBlock will approximately keeps the size of outout same as input. As a reuslt, the final feature map of encoder (*x5*) in Dilated Unet will be 4 times larger than the Basic Unet, where there are 2 ResBlock2 in the last 4 blocks. Here ***DilatedResBlock2*** and ***DilatedResBlock4*** have a dilation of 2 and 4 respecitvely, to achieve differenct size of receptive field. Since DilatedResBlock does not change the channel number of output feature maps, we can keep the decoder same as Basic Unet. The code of Dilated UNet (including *encoderDilation* and *decoderDilation*) can be found in *model.py*.

A.1.3 SPP UNet

Based on Dilated UNet, we can further improve the structure by adding a pyramid pooling block proposed in [4]. The block will be added after the last feature map (*x5*) in the encoder, as shown in Figure 3.

Here, for the pyramid pooling blcok, we use the same one mention in [4], and borrow the figure to illustrate it. The output of pyramid layer (*x6*) will be same size but twice deep (i.e. channel number $\times$ 2) as input. Note that the original input feature map *(x5)* to the pymarid pooling block has been concatenated to the output. Hence in the decoder, we directly use *x6* to replace *x5* in the network for concatenation, and keep the other parts same for a fair comparison to the previous two structures. The code of SPP version of Dilated UNet (including *encoderSPP* and *decoderSPP*) can be found in *model.py*.

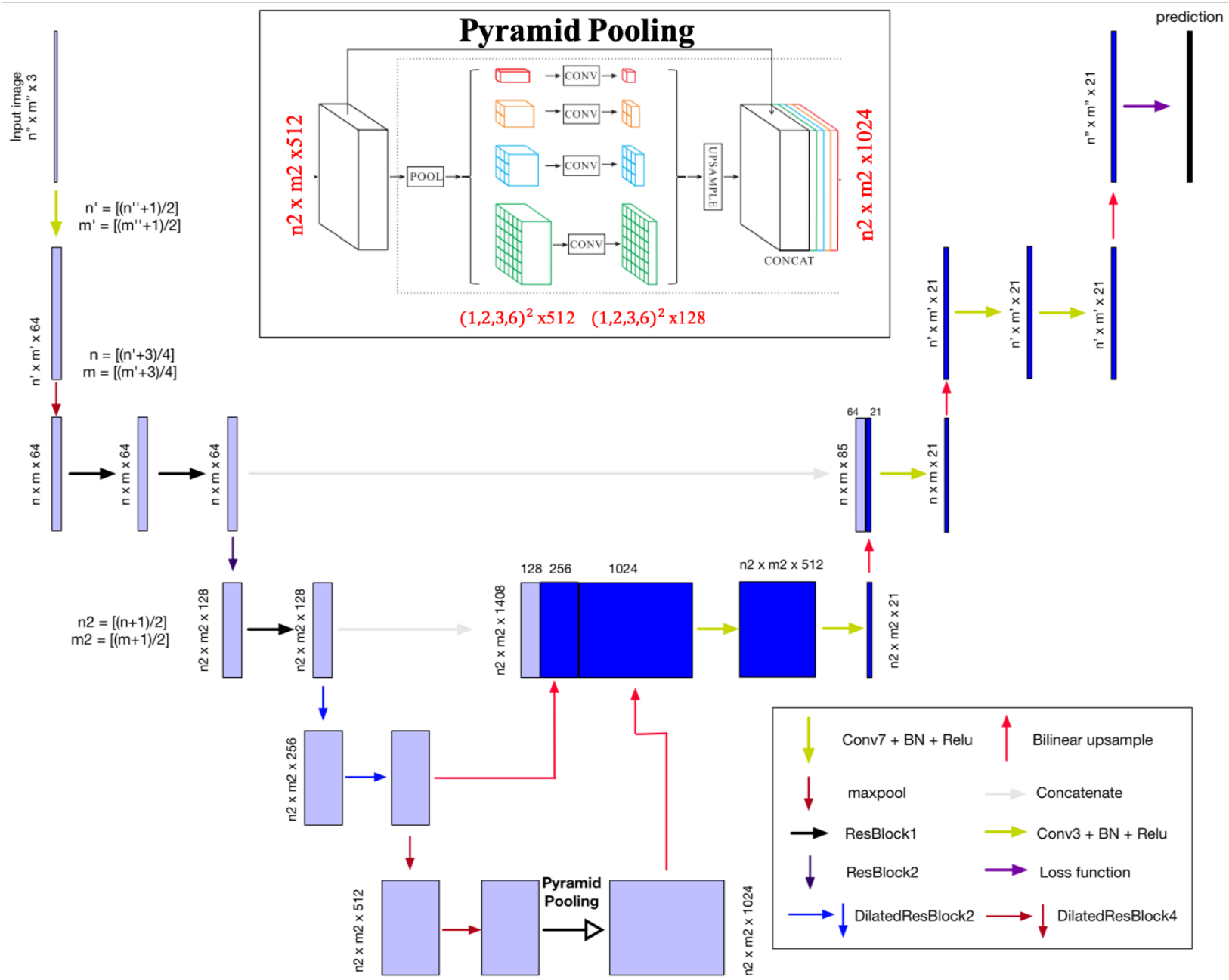

Figure 3: Pyramid pooling block based on Dilated UNet stucture.

A.2 Training Details

Based on the models we built, we can train the 3 networks on VOC12 dataset. For a fair comparison, the training details will be same for all networks, and we use the same training codes (i.e. *train.py*). The training details are listed below.

A.2.1 Loss Function and Decoder Structure

We use a pixel-wise cross-entropy loss function based on the softmax of output feature map, as demostrated in [2]:

$$p_k(\mathbf{x}) = \frac{exp(a_k(\mathbf{x}))}{\sum_{k'=1}^{K} exp(a'_k(\mathbf{x}))} \tag{1}$$

$$Loss = \sum_{x \in \Omega} log(p_{l(\mathbf{x})}(\mathbf{x})) \tag{2}$$

where k denotes the feature channel, $\mathbf{x}$ denotes the pixel position, $a_k(\mathbf{x})$ denotes the activation in the feature map. In the actual codes, we use the mean of the loss function above as an equivalent.

As for the decoder part, we use the given default one without change.

A.2.2 Other Training Details

It turns out that there are other hyper-parameters and strategies are crucial towards a fair accuracy. We summerized the most related ones here as following:

- **Optimizer** We first try SGD with momentum and weight decay. But finally we choose ***Adam*** without weight decay, since it turns out to be faster and more robust.

- **Pretrained Weights** We load ResNet18 weights for every encoder. The Basic UNet encoder has same structure as the ResNet 18 so that it can be directly loaded. For Dilated UNet, though the sturcture changed, the learnable kernels have same parameter shape and number as Unet in encoder, so that it can also be loaded directly. For the SPP UNet, we loaded the parameters from RetNet for the encoder before the pyramid pooling block.

- **Learning Rate** It turns out for a fair accuracy, it is crucial to use different learning rates for encoder and decoder, since the encoder is loaded from pre-trained ResNet18 while the decoder is trained from scratch. After some effort for finding the suitable learning rates, at last we choose **1e-4 as encoder learning rate and 1e-2 for decoder**. Similar to what we do in the Face recognition project, we decrease both the learning rates by 10 times after 800 iterations (the *iterationDecreaseLR* flag in *train.py* can tune this).

- **Batch Size** It turns out that larger batch size can lead to a faster convergence and better training accuracy. However, due to the GPU memory limit and the size of dataset, the batch size cannot be too larger either. We started experiments with 16

as the batch size and ended up with batch size equals to **32**, to balance the resource constraint and training performace.

- **Training Epoches** With all the training details above, it turns out that our accumulated training loss will keep decreasing while the accumulated training accuracy can keep increasing. However, due to the cluster resource limit (6 hours wall time), and more importantly, in order to prevent overfitting (since the dataset is relatively small), we didn't train for a very long time in the final experiments. Here, we choose 24 epoches for all three networks training and compare the results based on this point. This can give a not perfect yer fair enough test accuracy.

- **Traning Data Preprocess** When load the data with the given dataloader, we explicitly set the shuffle flag in PyTorch dataloader to True, and crop the input image to 300×300 size.

A.3 Training Results

We train the 3 networks based on the model and details described above. The accumulated training loss and training accuracy are shown in Figure 4. From Fig. 4, we can see that all the losses are constantly decreasing and accuracy are increasing during the whole training procedure, and we believe the performance can be better if we keep training for some more epoches.

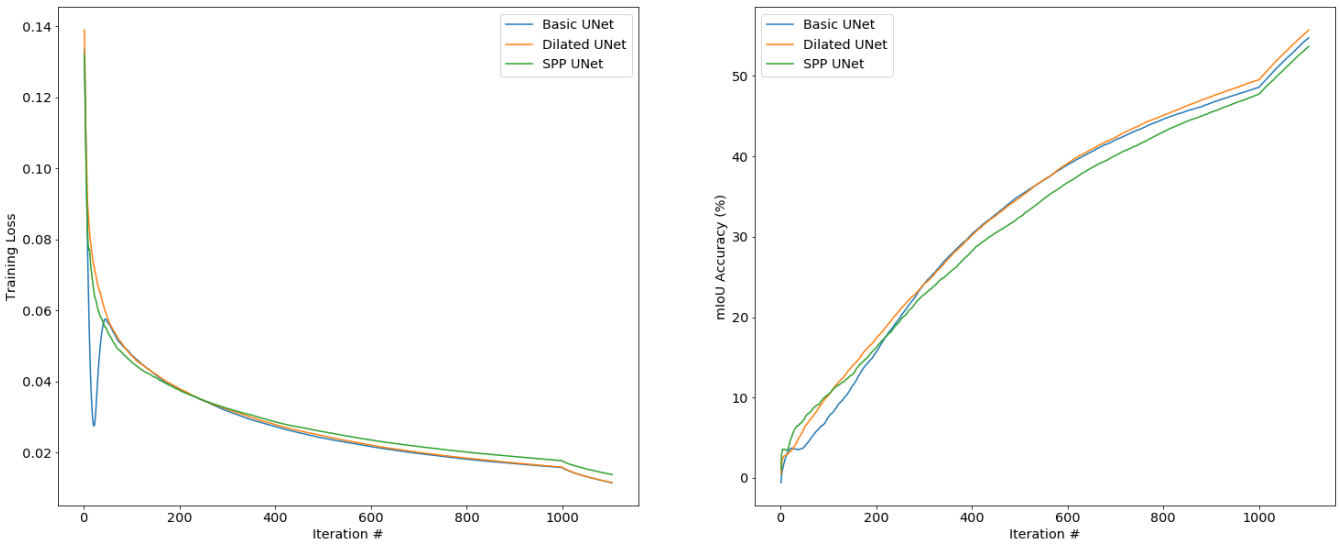

Figure 4: Accumulated training loss (left) and training accuracy (right) of three networks. Quadratic interpolation is used for smoothing curves.

A.4 Test Accuracy

The test IoU accuracy results about the 21 classes and mIoU are summarized in Table 1. We will discuss the results in a later subsection.

Structure	BG	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow
UNet	89.88	69.16	39.57	58.79	47.05	43.68	66.95	65.68	61.44	16.06	41.70
Dilated	89.52	68.50	40.50	57.20	48.54	47.67	66.90	63.83	63.92	12.27	43.41
SPP	87.72	55.23	28.27	49.59	35.62	45.45	70.81	57.36	59.6	16.73	47.35
Dining Table	Dog	Horse	Motorbike	Person	Potted Plant	Sheep	Sofa	Train	TV Monitor	Average	
30.10	49.28	39.81	63.61	71.53	28.49	54.46	29.41	57.34	45.01	50.91	
28.01	54.53	44.46	60.41	69.73	26.70	57.57	28.86	57.44	47.41	51.30	
35.13	46.34	42.93	52.93	66.58	29.46	50.21	20.57	54.67	42.6	47.39	

Table 1: IoU test accuracy of three networks on 21 classes and the mean value.

A.5 Sample Segmentation Output

Here we pick up 4 different input images from the prediction of three networks respectively, ans shown with the original image and ground truth label. Results show in Figure 5.

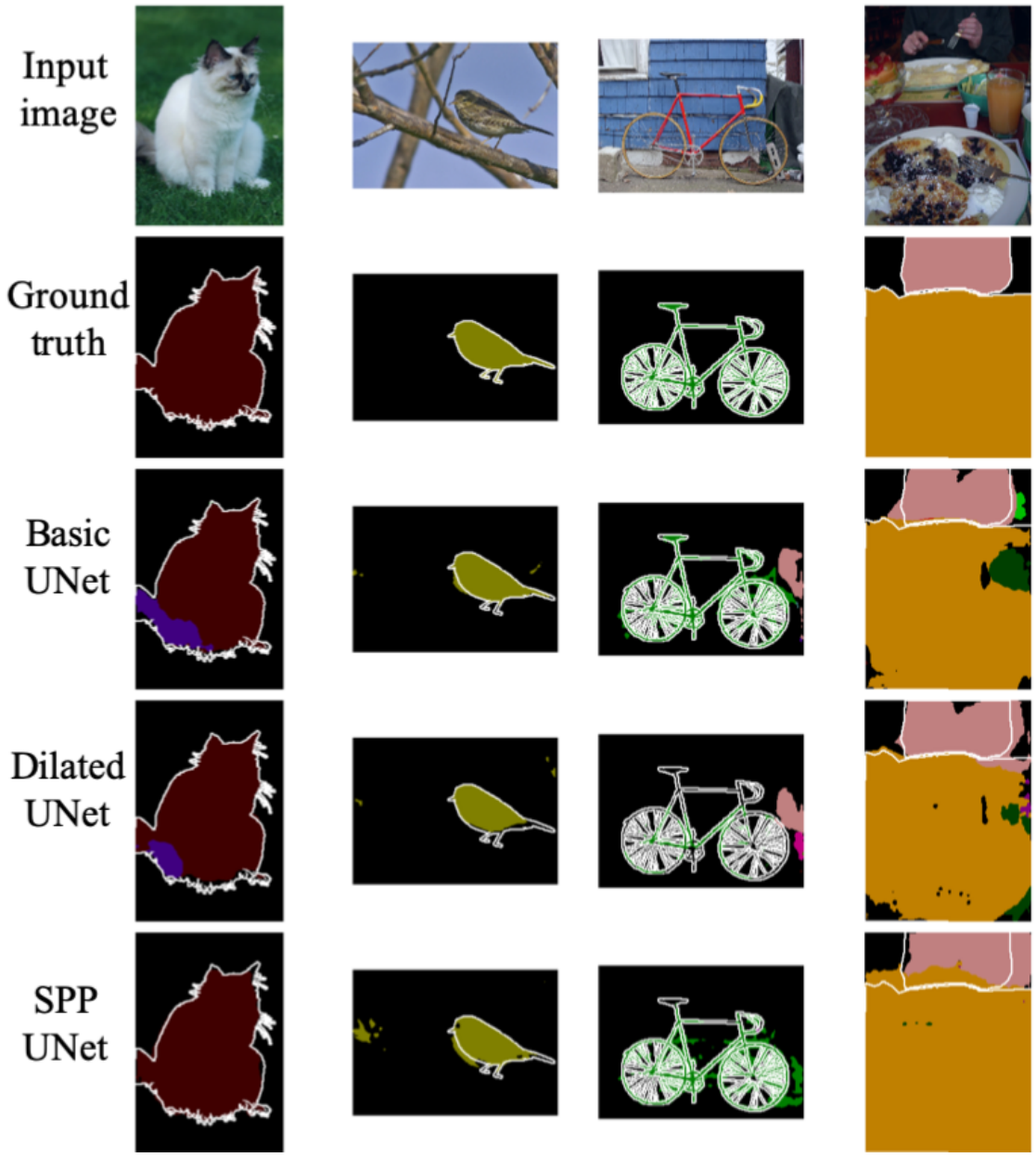

Figure 5: Sample predictions for 4 different input images from three networks.

Here, the four images we choose are showing pretty good results. For different network structures, some observation can be found:

- **Dilated UNet is better than the UNet.** This agree with the training loss and training accuracy we observe from Fig. 4 and test accuracy we observe in Tab. 1. The

reason the dilated network can be better, as we discussed in the lecture, is that it can have larger receptive field compared to normal convolution. Also we didn't observe the griding artifacts in dilated UNet.

- **SPP net can be better than the previous two models**, especially on some specific classes like cow. In Fig. 5 we see that for the cat image, only SPP UNet can predict the whole cat.

- **SPP UNet more likely to predict same category for nearby region**. We can see from Fig. 5, SPP UNet tends to predict the whole large region with same category, like the whole cat or the whole region as dining table. For the mislabeled background in the third image, SPP UNet is also tending to mid-predict the wrong background pixels as bicycle, which is the main object of the image. This can be resulted from that pyramid pooling can take more information from the context. And this can be much more reasonable in many case (e.g. predict a whole cat to be cat, but not half-cat-half-cow in the image above).

- **SPP can have problem.** From the test accuracy in Tab. 1, we can see that the overall average mIoU for SPP is the lowest among the three. Besides the training reason (we compared three with same training iteration while SPP has slightly more training-from-scratch parameters), one other thing we observed from the prediction (though not included in Fig. 5), is that **SPP can sometimes mistakenly predict a whole object to a wrong category**, e.g. a small dog in the image can be predicted as a whole cat, or a whole sheep be predicted to be whole cow category. This can significantly reduce the test accuracy. But we believe by some proper adjustment, if we can fix this whole object missing error, the SPP UNet can outperform the others, and this weak point which decrease the test mIoU can actually be fixed and be a strong point as we discuss above.

A.6 Discussion

In this part, we designed and trained three different UNet structures for segmentation. After careful design and some experiments, we finally successfully train the nets and get fairly good results. To conclude this part, here we want to discuss some choices/steps that help or are supposed to be helpful in this project:

1. Choices that improve the accuracy

    - **Advanced network structure** Just as one cloud assume, that a more advanced network structure can have better accuracy, e.g. dilated UNet outperforms basic UNet. And from the discussion in Sec A.5, we can see that by a proper adjustment, SPP UNet is also very promising to outperform dilated UNet.
    - **Take advantage of dilation convolution and pyramid pooling.** Dilation convolution can achieve larger receptive field with same number parameter, and pyramid pooling can effectively use more context information. These are the key

reasons that the latter two structures are more advanced and supposed to have better performance.

- **Appropriate training strategy** It turns out the the training strategy is crutial to get everything work. We first tried use single *optimizer* for training and the training loss/accuracy was always stuck, though it was still very slowly decreasing/increasing. It turns out that using different learning rate for *encoder* and *decoder* is crucial in this project. We believe we can get better accuracy with same network structure but more proper training strategy if time allows.

2. Strategies for further improvement

   - **Data Argumentation** The VOC2012 training dataset is pretty small with only 1464 images total. To overcome overfitting and for a better result, one thing will definitel help is data argumentation, such as flipping the images or chaning the intensity
   - **Training longer** With data argumentation, we can consider the overfitting problem can be suprressed, so that one other thing we assume to be helpful is then train longer. Actually our current nets is still reducing loss and increasing accuracy but we implement an early stop because we are afraid of overfitting.

B SSD Object Detection

B.1 Evaluation on VOC Dataset

In this section, we evaluate the pre-trained SSD [5] model on the PASCAL VOC 2012 dataset for object detection job. By running the *eval.py*, the average precision (AP) results for each category are evaluated as shown in Table 2, note that here in the evaluation, we use the VOC07 average precision metric as elaborated in next subsection.

Category	Aeroplane	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow
AP	0.7818	0.7200	0.6474	0.4651	0.3865	0.7714	0.6855	0.8296	0.4571	0.6483
Dining Table	Dog	Horse	Motorbike	Person	Potted Plant	Sheep	Sofa	Train	TV Monitor	Mean
0.5567	0.7803	0.7255	0.7716	0.7421	0.3492	0.6679	0.5870	0.7815	0.6457	0.6499

Table 2: Average precision (AP) evaluation results for each category of VOC dataset using pre-trained SSD model.

B.2 SSD Properties

B.2.1 Average Precision Calculation

Average precision (AP) used as object detection metric, is computed the average precision value for recall value over 0 to 1. Remember that the precision and recall is defined as:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{3}$$

$$Recall = \frac{True\ Positive}{True\ Positive + \ FalseNegative} \tag{4}$$

The AP metric is used in the evaluation in Sec. B, and it is calculated in *eval.py*: Line 163-191:

```
def do_python_eval(output_dir='output', use_07=True ):
    cachedir = os.path.join(devkit_path, 'annotations_cache')
    aps = []
    # The PASCAL VOC metric changed in 2010
    use_07_metric = use_07
    print('VOC07 metric? ' + ('Yes' if use_07_metric else 'No'))
    if not os.path.isdir(output_dir):
        os.mkdir(output_dir)
    for i, cls in enumerate(labelmap):
        filename = get_voc_results_file_template(set_type, cls)
        rec, prec, ap = voc_eval(
           filename, annopath, imgsetpath.format(set_type), cls,
              ... cachedir,
```

```
13              ovthresh=0.5, use_07_metric=use_07_metric)
14          aps += [ap]
15          print('AP for {} = {:.4f}'.format(cls, ap))
16          with open(os.path.join(output_dir, cls + '_pr.pkl'), 'wb'
                ... ) as f:
17              pickle.dump({'rec': rec, 'prec': prec, 'ap': ap}, f)
18      print('Mean AP = {:.4f}'.format(np.mean(aps)))
19      print('~~~~~~~~')
20      print('Results:')
21      for ap in aps:
22          print('{:.3f}'.format(ap))
23      print('{:.3f}'.format(np.mean(aps)))
24      print('~~~~~~~~')
25      print('')
26      print('
            ... --------------------------------------------------------------
            ... ')
27      print('Results computed with the **unofficial** Python eval
            ... code.')
28      print('Results should be very close to the official MATLAB
            ... eval code.')
29      print('
            ... --------------------------------------------------------------
            ... ')
```

In the snippet above, besides the read in and output codes that are irrelevant to AP calculation, AP for each category is calculated in the function *voc_eval()* in *eval.py* Line: 228-361. We do not include the entire function here since it has too many lines.

In the function *voc_eval()*, it first reads in the ground truth and predicted data respectively for calculate AP. After doing so, the precision, recall and AP are calcualted in the following steps:

1. Sort the bounding boxes (BB) for prediction with the confidence;
2. Loop through the prediction dataset and mark true positives (TPs) and false positives (FPs), and hence the False Negative (FNs);
3. Compute precision and recall using the definition equations above;
4. Calculate AP using the precision and recall above, by the function *voc_ap()* (*eval.py* Line 194-225). Here, the function provides two different metrics correspond to PASCAL VOC metric before and after 2010, which use 11 point metric and an accumulated method, respectively. The specific method to use depends on the input argument *use_07_metric*. In the actual evaluation above, we used VOC07 metric, it took 11 recall values between 0 and 1 and calculate the average precision of these 11 recall values as defined;

5. Return precision (*prec*), recall (*rec*), can AP (*ap*).

B.2.2 Speed Improvement of SSD

SSD has a significant improvement in spedd rate, which can be 59FPS in test stage on a Titan X graphic card. Compared to Faster-RCNN, the fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature stage.

In region-based convolutional neural network (R-CNN) objection detection method, region proposals need to be first extracted from the image sample as candidates for further CNN process, which is the bottleneck for real-time running for this type of method. Although Faster-RCNN has accelerated this region proposal pocess by replacing the conventional selective search, which greedily merges superpixels based on engineered low-level features, by a small end-to-end CNN, namely region proposal network (RPN), to reduce the time, it is still not fast enough for real time application (5FPS on a GPU). **SSD**, on the other hand, as the name suggests, is a single shot method that completely eliminates proposal generation and subsequent pixel oor feature re-sampling stages and encapsulates all computation in a single network. This is achieved by discretizing the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At test time, the network generates scores for the presence of each object category in each defualt box and produces adjustments to the box to better match the object shape. Consequently, this single shot method without any region proposal related part in the network, result in a simple network and can be much faster compare to Faster-RCNN

B.2.3 Handling Negative Bounding Box

Both in Faster-RCNN and SSD method, there are multiple bounding boxes centered at each cell in feature maps, namely anchors in Faster-RCNN and default boxes in SSD. Consequently, the negative samples those bounding boxes are much more than positive samples (actual object bounding box), which will bias the training towards negative samples. To handle this, Faster R-CNN and SSD use different schemes. **In Faster R-CNN**, it randomly chooses 256 anchors in an image sample to pass through the network to compute the loss function. Among these anchors, the samples negative and positive anchors have a ratio upto 1:1 to achieve balance. And if there are not enough 128 positive samples in an image, it will pad the mini-batch with *negative* ones. On the other hand, **in SSD**, it uses a non-maximum suppression to achieve the hard negative mining. Here, instead of using all the negative examples, it sorts negative boxes using the highest confidence loss (calculated from SSD network) for each box and pick the top ones so that the ratio between the negative and positive is at most 3:1. This method is proved to lead to faster optimization and a more stable training.

B.3 Example Objection Detection Results from SSD

In this part, we will show the results from the test phase of SSD, namely the objection detection prediction results for some randomly chosen images from different sources of dataset.

Here, we show two randomly chosen images per dataset from PASCAL VOC2007, PASCAL VOC2012, and COCO, respectively. All the images are chosen from the validation part in the dataset. The reason that we choose from these four dataset is that there are mentioned in [5]. To do this, we first run the Linux script in *data/scripts/* to download and pre-process on the dataset, and then change the root path for each dataset accordingly in *demo.ipynb* to feed forward through SSD. We show the final results with a confidence threshold equals to 0.6. The results are shown in Figure 6. The result are pretty good on the examples. Note that for COCO dataset, since the pre-trained net on VOC2012 dataset only has 20 categories, some images have instances cannot cover by these categories.

Figure 6: SSD objection detection results of some sample images. Sample images randomly chosen from VOC2012 (first row), VOC2007 (second row) and COCO dataset (third row).

References

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascalnetwork.org/challenges/VOC/voc2012/workshop/index.html.

[2] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234-241. Springer, Cham, 2015.

[3] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 472–480, 2017.

[4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.