EE569 – HW5

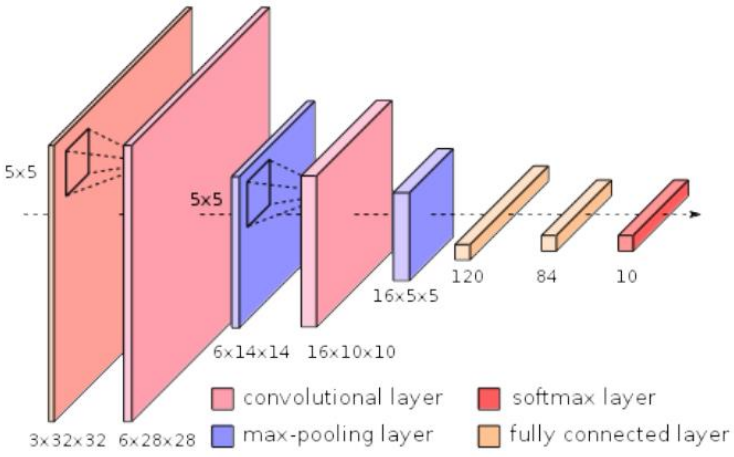
Mozhdeh Rouhsedaghat

2726554211 rouhseda@usc.edu

Abstract and Motivation:

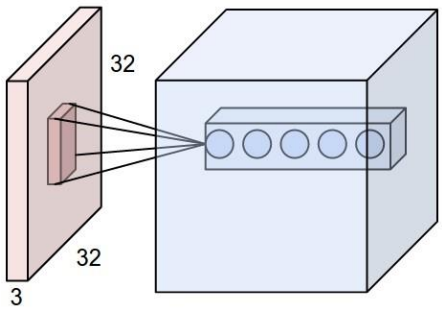
Convolutional Neural Network is currently the state-of-the-art computer vision tool which gives high accuracy in different applications such as image classification, image segmentation, object detection, etc. when enough and proper training data is provided. CNN's algorithm had been introduced some decades ago, but it achieved the current attention and succeed to get a high accuracy in practice from 2012 as a result of availability of the required processing power and training data.

Approach and Procedures:



The above network is the architecture of LeNet-5 which is a simple CNN. It consists of 2 conv layers followed by max pooling layer and three fully connected layers. This simple network can be used for simple applications such as digit classification. I will describe the its components in details below:

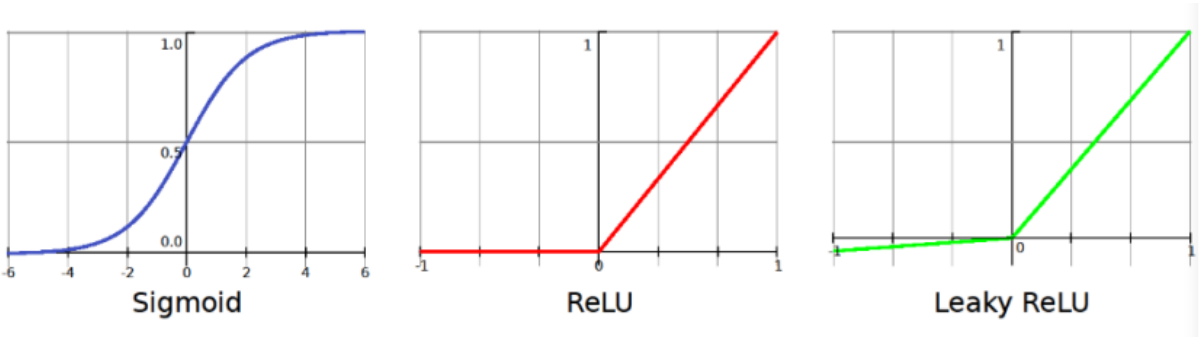
Convolution layer is the result of filtering the previous layer with several filter with the same size and the depth of the previous layer. The number of filters represents the depth of the conv layer and their weights will be determined by backpropagation which I will explain later. Conv layers aim to extract features of the input image.



The conv formula of the filter F with window size k is shown below:

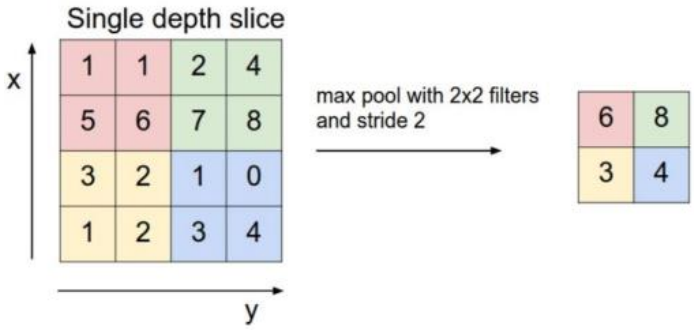
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i+u, j+v]$$

After each conv and fully connected layer we have a nonlinear **activation function**. They increase the nonlinearity of the system. The most common activation functions are shown below:



I have used ReLU in LeNet implementation. Its formula is output=max (0, input)

Max pooling layer chooses the strong features and discards weak ones. It considers nonoverlapping windows with the same size and within this window only keep the max value and discards small values. G=max(f)



Fully connected layer mainly does the classification part of the whole network based on features which are extracted from conv layers. Each neuron in fully connected layer is connected to each output of the previous layer and its weights are determined by backpropagation. ($y= w^Tx+ b$ where w is the filter weight and b is the bias term)

SoftMax layer is used at the last stage to normalize the output into the probability distribution of output classes. The goal is to give a 1 value to the correct label and zero to others, but in practice we won't see a perfect one hot vector at the output and the value of each class is the probability for it to be the corresponding class of the input image. The standard soft max function has the below formula:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Loss function measures the difference between the output of the network and the ideal output. We use it to determine the weights of the filters in the network. One approach is to use gradient descent for determining weights. In this approach we find the gradient of the loss function with respect to every parameter and change them in the negative direction of the gradient by learning

rate*gradient value. We continue to do this till the gradient values become near to zero and the weights converge to a specific value.

For this project we first describe the LeNet-5 architecture. Then train the network with 60000 28*28*1 training images of MNIST data set and test the trained network with 1000 test images of MNIST dataset.

Results:

a)

- I described the mentioned components in the previous part.

-Overfitting problem is the situation in which by increasing the training accuracy, test accuracy decreases. It means the model is fitting the training data more than needed. It may be because of simplicity of the model or lack of enough training data or lack of enough variation in the training data.

There are several approaches to solve it, such as data augmentation, regularization, dropout, early stopping. Data augmentation is to create new data in addition to the training data by modifying it to increase the num of training data and its diversity. Regularization is to add an extra term to the loss function to prevent weights to increase free to completely fit the training data. Dropout is to disable some weights in each layer from updating randomly in each back-propagation step in training; in testing all parameters are used. Early stopping is to stop training before convergence, at the point which testing accuracy reaches its max value and before it starts decreasing.

-CNN is like a block box which does both feature selection and classification based of those features as a whole. Its strength lies behind finding representative and strong features by back propagation instead of using some ad hoc and very basic features such as previous methods. As CNN is a supervised learning method and uses a lot of labeled training data to find appropriate parameters which give a prediction of the label with least error. This data driven approach can find strong features while in the previous approaches we only used some simple features like color or grad magnitude. So previous methods can't compete with CNN.

- Loss function measures the difference between the output of the network and the ideal output. We use it to determine the weights of the filters in the network. We have used cross entropy loss function to measure this difference. Its equation is shown below:

$$C = \sum_{x} [y log(y(x)) + (1 - y) \log (1 - y(x))]$$

In which y is the ideal output and y(x) is the output of the network.

This network consists of several connected layers one after another which all have some differentiable functions. We can find the gradient of the loss function with respect to each parameter using the chain role; in a way that first we find the gradient with respect to the

parameters of the last layer, then we propagate it to the previous layer and so on. We can use gradient descent rule to find a local minimum for loss function. This rule consists of several iterations, in each iteration we update all the parameters by changing them in the negative direction of the gradient by “learning rate*gradient of the loss with respect to the parameter” value. We continue to do this till the gradient values become near to zero and the weights converge to a specific value. In other words:

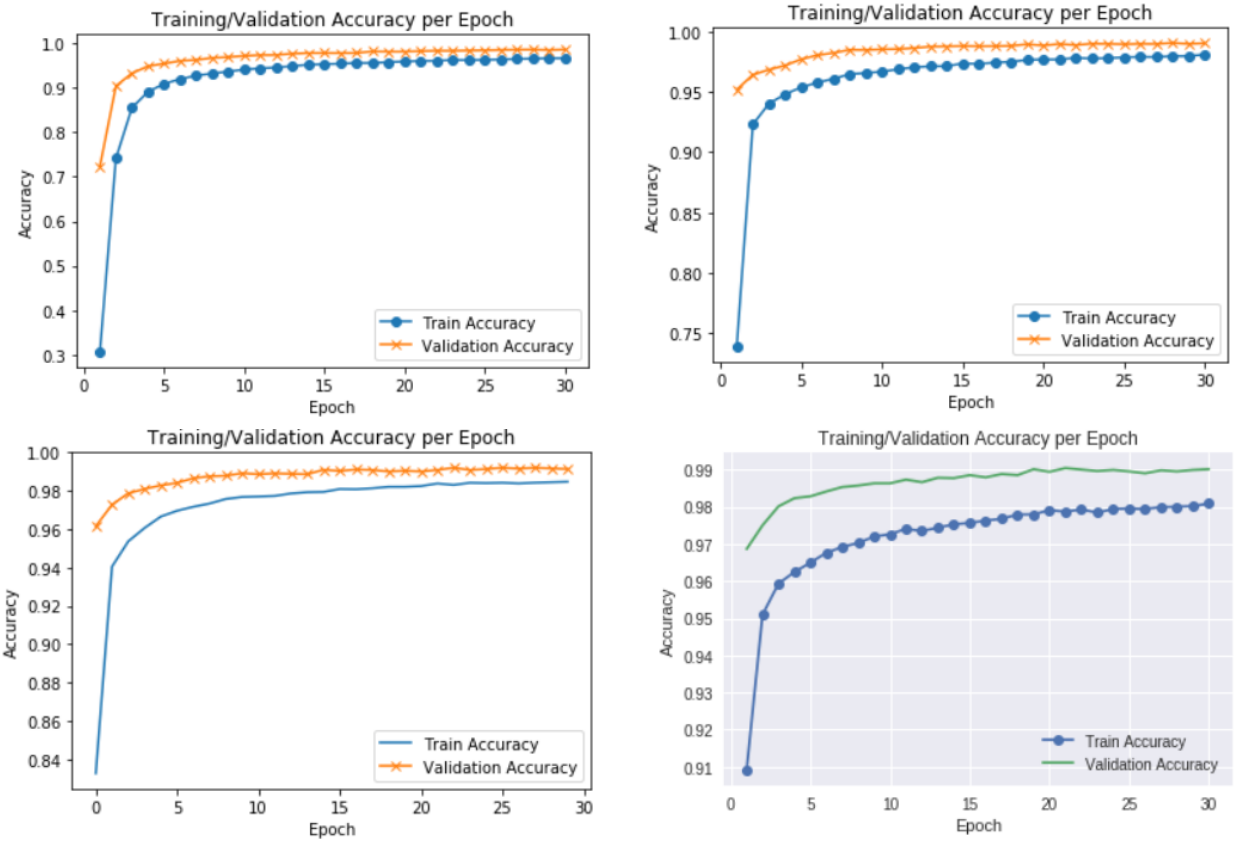
$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

When J is the cost function and a is the learning rate.

b)



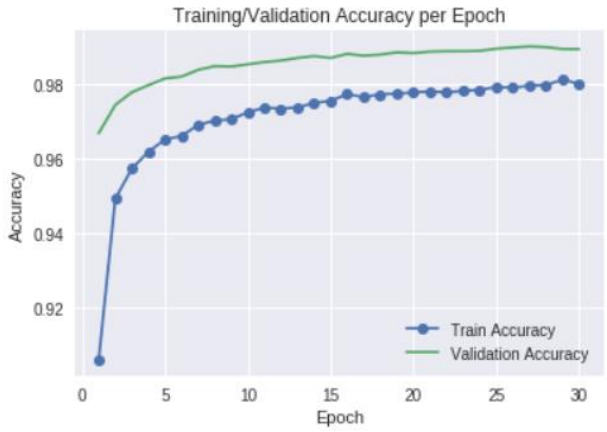


Fig1: accuracy per epoch curve for optimizer=sgd, batch size=150, validation accuracy .9861(upper left), optimizer=sgd, batch size=32, accuracy .9901 (upper right), optimizer=sgd, bach size=16, accuracy .9913 (middle left), optimizer=adagrad, batch size=16, accuracy .9901 (middle right), optimizer=adagrad, batch size=8, accuracy .9891 (lower) for 30 epochs

As you see by decreasing the batch size from 150 to 8 the accuracy increases faster at the beginning, but when batch size gets very small at each step the loss function doesn't necessarily move toward local minimum; it moves in a stochastic way till it converges. According to curves the performance of batch size of 16 is a little better than batch size of 8 and the performance of batch size of 16 is also better than batch size of 32 and 150.

I also tried both sgd and adagrad optimizers for training. Sgd has a smaller convergence rate while Adagrad decreases the loss quickly at first but for small losses (last epochs) it has difficulty to optimize more.

(I have run the blue plot on google colab and the white plots on my pc)

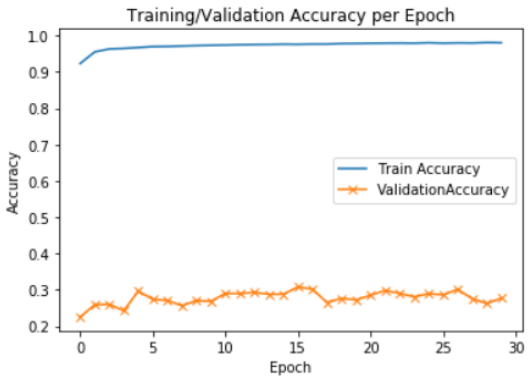
Mean of test accuracies: 0.9893

Std of test accuracies:0.002

The best accuracy was 0.9913 related to sgd with batch size=16

c)

It has a poor performance on negative images. Test accuracy=0.2769. Below you can see the training and test accuracy per epoch:



The reason is that CNN doesn't understand the digits. It can just classify images which are very similar to training data set. In fact, CNN is very dependent to training data.

We can add negative images to training data and shuffle the training data and then train the network using it. In this case CNN will recognize them, as it is trained with similar input images. You can see the related curve below:

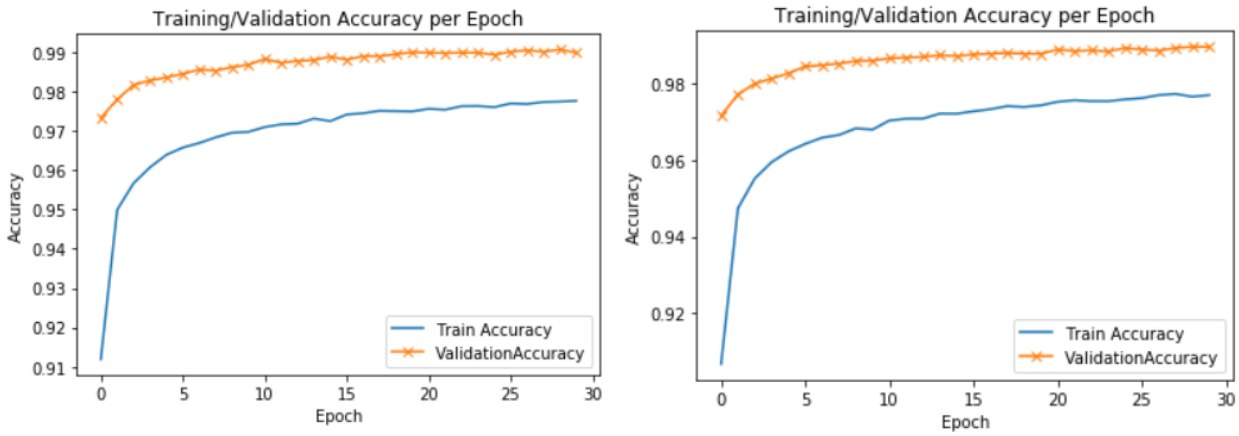


Fig3: accuracy per epoch curve for training data and negative validation data (left) and for original (positive) validation data (right)

As you see now we have an acceptable test accuracy for both negative (0.9899) and positive images (0.9896).

Discussion

I also tried to train the LeNet-5 with a training data which has positive training images at the first half and negative images at the second half (without shuffling). The result was not acceptable. In

fact, when the data is not evenly distributed in terms of input variations and different classes of input, CNN won't be optimized perfectly. In this case by putting negative images at the second half of the training data, on the second half of the training data CNN begins to forget about the first half of the training data and parameters will change to optimize the network only for the negative images. Below you can see the training and validation accuracy per epoch for this case: