# Image Analysis Project Report

Weiyao Xie

December $8^{th}$ 2020

## 1 How CNN works

In this section I will explain how CNN works based on my understanding. I would like to break down this section into three parts: CNN Overview, Types of Layers and Parameter Sharing.

### 1.1 CNN Overview

CNN can be viewed as a modification of regular neural networks. Instead of neurons in one layer are fully connected with those in another layer, neurons in CNN are only connected with neurons in the receptive field. Similar to neurons in neural network, each neuron receives some input and perform a dot product and followed by a non-linearity function such as ReLU. In order to use the filter to cover the whole image, we can slide the filter across the image, like what is shown in figure 1. The receptive region is the same size as the filter. Each filter will then combine the output into a feature map, where the activation will be high if a certain feature is found. The main advantage CNN has over Neural network is the computational efficiency. While neural network is good at handling structured data, when it comes to images with thousand of pixel values, it becomes extremely inefficient to train a neural network. We need to sacrifice number of parameters in order to speed up our training process. However, each filter in CNN only focus on a receptive region of the input. It is much faster to train a CNN with deeper structure and more parameters. Thus it is common to assume that the input for CNN is unstructured data with very high dimension, like images and embeddings.

### 1.2 Types of layers

There are three types of layers that are primarily used in CNN: Convolutional layers, Pooling layers and fully connected layers.

#### 1.2.1 Convolutional Layers

Convolutional layers compute output of neurins that are connected to local regions in the input, each computing a dot product between their weights and the small region they are connected. Each convolutional layer contains a set of learnable filters that can be small spatially, and has consistent depth with the input. During the forward pass, CNN slides each filter across the spatial

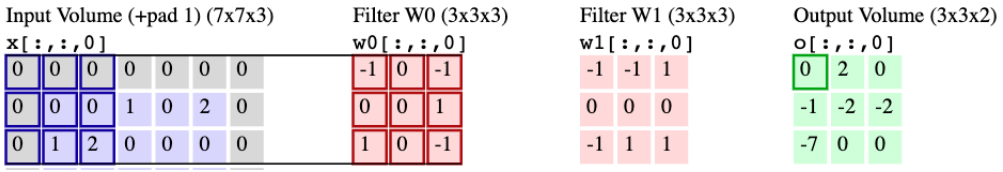


Figure 1: How each filter takes input from a receptive region instead of connected to all inputs. The image is taken from course resources from Stanford CS231n.

dimension of input and get dot product. Each filter will produce a feature map that indicates whether there is any feature captured at certain spatial location. Generally, we say the filters activate when they see certain features.

### 1.2.2 Pooling Layers

There are two major advantages of applying a pooling layers after Convolutional layers. First, pooling layers reduces the output of convolutional layers spatially, which greatly fasten the computation time. Another advantage for using pooling layers is to reduce the probability of overfitting. Using pooling layers only reduces the spatial size but keeps the depth the same. Let W, H, D and K stand for width, height, depth of the input, and number of filters. F, P and S stand for spatial extent(filter size), zero padding and number of strides. The output representation produced by pooling layer has size:

$W\_new = (WF + 2P)/S + 1$

$H\_new = (HF + 2P)/S + 1$

$D\_new = K$

There are two frequently used pooling layers Maxpooling and average pooling. As shown in figure2, this is an example of how a Maxpooling layer reduces the spatial size.

### 1.2.3 Fully Connected layers

After we get the feature maps from CNN, it is very common to implement an extra neural network on top of CNN. When we are constructing the CNN, the final logits we get may have different dimensions from our expected class labels.

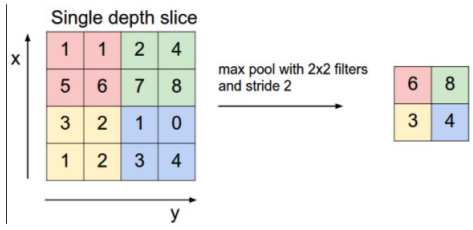


Figure 2: How pooling layer reduce the representation spatially. The image is taken from course resources from Stanford CS231n.

To map the feature/logits into the same dimension with our labels, we can train a neural network. For our problem, we are doing multiclass classification. We need to put the representations produced by CNN into 10 class from 1-10. What I did was to train a two-layer fully connected neural network. The output layer uses softmax activation function an output 10 classes.

# 2 Model Selection and Hyperparameters Tuning

### 2.1.1 Lenet-5

The first CNN architecture I tried is the classic Lenet-5 because it is known for having good performance on MNIST dataset which is a similar task as our task. Since Lenet-5 is proven to be good at classifying digits, I think I should first try this architecture before coming up with my own CNN model. The architecture of Lenet-5 is shown in figure 3.

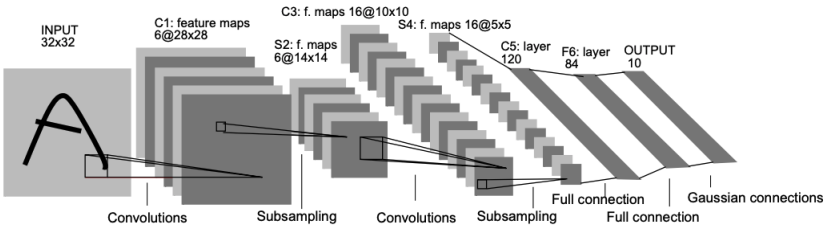


Figure 3: How pooling layer reduce the representation spatially. The image is taken from course resources from the paper linked above.

Then I got the results shown in the following table.

| training loss | training accuracy | validation loss | validation accuracy |
|---|---|---|---|
| 0.1515 | 0.9567 | 0.5319 | 0.8587 |

Due to the space limit. I only displayed the results for my last epoch which is 21st epoch. The details are available in the python file. Basically, there is one distinct pattern I noticed. The training accuracy keeps going up while the validation accuracy stops improving in the middle. I think this is caused by overfitting of the model, so I tried to modify the Lenet-5 by adding some more regularization on it. I add two dropping layers into the architecture. One is right after the last convolutional layer, and the other one is right after the first fully connected layers. Both dropping layers have 0.3 possibility to drop each

neuron. The result I got is displayed below.

As we can see from the table, although the difference between training and

| training loss | training accuracy | validation loss | validation accuracy |
|---|---|---|---|
| 0.5559 | 0.8307 | 0.4594 | 0.8606 |

validation accuracy is much smaller than before, the validation accuracy has not been improved a lot. So I decided to modify the architecture to add more complexity into the model.

### 2.1.1 Modified Model

Based on the observations above, I think it would be a good idea for me to modify the model so that the complexity will be higher thus the model may perform better on our more complicated task. I first added more convolutional layers into my model. The new model now have 8 convolutional layers, which enables the model to learn more features at different levels. The higher number of learnable parameters can also helped the model perform better at our task. I also modified the number of kernels in the convolutional layers. At the lower layers, I chose smaller number of kernels like 16, and then when the CNN went deeper, I gradually increased the number of layers. The reason is that at lower layer, kernels are learning about lower level features like vertical lines and straight lines. We do not need to have so many kernels to capture those features. But when CNN goes deeper, lower level features will combine into more complicated features that help CNN to classify digits. There are much more combinations of features so it will be helpful to have more kernels in deeper layers to capture as many features as possible. At the end of the architecture, I also added another extra fully connect layer to increase the complexity. The architecture is shown below.

### 2.1.2 Hyperparameters Tuning

During the selection process, I experiment with different combinations of hy-

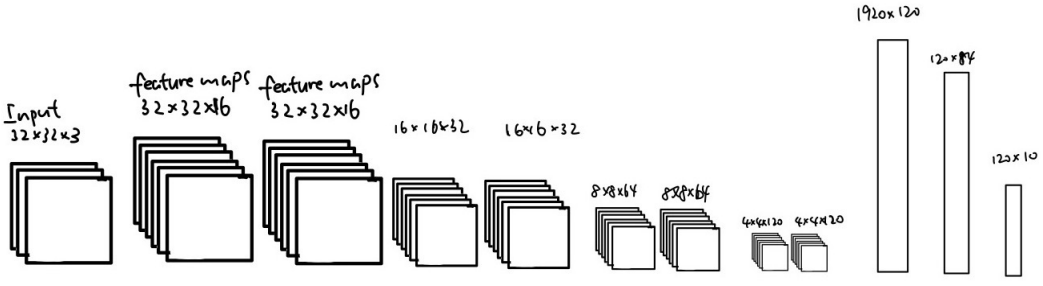


Figure 4: Modified CNN architecture.

perparameters. The hyperparameters I shown in the code are the final ones. First I tried two different kinds of activation function: ReLU and tanh. The activation function eliminates the linearity between each layer so that the work will not be trivial. Then I also tried out the padding option in the convolutional layers. Initially, the parameter I used was "valid" which does not make 0 padding around the input. However, when I tried to build deeper CNN model, I realized if there was no padding, the input size got smaller too fast that I could not add extra layers. Then I used "same" instead of valid which kept the size of feature maps same as the input size. For the similar reason, I also changed the kernel size from 5*5 to 3*3. The most influential hyperparameter I tuned was number of kernels in each layer. The number increases when the model goes deeper because lower level features will be combined into higher level futures in deeper levels.

# 3 Final Results

In this section, I will show that my modified table surpasses all the other models I showed before. Despite the training time of the modified model is much longer than Lenet-5 and regularized Lenet-5, the modified model got much higher validation accuracy than both models. The validation accuracy is around 92%-93% which is higher than the requirement for this project. So I decide to use the modified model as the final version on test data and will no longer change its architecture. The Validation score for all models are shown below. The test

| Model | training accuracy | validation accuracy |
|---|---|---|
| Lenet-5 | 0.956 | 0.858 |
| reg Lenet-5 | 0.831 | 0.861 |
| Modified | 0.961 | 0.929 |

results for modified model are shown in next table.

| Model | test loss | test accuracy |
|---|---|---|
| Modified | 0.27 | 0.931 |