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

**Hao XiangHuimin Zeng**

Computer Science & Engineering Computer Science & Engineering

UC San Diego UC San Diego

La Jolla, CA 92093 La Jolla, CA 92093

*haxiang@ucsd.edu zenghuimin@yahoo.com*

**Xingyi YangZhenrui Yue**

Computer Science & Engineering Computer Science & Engineering

UC San Diego UC San Diego

La Jolla, CA 92093 La Jolla, CA 92093

*x3yang@ucsd.edu 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, we first described the provided dataset: Cityscapes Dataset [1], which contains street view images from over 50 cities in Europe with labeling data from 30 classes for each pixel, such as road, person, vehicle etc. Then, we implemented a baseline convolutional neural network (CNN) of five convolutional layers and five deconvolutional layers with perform the sematic segmentation task. We utilized softmax function for each pixel as the output layer and cross-entropy loss to be the objective function, and finished the data loader, performance indicator and the training function. The baseline model achieved XXX accuracy on this dataset after XXX epochs of training, later, we created a few more advanced CNN models with different network structure to achieve a better segmentation performance, these models were trained, tested and their performances were presented in this report visually. We tested these models on transformed images, imbalanced images and applied optimizations on the network structures to find the most suitable CNN networks for this assignment. One of our models with XXX achieved the best overall accuracy and has XXX in XXX as well as XXX in XXX. In the last section, individual contributions to this programming assignment was briefly described.

**1 Introduction**

Cityscapes Dataset is an urban visual understanding dataset based on over 50 cities in Europe. 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:

This considers the input number and the output number 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:

Where represents the batch mean, the standard deviation of the batch and 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 publishments 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 Deep Residual Network (ResNet)**

In this part, we introduced a residual learning convolutional neural network by He et al. [[2]](https://arxiv.org/pdf/1512.03385.pdf), 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 must be made to match to output dimension, this is shown in the following formula, where represents the residual layer and the projection matrix in case of dimension change [2].

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 [3], 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 [2].

图片包含 文字, 地图

描述已自动生成

Figure 1: Model Structure and Evaluation Results of ResNets [2]

**2.2 Convolutional Networks for Segmentation (U-Net)**

This part aims to introduce a specific convolutional network for biomedical image segmentation by Ronneberger et al. [[4]](https://arxiv.org/pdf/1505.04597.pdf), 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 [4].

图片包含 屏幕截图

描述已自动生成

Figure 2: U-Net Structure [4]

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.

**2.3 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 [[5]](https://arxiv.org/pdf/1409.1556.pdf). 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. [[6]](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf), 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.

图片包含 文字, 地图

描述已自动生成

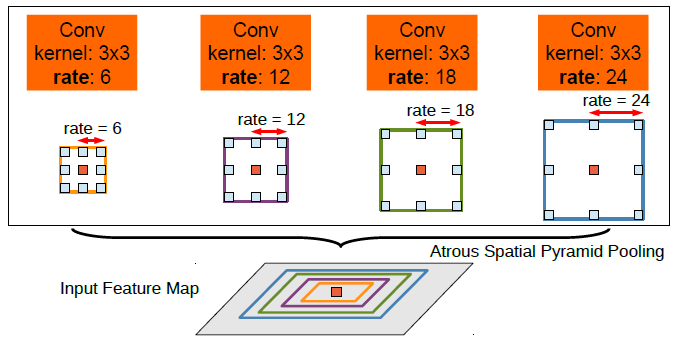
Figure 3: VGG-16 and VGG-19 Structure [5]

**2.4 DeepLab**

DeepLabv3 is a deep convolutional neural network (DCNN) proposed by Chen et al. [[7]](https://arxiv.org/pdf/1706.05587.pdf) in the year of 2017, before DeepLabv3, there were also DeepLabv1 [[8]](https://arxiv.org/pdf/1412.7062v3.pdf) and DeepLabv2 [[9]](https://arxiv.org/pdf/1606.00915.pdf) 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.

图片包含 物体

描述已自动生成

图片包含 屏幕截图

描述已自动生成

Figure 4: Atrous Convolution, Atrous Spatial Pyramid Pooling and DeepLabv3 [7][8][9]

**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.

|  |  |
| --- | --- |
| **Layer** | **Baseline Configuration** |
| 1 | Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, stride=2, padding=1, dilation=1), BatchNorm2d(num\_features=32), ReLU |
| 2 | Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=2, padding=1, dilation=1), BatchNorm2d(num\_features=64), ReLU |
| 3 | Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=2, padding=1, dilation=1), BatchNorm2d(num\_features=128), ReLU |
| 4 | Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3, stride=2, padding=1, dilation=1), BatchNorm2d(num\_features=256), ReLU |
| 5 | Conv2d(in\_channels=256, out\_channels=512, kernel\_size=3, stride=2, padding=1, dilation=1), BatchNorm2d(num\_features=512), ReLU |
| 6 | ConvTranspose2d(in\_channels=512, out\_channels=512, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=512), ReLU |
| 7 | ConvTranspose2d(in\_channels=512, out\_channels=256, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=256), ReLU |
| 8 | ConvTranspose2d(in\_channels=256, out\_channels=128, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=128), ReLU |
| 9 | ConvTranspose2d(in\_channels=128, out\_channels=64, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=64), ReLU |
| 10 | ConvTranspose2d(in\_channels=64, out\_channels=32, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=32), ReLU |
| classifier | 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.

|  |  |
| --- | --- |
| **Layer** | **Modified U-Net Configuration** |
| 1 | Conv2d(in\_channels=3, out\_channels=64, kernel\_size=3, padding=1), ReLU, Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, padding=1), BatchNorm2d(num\_features=64), ReLU |
| 2 | MaxPool2d(kernel\_size=2), Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, padding=1), BatchNorm2d(num\_features=128), ReLU, Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, padding=1), BatchNorm2d(num\_features=128), ReLU |
| 3 | MaxPool2d(kernel\_size=2), Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3, padding=1), BatchNorm2d(num\_features=256), ReLU, Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, padding=1), BatchNorm2d(num\_features=256), ReLU |
| 4 | MaxPool2d(kernel\_size=2), Conv2d(in\_channels=256, out\_channels=512, kernel\_size=3, padding=1), BatchNorm2d(num\_features=512), ReLU, Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, padding=1), BatchNorm2d(num\_features=512), ReLU |
| 5 | MaxPool2d(kernel\_size=2), Conv2d(in\_channels=512, out\_channels=1024, kernel\_size=3, padding=1), BatchNorm2d(num\_features=1024), ReLU, Conv2d(in\_channels=1024, out\_channels=1024, kernel\_size=3, padding=1), BatchNorm2d(num\_features=1024), ReLU |
| deconv1 | Upsample(scale\_factor=2, mode='bilinear', align\_corners=True), Conv2d(in\_channels=1024, out\_channels=512, kernel\_size=3, padding=1), BatchNorm2d(num\_features=512), ReLU |
| 6 | Conv2d(in\_channels=1024, out\_channels=512, kernel\_size=3, padding=1), BatchNorm2d(num\_features=512), ReLU, Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3, padding=1), BatchNorm2d(num\_features=512), ReLU |
| deconv2 | Upsample(scale\_factor=2, mode='bilinear', align\_corners=True), Conv2d(in\_channels=512, out\_channels=256, kernel\_size=3, padding=1), BatchNorm2d(num\_features=256), ReLU |
| 7 | Conv2d(in\_channels=512, out\_channels=256, kernel\_size=3, padding=1), BatchNorm2d(num\_features=256), ReLU, Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3, padding=1), BatchNorm2d(num\_features=256), ReLU |
| deconv3 | Upsample(scale\_factor=2, mode='bilinear', align\_corners=True), Conv2d(in\_channels=256, out\_channels=128, kernel\_size=3, padding=1), BatchNorm2d(num\_features=128), ReLU |
| 8 | Conv2d(in\_channels=256, out\_channels=128, kernel\_size=3, padding=1), BatchNorm2d(num\_features=128), ReLU, Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3, padding=1), BatchNorm2d(num\_features=128), ReLU |
| deconv4 | Upsample(scale\_factor=2, mode='bilinear', align\_corners=True), Conv2d(in\_channels=128, out\_channels=64, kernel\_size=3, padding=1), BatchNorm2d(num\_features=64), ReLU |
| 9 | Conv2d(in\_channels=128, out\_channels=64, kernel\_size=3, padding=1), BatchNorm2d(num\_features=64), ReLU, Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3, padding=1), BatchNorm2d(num\_features=64), ReLU |
| classifier | Conv2d(in\_channels=64, out\_channels=self.n\_class, kernel\_size=1) |

Table 1: 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.

|  |  |
| --- | --- |
| **Layer** | **Modified ResNet-50 Configuration** |
| N/A | Pre-trained ResNet-50 |
| deconv1 | ConvTranspose2d(encoder\_out\_chnnel[backbone], out\_channels=512, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=512), ReLU |
| deconv2 | ConvTranspose2d(channels=512, out\_channels=256, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=256), ReLU |
| deconv3 | ConvTranspose2d(channels=256, out\_channels=128, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=128), ReLU |
| deconv4 | ConvTranspose2d(channels=128, out\_channels=64, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=64), ReLU |
| deconv5 | ConvTranspose2d(channels=64, out\_channels=32, kernel\_size=3, stride=2, padding=1, dilation=1, output\_padding=1), BatchNorm2d(num\_features=32), ReLU |
| classifier | Conv2d(in\_channels=32, out\_channels=self.n\_class, kernel\_size=1) |

Table 2: 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.

|  |  |
| --- | --- |
| **Layer** | **ASPP Model** |
| 1 | Conv2d(in\_channel, out\_channel=128, kernel\_size=(1,1), stride=1, padding=0, dilation=1) |
| 2 | Conv2d(in\_channel, out\_channel=128, kernel\_size=(3,3), stride=1, padding=6, dilation=6), BatchNorm2d(num\_features=128), ReLU |
| 3 | Conv2d(in\_channel, out\_channel=128, kernel\_size=(3,3), stride=1, padding=12, dilation=12), BatchNorm2d(num\_features=128), ReLU |
| 4 | Conv2d(in\_channel, out\_channel=128, kernel\_size=(3,3), stride=1, padding=18, dilation=18), BatchNorm2d(num\_features=128), ReLU |
| 5 | AdaptiveAvgPool2d(output\_size=(1,1)), Conv2d(in\_channel, out\_channel=128, kernel\_size=(1,1), stride=1, padding=0, dilation=1), BatchNorm2d(num\_features=128), ReLU |
| 6 | interpolate(self.pooling(layer5), size=input.shape[2:], mode='bilinear', align\_corners=True) |
| 7 | concatenate(layer1, layer2, layer3, layer4, layer6) |
| 8 | Conv2d(in\_channel=128\*5, out\_channel, kernel\_size=(1,1), stride=1, padding=0, dilation=1), BatchNorm2d(out\_channel), ReLU |

Table 2: 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.

**4 Results**

This part included the training, validation and test results of the models we mentioned in the last section.

**4.1 Baseline Method (Fully Convolutional Network)**

**5 Discussion**

This part included the individual contributions of the two team members Huimin Zeng and Zhenrui Yue in the programming assignment.

**5.1 Huimin Zeng**

I report.

**5.2 Zhenrui Yue**

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**

XXXX

**6.2 Huimin Zeng**

XXXX

**6.3 Xingyi Yang**

XXXX

**6.4 Zhenrui Yue**

XXXX