EE 569: Homework #5: Image Recognition with CNN
CNN Training and Its Application to the MNIST Dataset

(a) CNN Architecture and Training

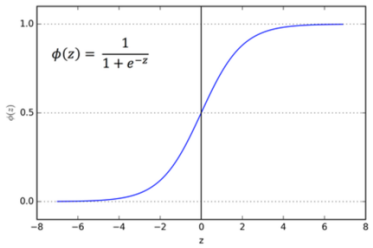
Describe CNN components in your own words: 1) the fully connected layer, 2) the convolutional layer, 3) the max pooling layer, 4) the activation function, and 5) the softmax function. What are the functions of these components?

1) **Fully connected layer**:
   Fully connected layer is at the very end of a CNN. All the neurons in these layers are connected to all other neurons. It gives a vector which has class probabilities. We determine the class by assigning it to the class with maximum probability. First it randomly assigns weights to the network and learns the weights by back propagating the error.

2) **Convolutional layer:**
   All the neurons in Convolutional layer are not connected to each other. These layers act as an image filter and create a modified image. But unlike Laws filters, these weight values aren't defined. The network learns these filter values on its own. These are helpful in extracting low level features like edge, color, gradient. It is also used for dimension reduction of the input image.

3) **Max pooling layer:**
   Pooling layer is another dimensionality reduction method. It performs noise suppression. Average pooling only reduces dimension and performs much worse than max- pooling. Max pooling acts like attention in human vision. It takes into consideration high value and disregards the small values while Average pooling acts like a blurred vision.

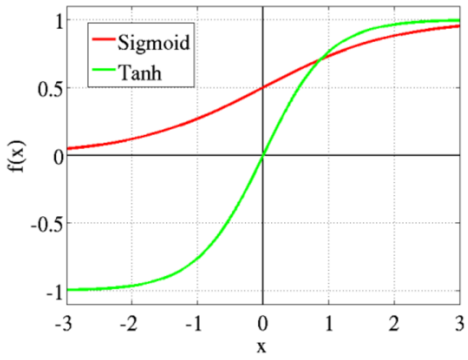
4) **Activation function:**
   We use an activation function to introduce non-linearity to our so-far linear network. Linear activation function can be used. But, the high performance of a CNN cannot be obtained by linear activation function. Depending on the requirement of our range, and performance we can use different activation functions:
   1. <u>Sigmoid:</u>



   The primary reason of using sigmoid being it trasfers the entire range to values between 0 and 1. Which can be used as probability values. Also, it's differentiable and monotonic for the backward propagation.

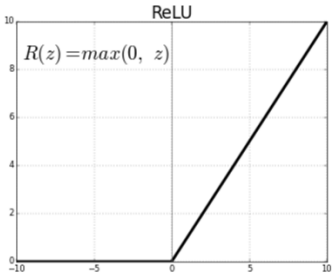
Network might get stuck during training using sigmoid.

2. Tanh



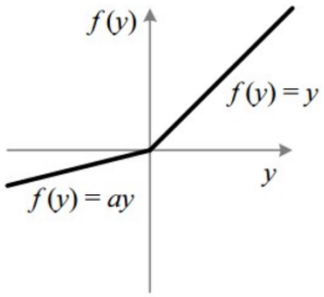
Tanh is very similar to sigmoid except it maps high negative values to -1 and positive to +1. It is differentiable and monotonic. It's derivative is not monotonic. It is mainly used for classification of 2 classes.

3. ReLU



Range of ReLU is from 0 to infinity. RELU is the most commonly used activation function in CNN. It is monotonic and differentiable only except for 0 which is manageable. It clips the negative response completely. In a cascade system, different polarity responses cause a lot of confusion as positive and negative correlation are good. But interpreting them in a cascade system is very difficult. Hence, clipping the negative response gives a good performance.

4. Leaky ReLU



Sometimes, we don't want to completely clip the negative responses. Hence, we use Leaky ReLU. It's monotonic and ranges from negative infinity to positive infinity. And gives a very small value for negative responses.

To determine the best activation function, it's best to cross validate over all of them and choose the one with best consistent performance.

5) **Softmax function:**

$$\sigma(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}} \quad \text{for } i = 1, \ldots, K.$$

Softmax function takes an activation function which converts a k-D vector into k-D normalized vector of probabilities. It is better than direct normalization as it takes exponent of each value before normalizing. Highlighting important values and suppressing less important values. It is usually added in the last layer where we have to actually classify the input vector into one of the k- class.

What is the over-fitting issue in model learning? Explain any technique that has been used in CNN training to avoid the over-fitting.

Over- fitting happens when a model forms the fit to the data that not not generalizes the trend but also the noise in data. Since neural network has many parameters, which is good for learning. It has high potential for over fitting the data.
Following are some techniques used to prevent over-fitting:
Early stopping
Drop Out:

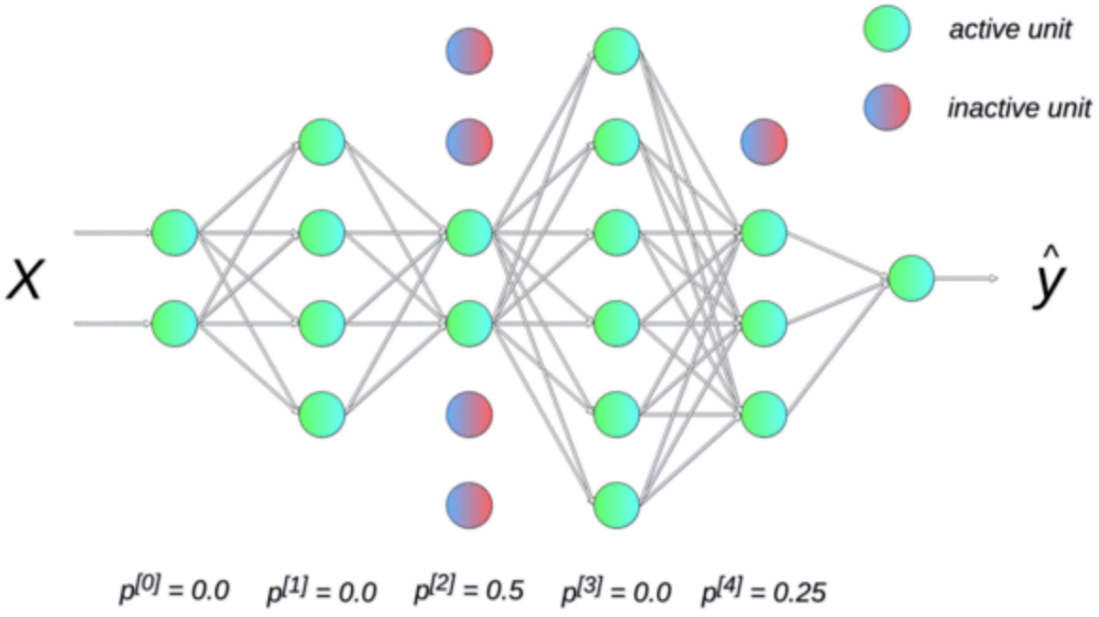


Figure 5: Visualization of a dropout.

Every layer can be given a dropout rate. This rate is a hyperparameter. In every iteration, neurons are dropped randomly depending on this dropout rate specified. This introduces randomness in the network and kills the bias. Thus, avoiding over fitting.

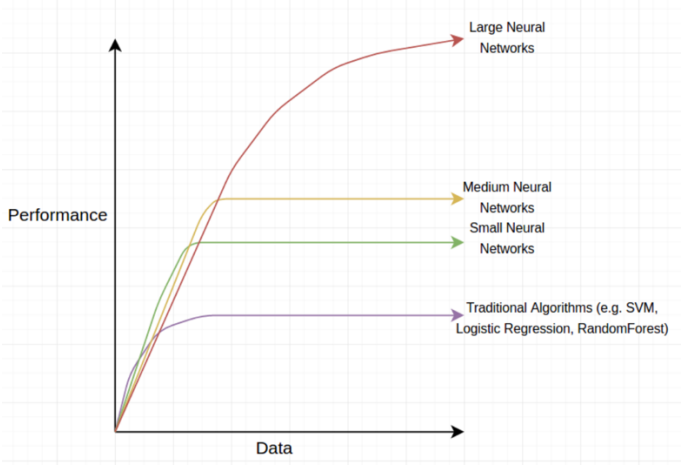
Why CNNs work much better than other traditional methods in many computer vision problems? You can use the image classification problem as an example to elaborate your points.

Traditional Machine Learning problem is divided into two parts:

1. Feature Extraction
2. Classification

We need domain-specific knowledge for feature extraction of every problem. Also, we extract these features manually in most cases. Classification algorithm optimizes over these features.
In CNN, the network does feature extraction for us. We do not have to manually do it. CNN not only optimizes over the classification network weights but also over the feature extraction layer in every epoch. This is one reason for better performance by CNN.

Also, many machine learning algorithms saturate after certain amount of data. Their learning rate becomes almost negligible beyond a certain amount of data. But Neural Network models keeps learning as we increase the amount of data.



In today's world, where we have access to huge data and computation power, CNN performs better with this large data.

Explain the loss function and the classical backpropagation (BP) optimization procedure to train such a convolutional neural network.

Loss function is defined in any machine learning algorithm to evaluate the model and perform parameter updates. When error is high, the loss function outputs a large number. Optimization algorithms are used to reduce this loss function for optimal parameters of the model.

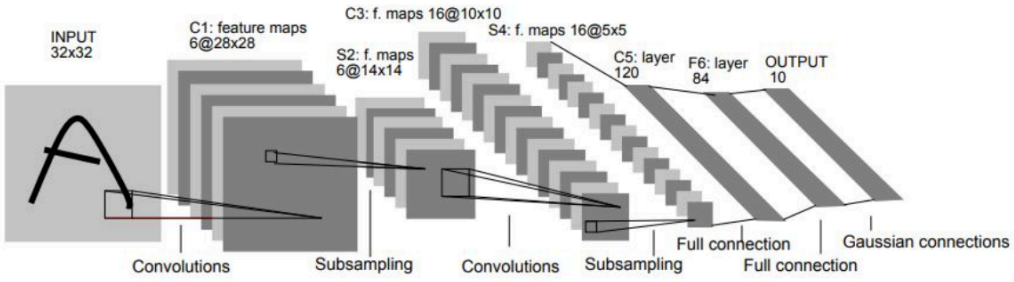
Back propagation is like a classic feedback control system model. We assign random weights to the neurons and do classification. We compute gradients by recursive application of chain rule. These

gradients are used for performing weight updates. Now, classification is done again to get the error. This process is continued till the weights converge. We need to check using validation data that we don't start overfitting the system.

(b) Train LeNet-5 on MNIST Dataset

Motivation:
It's the oldest proposed method for CNN. It's quite simple and straight forward both to understand and implement.

LeNet-5:



Original Image published in [LeCun et al., 1998]

It consists of two convolutional layers followed by pooling layer followed by a flattened convolutional layer and two fully connected layers and a final softmax layer.

First convolutional layer:
It converts a 28x28x1 input into 24x24x6. It has 6 filters and uses 5x5 filter for convolving. These convoluted images are padded by 2 to make it 28x28x6

First Subsampling layer:
We do max pooling using 2x2 matrix to reduce dimensionality. This is then passed through a ReLU activation function.

Second convolutional layer:
This has 16 filters of size 5x5. It converts image of size 14x14x6 into 10x10x16.

First Subsampling layer:
Again, convolution is followed by pooling and then ReLU activation function.

First Fully connected layer:
5x5x16 layer is flattened and these 400 inputs are fed an input to 1st fully connected layer. This layer has 120 neurons. All the inputs are connected to all 120 neurons. And ReLU activation function is used before passing it to next layer.

Second fully connected layer:

Output from 120 ReLU units is fed to this layer. It has 84 neurons. All are fully connected followed by ReLU activation.

Output layer:
The output layer has 10 neurons(10 classes). In this layer, we use softmax instead of ReLU to find probability of classes.

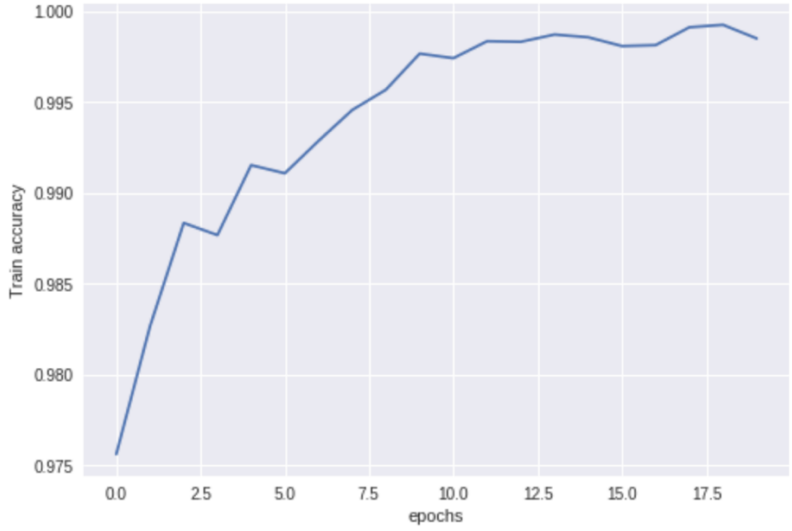
Approach:

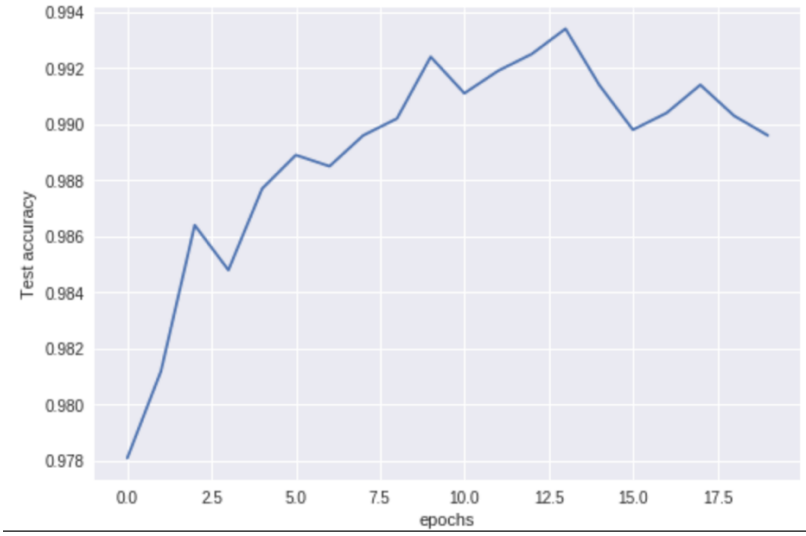
1. Transform image data to tensor.
2. Load train and test MNIST data.
3. Define the network using dimensions mentioned in LeNet-5 architecture.
4. Define Cross Entropy loss criterion and Stochastic Gradient Descent optimizer.
5. For each epoch, get the inputs, make the gradients zero and optimize the loss function.
6. Check the train accuracy and test accuracy.
7. Plot train and test accuracy vs. epochs.
8. Change the hyperparameters and find mean and variance for 5 settings.

Experimental Results:

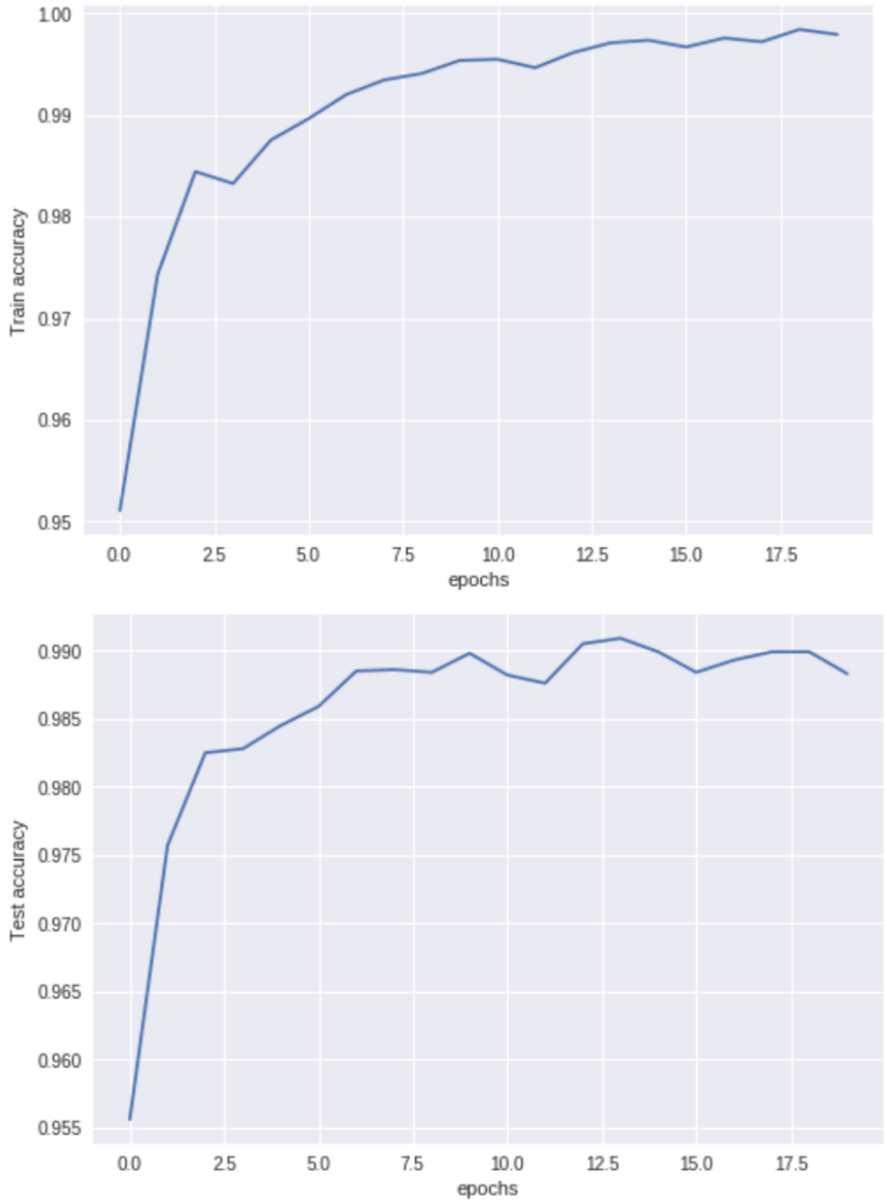
Batch Size as hyper parameter:
Epochs = 20, Learning rate= 0.001

Batch Size= 5
train accuracy 0.9985
test accuracy 0.9896

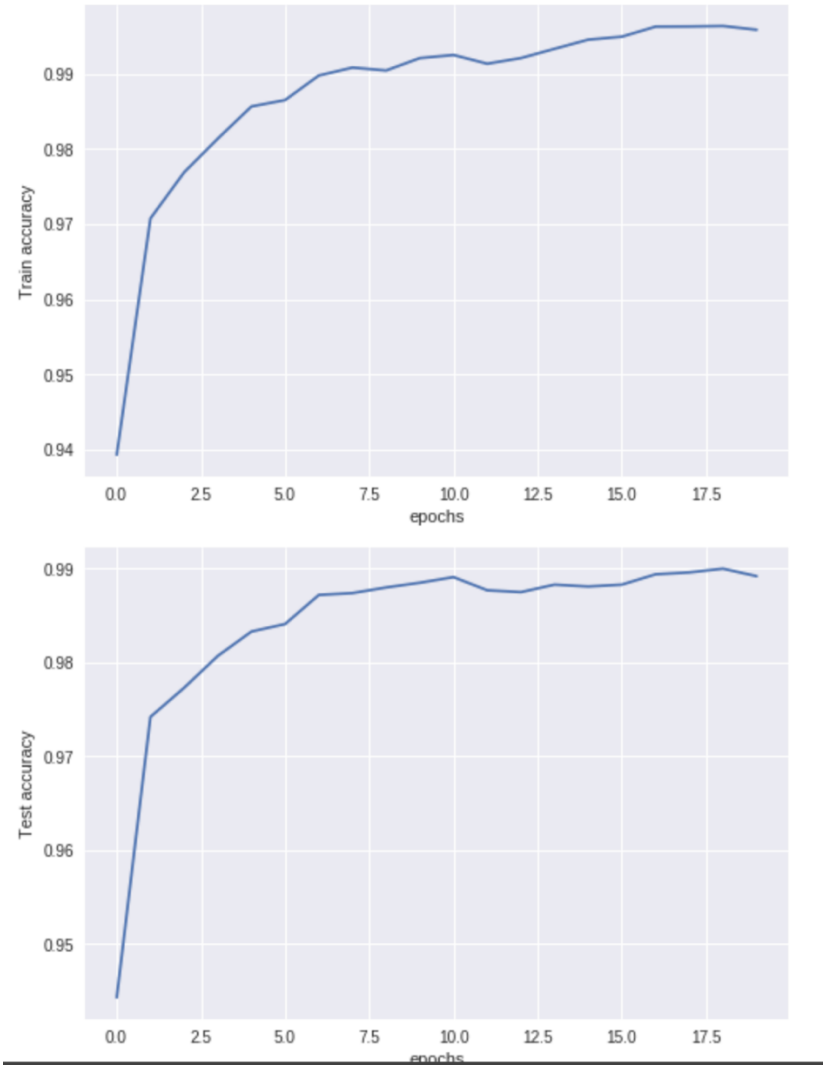




Batch Size = 10
train accuracy 0.9978833333333333
test accuracy 0.9883

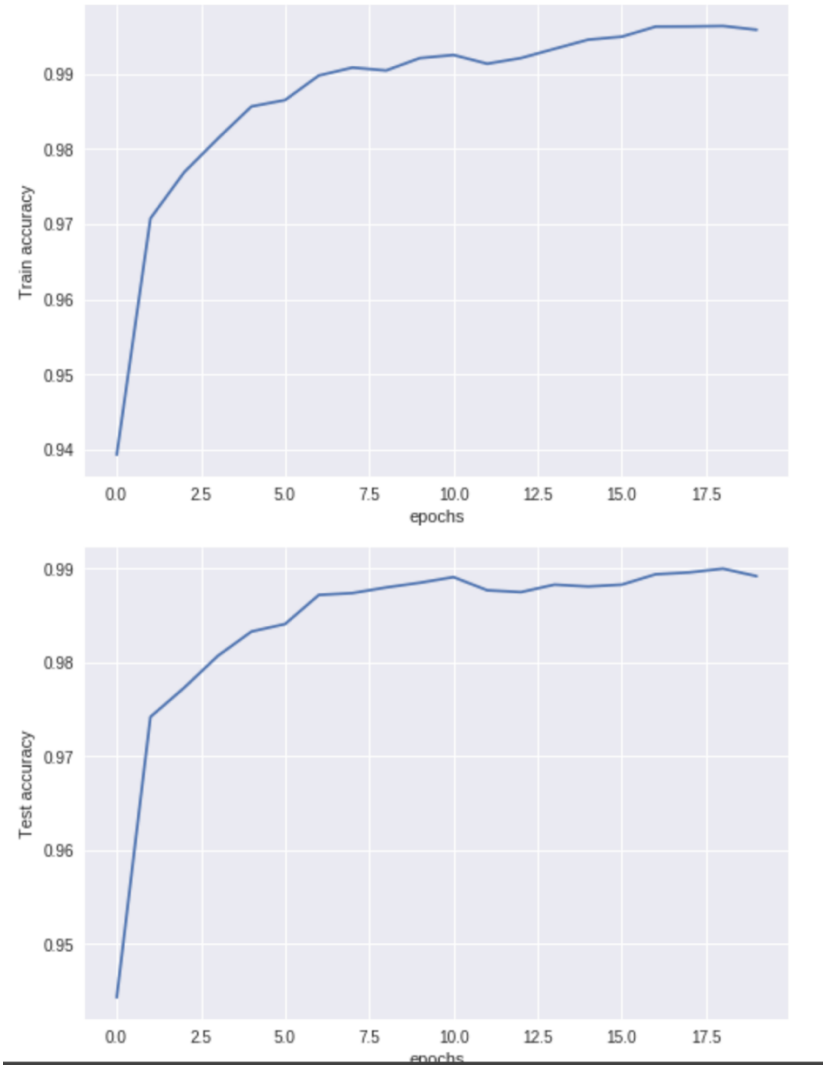


Batch Size = 20
train accuracy 0.9958333333333333
test accuracy 0.9892

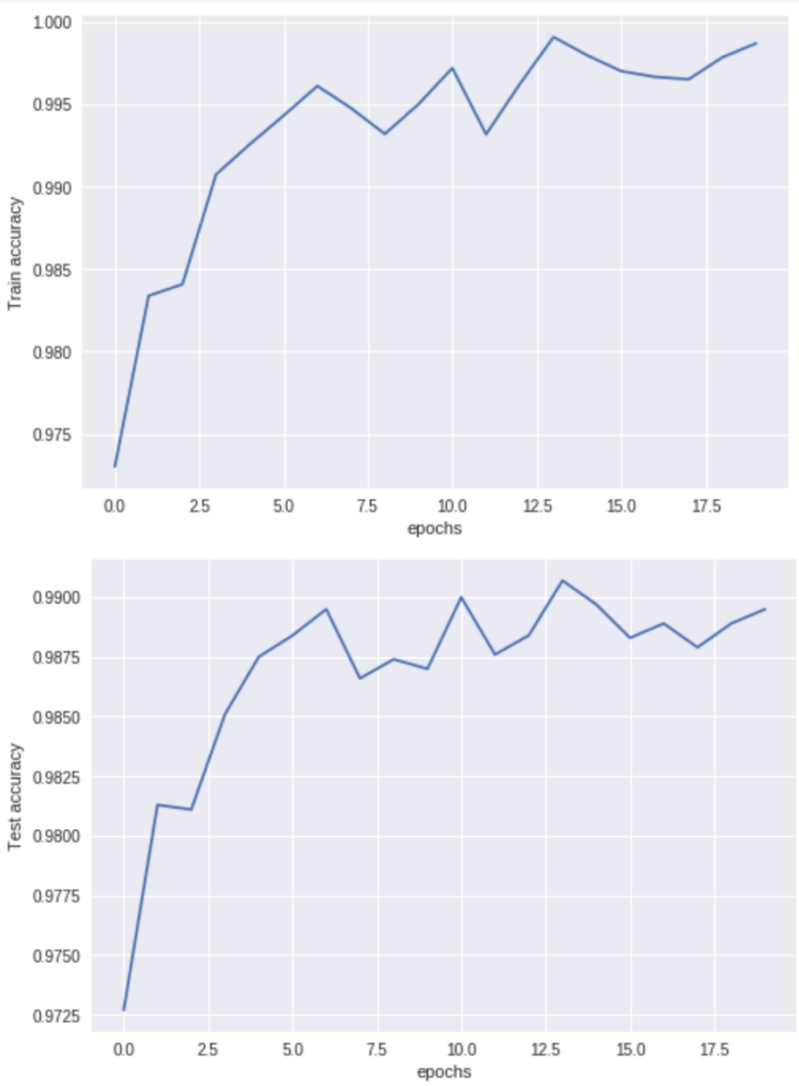


Learning rate as hyper parameter
20 epochs, batch size: 20

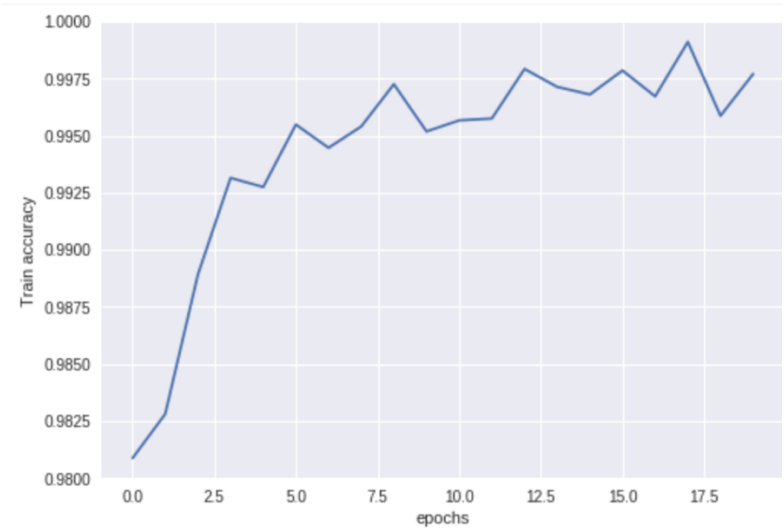
Learning rate= 0.001
train accuracy 0.9958333333333333
test accuracy 0.9892

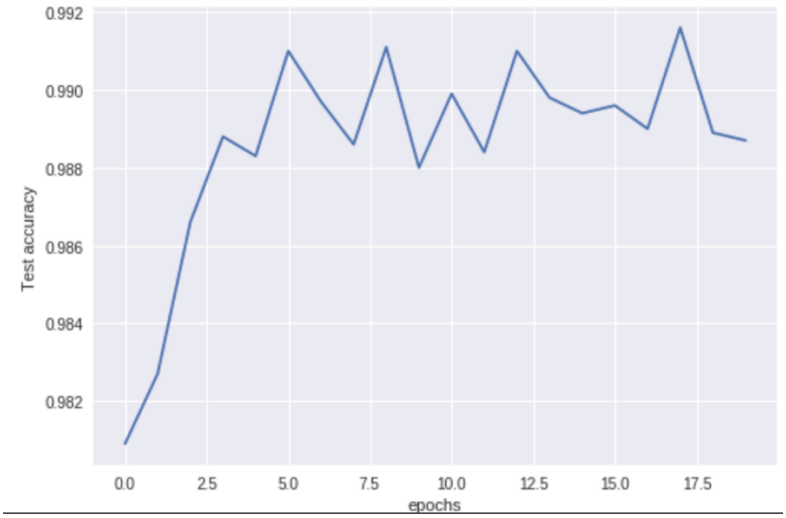


Learning rate = 0.008
train accuracy 0.9986833333333334
test accuracy 0.9895



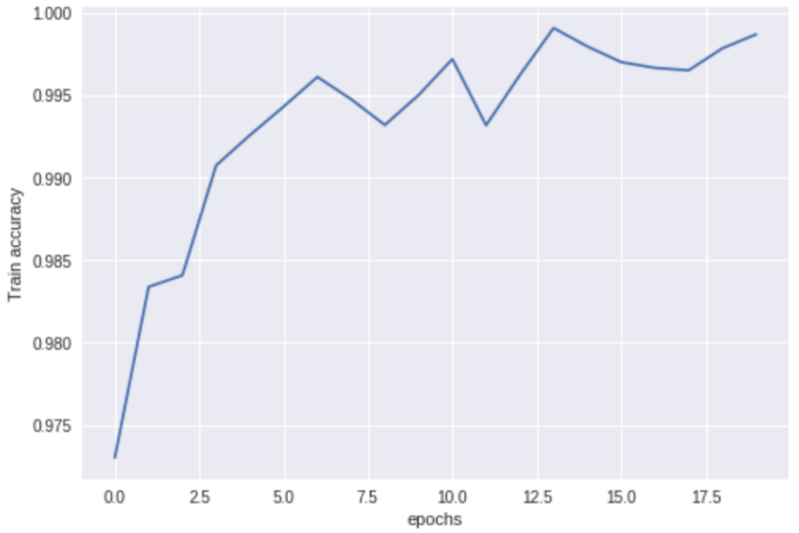
Learning rate= 0.01
train accuracy 0.9977
test accuracy 0.9887

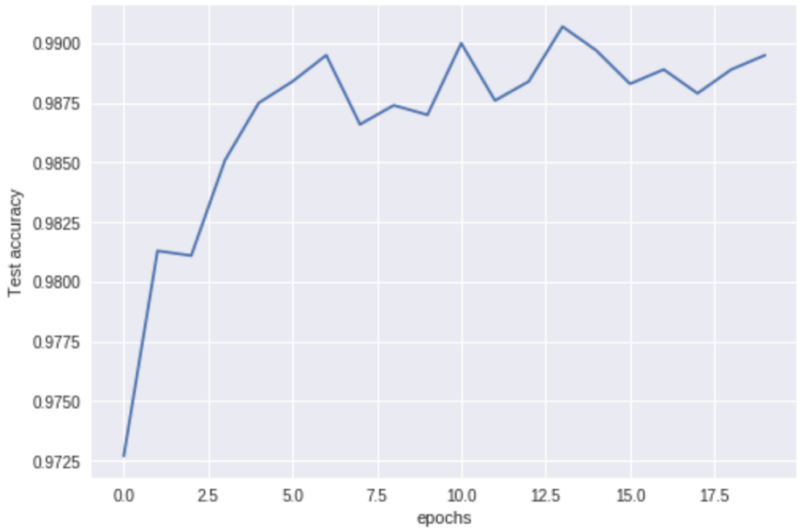




Number of Epochs as hyperparameter:
Learning rate: 0.008, batch size=20

Epochs= 20
train accuracy 0.9986833333333334
test accuracy 0.9895

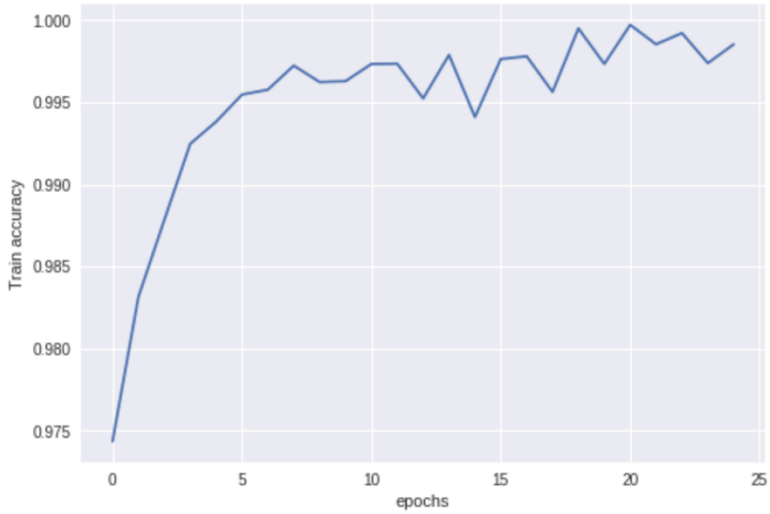


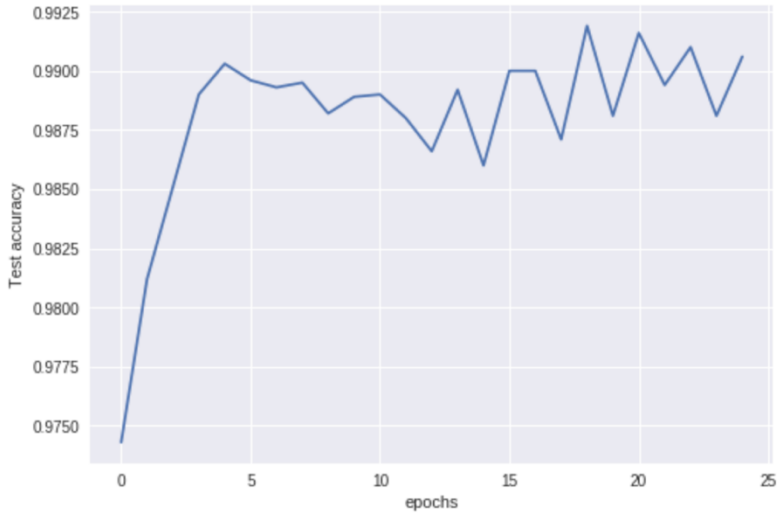


Epochs= 25:

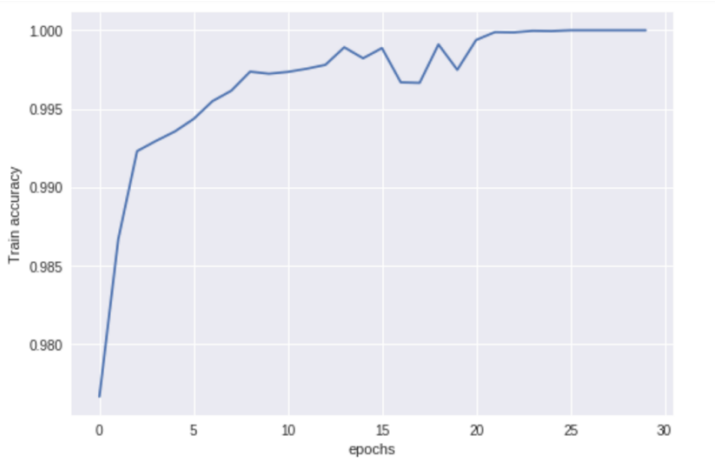
train accuracy 0.9985333333333334

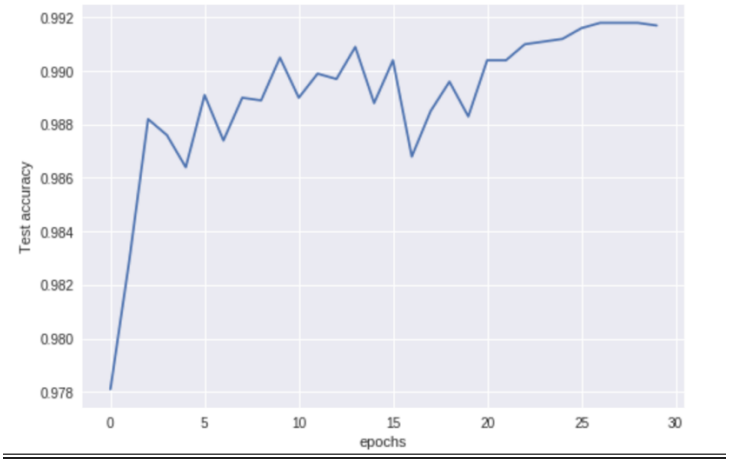
test accuracy 0.9906



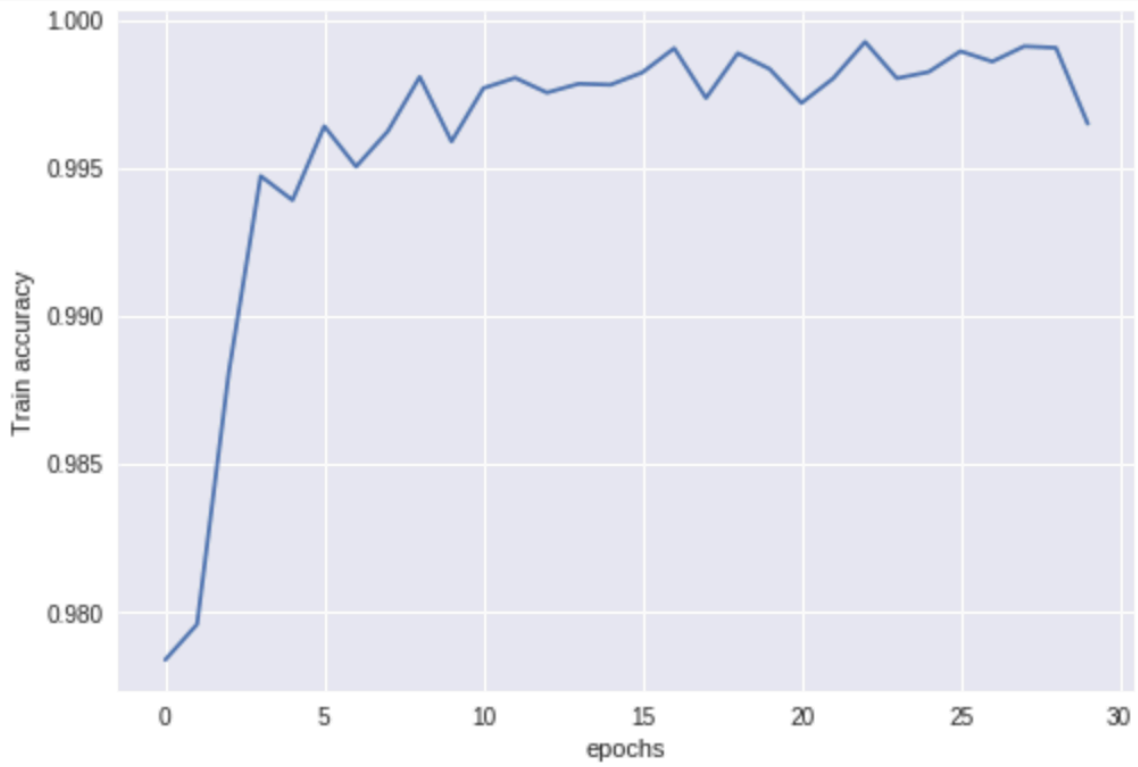


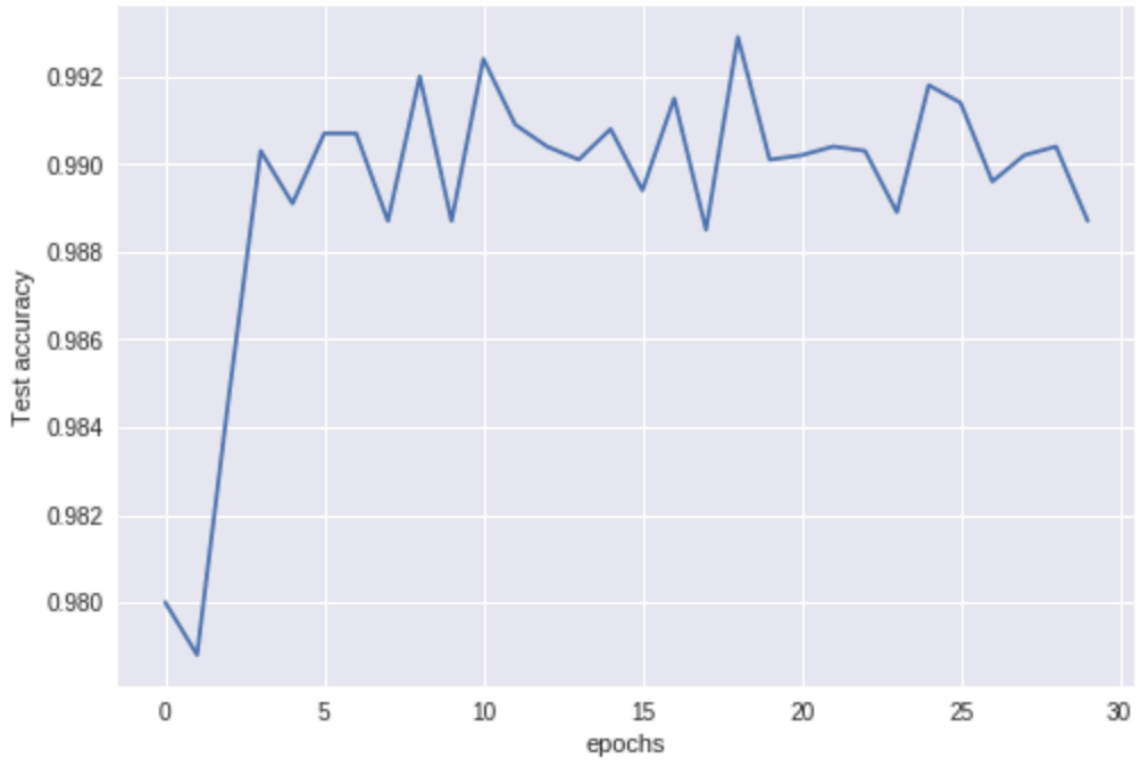
Epochs = 30:
train accuracy 1.0
test accuracy 0.9917





Using Adam optimizer:
train accuracy 0.9965
test accuracy 0.9887





Discussion:
As we can see the test accuracy by changing the hyper parameters is almost the same. It fluctuates with very less variance.

- Changing batch size does not change the test accuracy much. It takes more time for the network to finish same number of epochs. **Batch size of 20** is chosen as optimal. Even though batch size 5 gives slightly more accuracy, it takes much more time for the entire process.
- If we increase the **learning rate**, the fluctuation in the accuracy goes on increasing. An optimal of **0.008** is chosen.
- Increasing the number of epochs, both train and test error go on decreasing. For 30 epochs, we can observe the train and test reduced. But the difference in accuracies for 25 and 30 epochs is very little. Also, train error for 30 epochs is 0 indicating high chances of overfitting. Even though test accuracy increased, it's better to take **25 epochs**.
- Optimal optimizer: **SGD**

Mean and variance of train and test accuracies over above settings:
Mean:
Train: 0.9981619047619048
Test: 0.9896571428571427
Variance:
Train: 1.3737074829931981e-06
Test: 1.1510204081633022e-06
We can see that mean accuracy is above 98% for both test and train. Changing settings does not affect the performance of network much. Both variances are very low. We can see that test variance is also very less, generalization obtained is good.

(c) Apply trained network to negative images

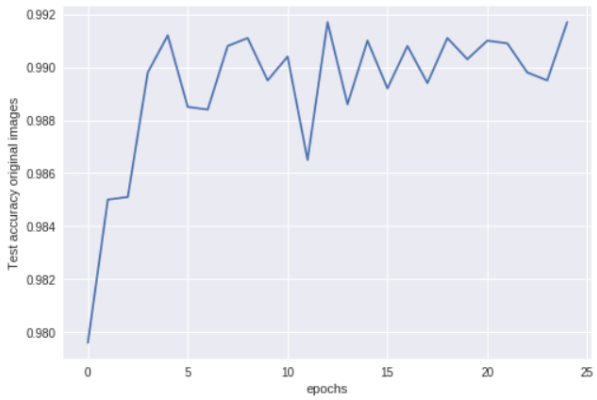
Motivation:
As humans, we can very easily detect the number even if image is negated. The goal in this part is to check if our trained neural network model succeds in recognizing the negated images. And if not, make changes in our metghod for it to give atleast 98% accuracy for these negated images.

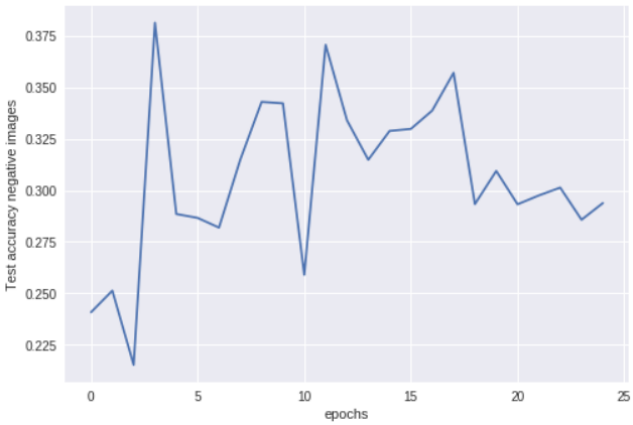
Approach:

1. Negate the images.
2. Pass them as test data and report the accuracy.
3. Train the classifier with augmented dataset of both the image and negated images.
4. Test again on the same classifier and report the new accuracy.

Experimental results:

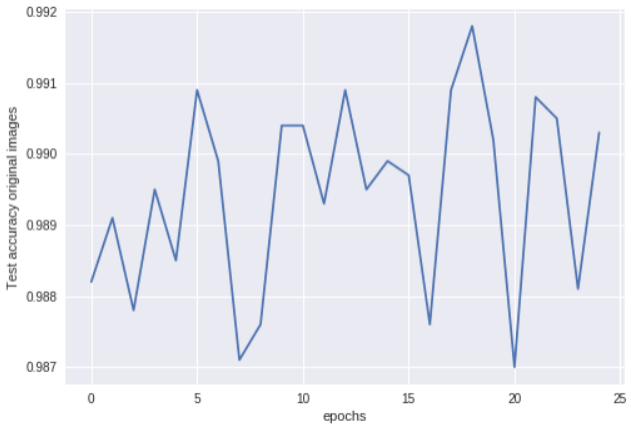
Original image vs negative image performance of using model trained in 1b

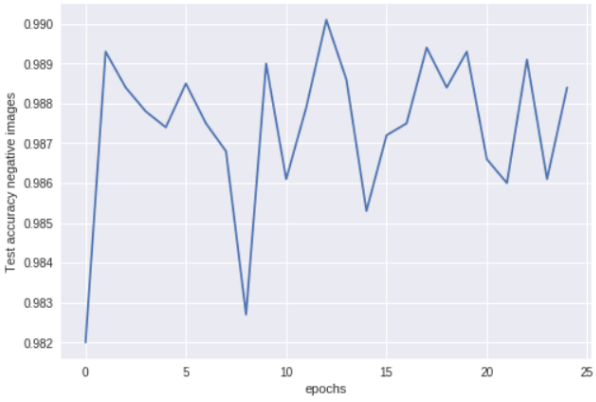




accuracy: 0.9917 **accuracy= 0.2938**

Original image vs negative image performance using augmented data:





accuracy: 0.9902 **accuracy: 0.98854**

Discussion:

We can see that our CNN model does not work well on negated data. One reason is when we train any classifier and test it on a dataset, we assume they are sampled from the same distribution. ie. Train and test data are similar and random shuffling shouldn't change the output by much variation. But, when we train for image with one background only and expect it to learn the pattern, it does it well for the same background. Also, if we give enough data with both backgrounds, the classifier learns from it. But, the LeNet5 network does not learn to classify a never seen background image even if it's just negation and very obvious for humans.

References:

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a
http://cs231n.github.io/optimization-2/
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html