HOMEWORK #5

University of Southern California

Lecturer: C.-C.JayKuo Issued: 3/19/2019 Due:4/7/2019

Report

1 CNN Training and Its Application to the MNIST Dataset

1.1 CNN Architecture and Training

1.1.1 Abstract and Motivation

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

1.1.2 Approach and Procedures

1. Describe CNN components

   (a) The fully connected layer
   The fully connected layer is connecting every neuron in one layer to every neuron in the former and latter layers. The last fully-connected layer uses a softmax activation function and generates the output vector that the entries ranges in $[0, 1]$. It represents the probability that the input image belongs to each class. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

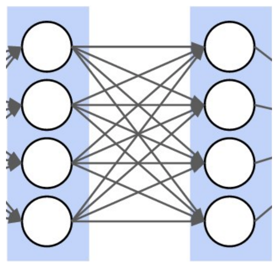


Figure 1: Fully Connected Layer

   (b) The convolutional layer
   The aim of a convolutional layer is to extract features of the input volume. The convolutional layer is connected with the input layer. We are going to apply matrix dot products between a **receptive field** and a filter on all the dimensions. Then we slide the filter with a stride and compute the dot products between the filter and all the new receptive field. **Filter** is a matrix used for dot product. It has the same volume as the receptive field.

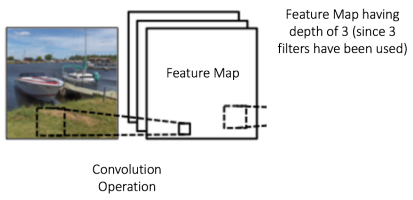

Feature Map having depth of 3 (since 3 filters have been used)
Feature Map
Convolution Operation

Figure 2: Covolution Layer

(c) The max pooling layer
Max pooling layer extract the maximum of the receptive field and reduce the spatial as well as the computing complexity of the model. The max pooling layer is usually a 2x2 filter and stride = 2 without parameters. It takes the maximum among the four entries.

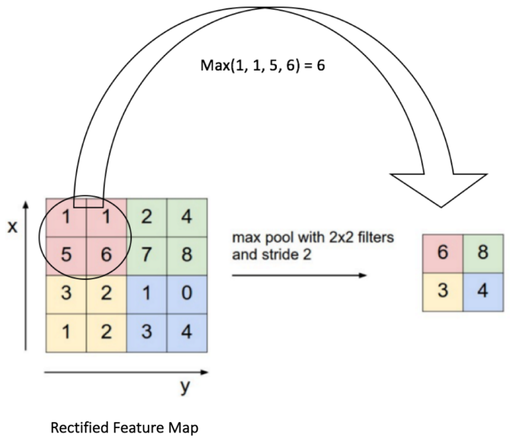


Figure 3: Max Pooling

(d) The activation function
The output of the matrix dot-product value obtained by the filter varies in $\mathbb{R}$, which is not bounded. The dot-product process is all linear and it has no real effect. The function's derivative of the model is constant. Thus, we decided to add "activation functions" for this purpose. This function is to make a judgment about whether to activate the neuron or not. Nowaday the popular active fuction is Rectified Linear Unit (ReLU). ReLU is

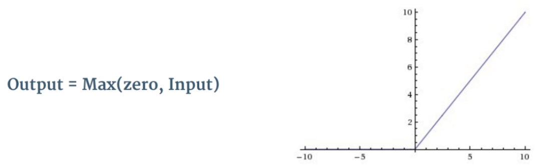


Figure 4: ReLU

an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.

(e) The softmax function
In general, the softmax function is defined as:

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{1}$$

$\boldsymbol{z}$ is a k-dimension normalized probability vector. It returns a value close to 0 or 1. It is desirable for a classifier model to learn parameters and helps training converge more quickly. Softmax is implemented through a neural network layer just before the output layer. The Softmax layer must have the same number of nodes as the output layer.

2. Over-fitting issue
Overfitting happens when your model fits too well to the training set. Then it becomes less general and can not fit the data that does not belong to the training set very well. Your training accuracy will be higher than the accuracy on the validation/test set. CNN is easy to

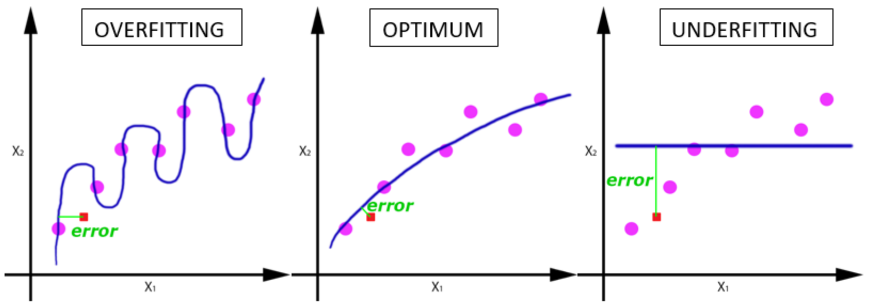


Figure 5: Underfitting To Optimal To Overfitting Curve

overfitting because the model contains so many parameters. There are several ways to reduce overfitting:

(a) Data augmentation. Exploit prior knowledge to add more training data.
(b) Add regularization such as $L1$ *norm* and $L2$ *norm*.
(c) Use randomly dropout. Randomly ignoring a certain percentage of neurons during the training process. It is a very efficient method and also makes training faster.
(d) Early stopping. Stop training when the performance on validation set stops improving.

3. Why CNNs work better
First of all, CNN model has much more parameters than the traditional filter. For instance, the non-local mean filter only has limit parameters to train while CNN has a large amount parameters and hyperparameters. Thus, the output of CNN model is more accurate.

Besides, CNN has the capability to handle different input image and train a general output while the traditional filter need to tune the parameter for each specific input image.

Generally speaking, CNN model exploit feature locality, therefore being able to model higher frequency features. The convolution filters pick out the spatial invariant features, like edges and ripples. These features are quantified in max pooling layer after each convolution.

As the image classification problem, CNNs can automatically extract feature from the image. Other methods, like SVM or knn, they are not as powerful as CNN. Thus, their learn less while comparing to CNN. Besides, they cannot extract feature from the image automatically and need to label the image bu human interference. Before CNN, we need to spend so much time on feature selection. Thus, CNN become the most popular tool for image classification.

Actually, the Universal Approximation Theorem (Cybenko, 89; Hornik, 91) indicates that a feedforward neural net with a single hidden layer can approximate any continuous functions (It might need a huge number of neurons though).

4. Explain the loss function and backpropagation optimization procedure
For a neural network, the loss function is define as:

$$\varepsilon(\boldsymbol{W}_1, ..., \boldsymbol{W}_L) = \sum_{n=1}^{N} \varepsilon_n(\boldsymbol{W}_1, ..., \boldsymbol{W}_L) \tag{2}$$

Where

$$\varepsilon_n(\boldsymbol{W}_1, ..., \boldsymbol{W}_L) = \begin{cases} \|f(\boldsymbol{x}_n) - \boldsymbol{y}_n\|_2^2 & for\ regression \\ \ln\left(1 + \sum_{k \neq y_n} e^{f(\boldsymbol{x}_n)_k - f(\boldsymbol{x}_n)_{y_n}}\right) & for\ classification \end{cases} \tag{3}$$

Where

$$f(x) = h_L(\boldsymbol{W}_L h_{L-1}(\boldsymbol{W}_{L-1} ... h_1(W_1 x))) \tag{4}$$

Where $\boldsymbol{W} \in \mathbb{R}^{D_l \times D_{l-1}}$, $D_1 ... D_l$ are number of neural of each layer. $h : \mathbb{R}^{D_l} \to \mathbb{R}^{D_l}$ is activation functions at layer l.

No matter how complicate the model is, our goal is to minimized the loss fuction. CNN use **SGD** method to optimize this non-convex model. Compute the gradient of this model with chain rules. Here comes the backpropagation algorithm. The main steps of the backpropagation is defined as follws.

For each $l = L, ..., 1$,
compute

$$\frac{\partial \varepsilon_n}{\partial \boldsymbol{a}_l} = \begin{cases} \boldsymbol{W}_{l+1}^T \frac{\partial \varepsilon_n}{\partial \boldsymbol{a}_{l+1}} \circ h'_l(\boldsymbol{a}_l) & if \quad l < L \\ 2(h_L(\boldsymbol{a}_L) - \boldsymbol{y}_n) \circ h'_L(\boldsymbol{a}_L) & else \end{cases} \tag{5}$$

update

$$\boldsymbol{W}_l \leftarrow \boldsymbol{W}_l - \eta \frac{\partial \varepsilon_n}{\partial \boldsymbol{W}_l} \tag{6}$$

Where

$$\boldsymbol{a}_l = \boldsymbol{W}_l \boldsymbol{o}_{l-1} \tag{7}$$

$$\boldsymbol{o}_l = h_l(\boldsymbol{a}_l), \qquad (\boldsymbol{o}_0 = \boldsymbol{x}_n) \tag{8}$$

1.2 Train LeNet-5 on MNIST Dataset

1.2.1 Abstract and Motivation

Yann leCun proposed a convolution neural network called LeNet-5 for handwritten and machine-printed characters recognization. In this section, I use LeNet-5 to train the MNIST dataset.

1.2.2 Approach and Procedures

The LeNet-5 is composed of three convolution layers and two max-pooling layers and two fully-connected layers. The classification is made by a softmax layer. This network was trained on MNIST dataset.

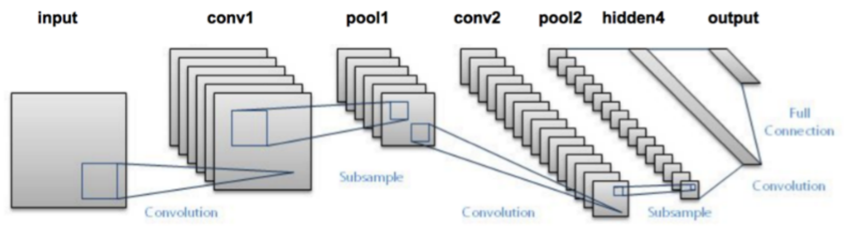


Figure 6: LeNet-5

1. First layer
   The input for LeNet-5 is a 28×28 grayscale image which passes through the first convolutional layer with 6 convolution filters with size 5×5 and a stride of 1. The result is a $28 \times 28 \times 6$ high dimension matrix.

2. Second layer
   Then this high dimension matrix passes through the max-pooling layer with filter size 2×2. Therefore, the dimension of the matrix is reduced to $14 \times 14 \times 6$.

3. Third layer
   Next, there is a second convolutional layer with 16 filters having size 5×5 and a stride of 1. In this layer, only 10 out of 16 filters are connected to 6 filters of the previous layer. The reason is that the number of connections is limited into a reasonable bound as well as break the symmetric of the network. The result matrix is $10 \times 10 \times 16$.

4. Fourth layer
   Again this layer is a max-pooling layer with filter size 2×2. Thus, the output will be reduced to $5 \times 5 \times 16$.

5. Fifth layer
   The fifth layer is a fully connected convolutional layer with 120 filters each of size 5×5. Each filter is connected to all the matrix obtained in the fourth layer. The result matrix is $1 \times 1 \times 120$.

6. Sixth layer
   The sixth layer is a dense layer with 84 filters of size 1×1.

7. Seventh layer
   Finally, there is a fully connected softmax output layer with 10 values range in $[0, 1]$ corresponding to the probability of digits from 0 to 9.

1.2.3 Experimental Result

I try 5 set of parameters and get five result plot. There are learning rate, number of epoch, batch size and the momentum. Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. One Epoch is when an entire dataset is passed forward and backward through the neural network only once. Batch Size is total number of training examples present in a single batch.

1. learning rate = 0.001, epoch = 15, batch size = 128;

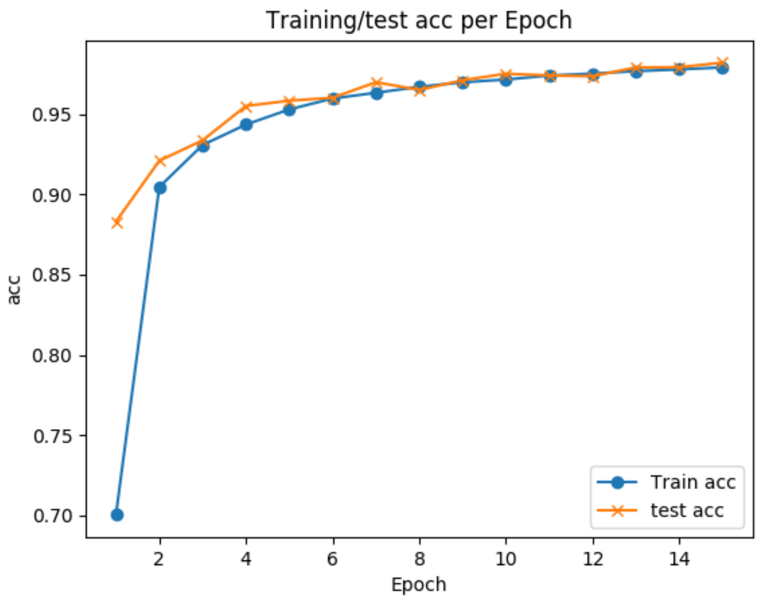


Figure 7: Learning Rate = 0.001, Epoch = 15, Batch Size = 128

The best result is 98.20.

2. learning rate = 0.01, epoch = 15, batch size = 128; The best result is 98.25.
3. learning rate = 0.01, epoch = 20, batch size = 128; The best result is 98.38.
4. learning rate = 0.01, epoch = 20, batch size = 64; The best result is 98.67.
5. learning rate = 0.01, epoch = 20, batch size = 64, momentum = 0.9, Nestrov = True; The best result is 99.23.

1.2.4 Discussion

As we can see, CNN yields a high accuracy on the training and validation dataset. The best result could reach higher than 99 percent accuracy. The mean of the model over the five parameters is 98.5540. The variance is 0.1744. This indicate that the performance of the LeNet-5 is stable and efficient.

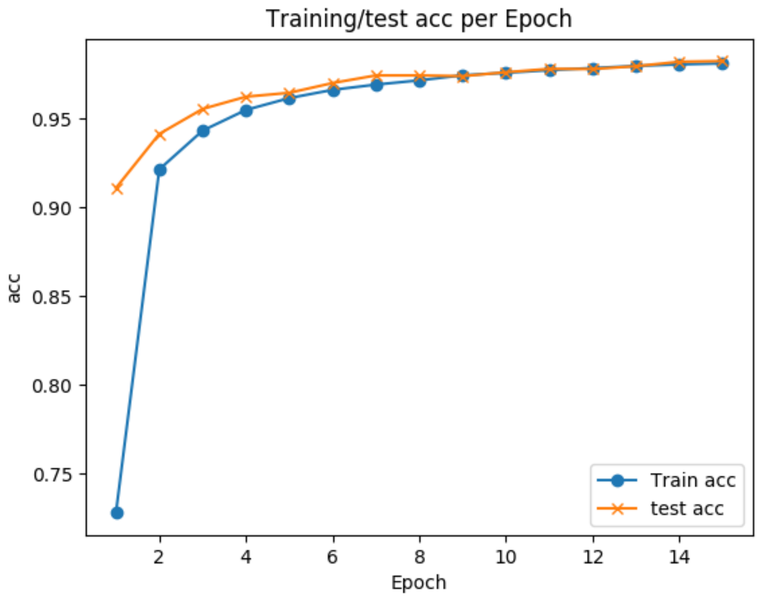


Figure 8: Learning Rate = 0.01, Epoch = 15, Batch Size = 128

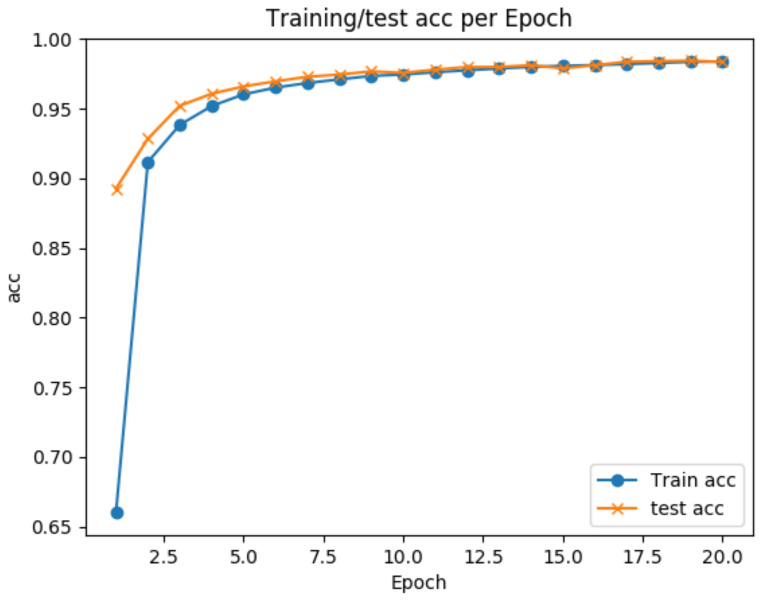


Figure 9: Learning Rate = 0.01, Epoch = 20, Batch Size = 128

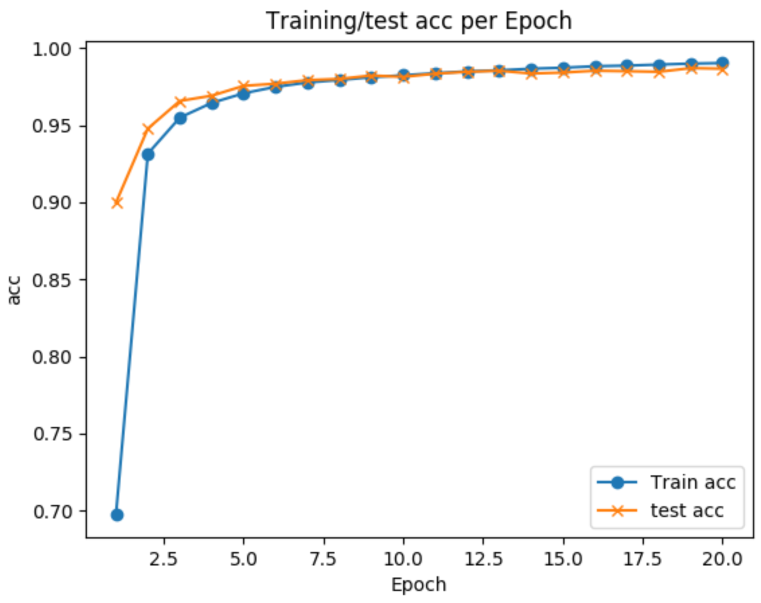


Figure 10: Training Accuracy And Test Accuracy Verse Epoch

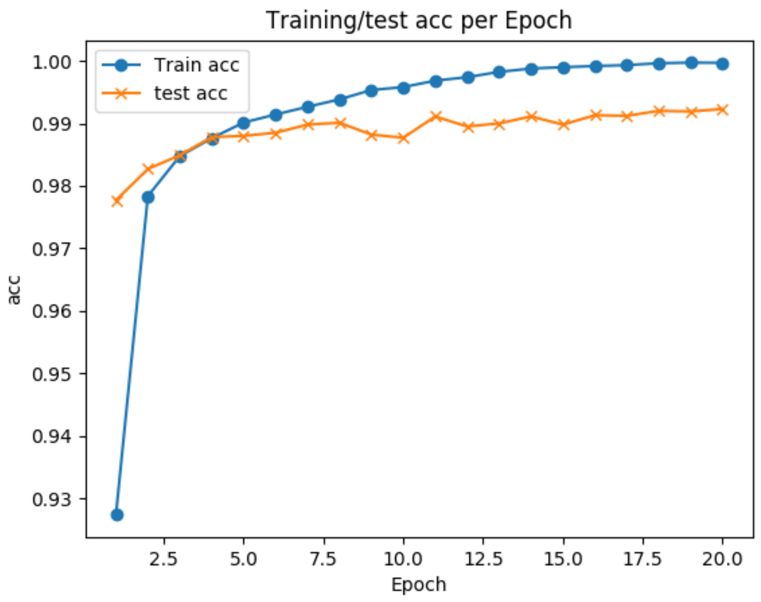


Figure 11: Training Accuracy And Test Accuracy Verse Epoch

CNN is hugely popular, achieving best performance on many problems, while It need a lot of data to perform well. Meanwhile, it needs a long time to train the model. Sometimes need to involve GPUs for massive parallel computing. Besides, it takes some work to select architecture and hyperparameters. The biggest problems is that it still not well understood in theory, which means it still a black box for scientists.

1.3 Apply trained network to negative images

1.3.1 Abstract and Motivation

As can be seen, LeNet-5 yield high accuracy on regular images. They however significantly under-perform when testing on negative images. Figure 11 shows that the test accuracy is quite lower than the training accuracy while predict with the negative image use the best parameters above. The accuracy is 0.32.

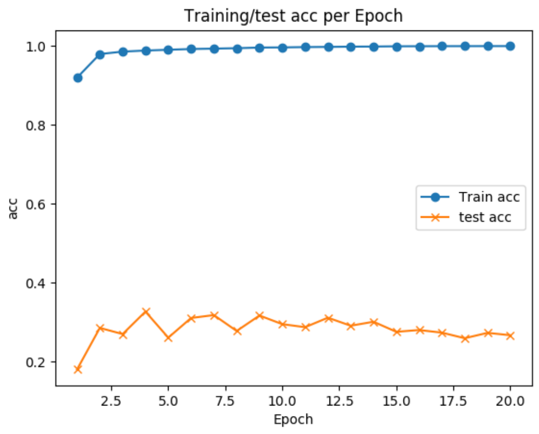


Figure 12: Lenet-5 Network Trained On Negative MNIST

1.3.2 Approach and Procedures

Use the negative image as the test images on LeNet-5. Plot the accuracy on regular validation images, regular test images, and negative test images versus the epoch number for LeNet-5. The result shows that the accuracy on the regular test image and regular validation image are matched with the training. However, when it comes to a negative image, it seems overfitting and performs badly on the testing data. **The reason** may be that the training data is insufficient thus the model cannot learn the invariant features of each class. When the test data is not distribute as the input data, which in our case is negative image, the model cannot distinguish between classes.

Thus, I decide to train the model with a negative image. Before training, I randomly select half of the training data set and inverse them into the negative images. Figure 13 shows some part of the Negative image data. It is easy for human to observe the number of them. But for LeNet-5, we need to re-train it with the new data set.

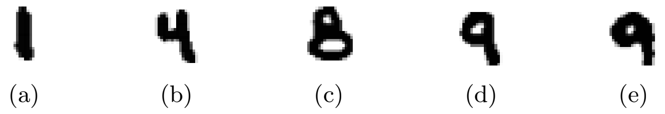
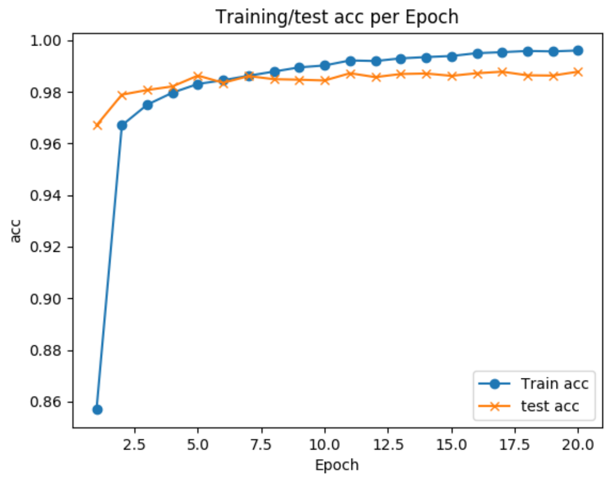


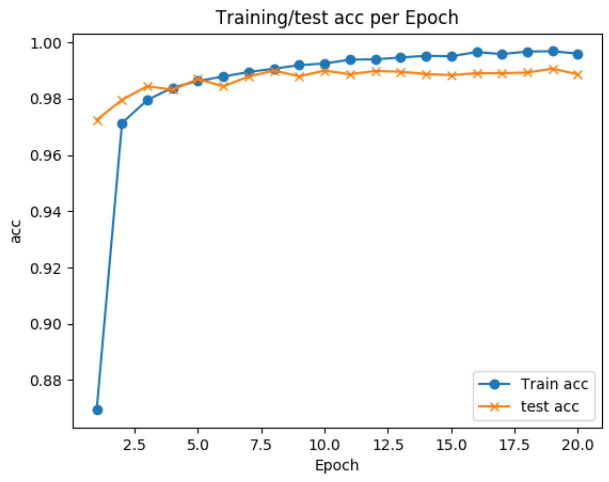
Figure 13: Negative Image

1.3.3 Experimental Result

Set the parameters: learning rate = 0.01, epoch = 20, batch size = 64, momentum = 0.9, Nestrov = True;



(a) Regular Image



(b) Negtive Image

Figure 14: Negative Image

The best result for regular image is 98.78, while for negative is 98.87.

1.3.4 Discussion

Despite the impressive performance of CNNs on images distributed similar to training data, their accuracy is much lower on negative images. If a transformed test data has a large difference to the original training data, the LeNet-5 might not be able to correctly classify it. To correct this drawback, my observations indicate that if we add a half of negative image into the training set, the performance of LeNet-5 will improve significantly.