MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

**EE569 - INTRODUCTION TO DIGITAL IMAGE PROCESSING**
**HW#4 - 04/29/2018**

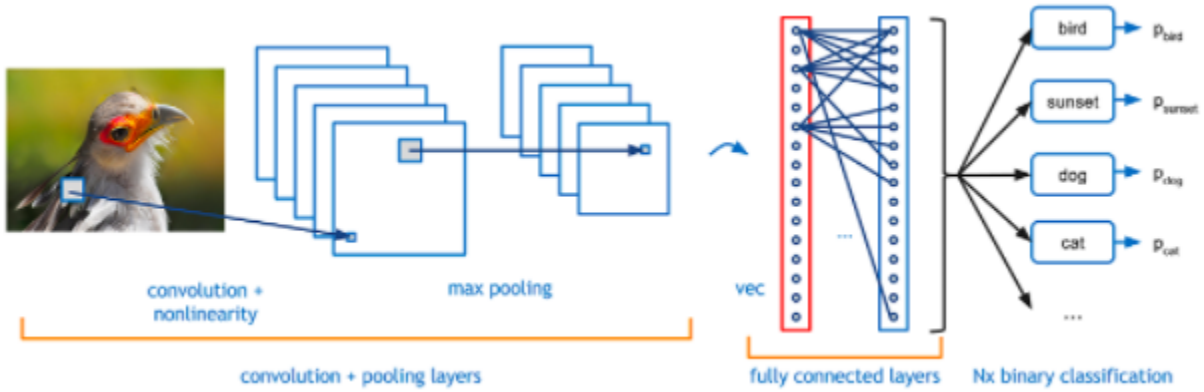
Table of Contents

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

# PROBLEM 1 : CNN training and its application to MNIST dataset

## 1.1ABSTRACT :

Artificial neural networks are inspired by biological neurons. Convolutional Neural Net(CNN) is a type of feedforward Artificial neural network.
CNN's are the most popular deep learning structure. During the ImageNet competition, the accuracy produced by AlexNet catalyzed CNN for image classification. CNN are primarily used in recommender system and image classification problems.
It is also computationally efficient and are finding applications as solutions for every real-world problem.




One of the initial attractive problems which neural networks tried to solve was the handwriting recognition using MNIST database. It produced an exceptional accuracy compared to classical image processing methods. In this problem, the aim is to classify the MNIST database using LeNet-5 architecture and then improving upon the structure to get better accuracies.

## PROBLEM 1 A )

## 1.2.A. APPROACH :

Convolutional neural networks are made up of multiple neurons which can learn weights and biases and adapt and produce an output. We presume that the inputs to our ConvNets are images. A neuron receives an input and performs dot product with the weights and adds the biases and is usually activated by a non-linearity. The output is usually signified using a differentiable score metric like accuracy and loss functions with respect to the learned network.
A network is said to be a deep network if it has more than one hidden layer. The network architecture is made up of three main types of layers- Convolutional Layer, Pooling Layer, and Fully-Connected Layer. Each specific layer takes a 3D volume as an input and transforms to an output of 3D volume.

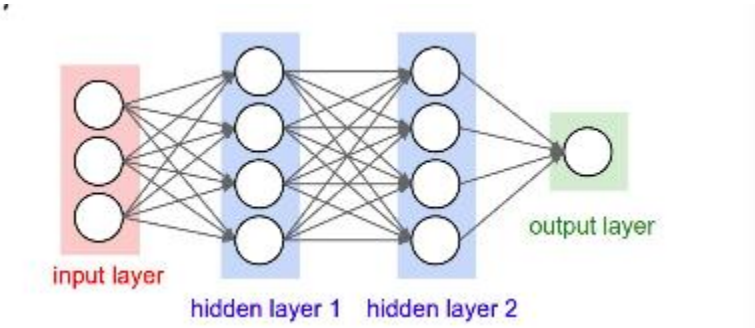
MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

The reason why a CNN needs multiple layers is to have a high level overview of how the CNN actually works. For example, we can understand that the first layer will try to detect edges, then next following layers will cluster them to similar shapes and then consider different object positions. Hence, a deeper CNN would be able to capture more non-linearities in the image but this increases the computation complexity as the number of training parameters increases.

**1.4.A. DISCUSSION :**

- **The CNN components are :**

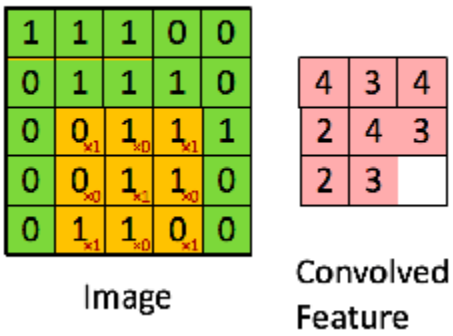
1. The fully connected layer




It is a multi-layer perceptron. It is one of the main methods using which the network learns non-linearities in the data. Neurons in this layer have connections to all activations in the previous layer. The connections do not jump or skip any layers. It has the concept of hidden layers too which gives the idea of a "black box" in the network. The activations are calculated using the matrix multiplication and added with a bias and is passed to the next layer. It has a single output layer which is basically the result of all the computations.

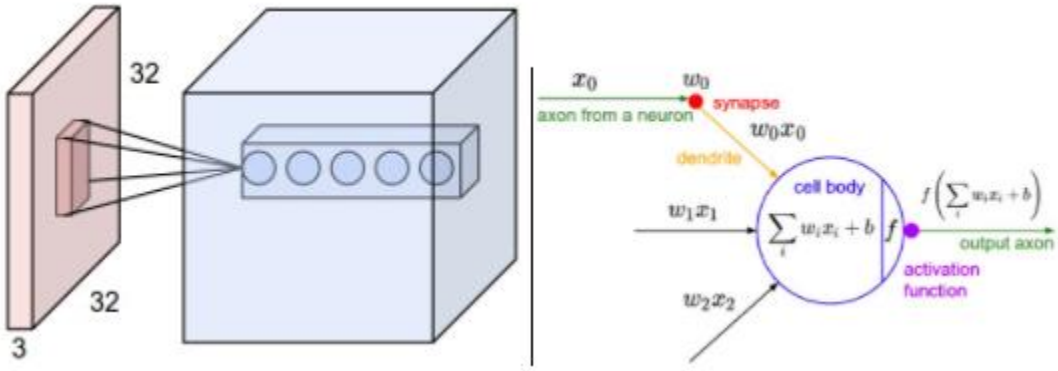
2. The convolutional layer

We know that in images, when we apply convolution of a filter with the image underneath it, we implement an element-wise multiplication and addition.




Source : https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

In CNN, the filter consists of a set of learnable parameters. The filter is usually spatially smaller than the image and extends to the volume of the image. Thus it gives us the local spatial dependencies of an image For example, an image with R,G,B channels will have a convolution filter applied to it which extends to all of the channels. Thus, the convolution operation in CNN maintains the volume of the input.




Source : http://cs231n.github.io/convolutional-networks/#overview

The convolution operation is accompanied by a stride function. When the convolution filter is applied, it moves across the width and height of the image in the row-major ordering method.We can stack multiple activation maps which are produced as a result of multiple convolutional filters.
For every pixel, the dot product with the elements of the filter and the input pixel values are calculated. This operation produces a 2D activation map with the responses to the specific filter. This way, the network learns the parameter values which activate for a particular visual feature during training. But we do need to specify parameters like number of filters, filter size, and architecture of the network.
The output 2D activation map produced is controlled by the below factors:

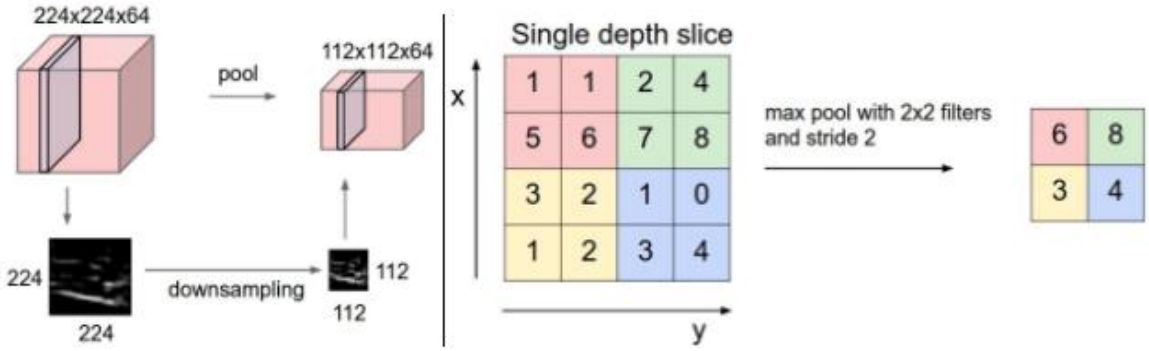
- Depth: This corresponds to the number of filters used for convolution. We can use the filters in every dimension of the depth to activate edges, blobs and other important features for image analysis.
- Zero padding : Applying zero padding enables us to apply convolution on the edge pixels as well.
- Stride : This is the number of pixels the convolution filter passes by, it is usually set to 1, if we don't want to miss out on each pixel. If the stride is set to 2, the filter jumps two pixels when sliding over the image. A larger stride size produces a smaller 2D activation map.

3. <u>The max pooling layer</u>

Pooling layer is usually inserted in between Convolution layers to reduce computation and over fitting. It down-samples the input spatially. This makes room for assumptions

about the underlying features in the sub-region. This usually does not use zero padding while computing a max of the block under consideration.

There are different types of pooling - Max-pooling, Min-pooling, and Average-pooling. The most common pooling operation used is Max pooling.



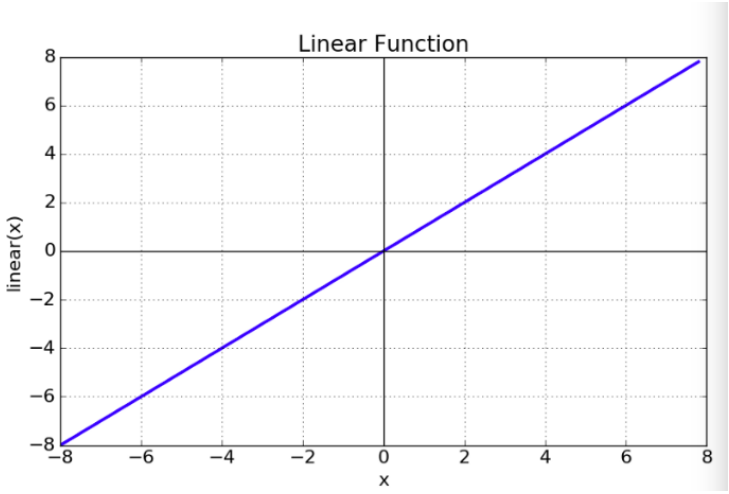

Source : http://cs231n.github.io/convolutional-networks/#overview

Above is an illustration of a max function which operates on a 2x2 block of image with a stride of 2. It finds the maximum value in every block and uses that to represent the entire block in the spatially reduced output. This does not change the depth of the image.
The pooling layer accepts two hyperparameters which is the size of the image block to be considered and the stride.
Pooling with a larger size of the block can lose out on a lot of important features. Hence, the size should not be too large to maintain the trade off between computation time and over-fitting over losing data.

4. The activation function

The activation function which can be applied t o a CNN can be divided into 2 types:

- Linear Activation function: As we can see from below, a linear activation or a transfer function maps the input to the output linearly. This is not confined to any output range and does not account for the complexity and non-linearities in the input layer.

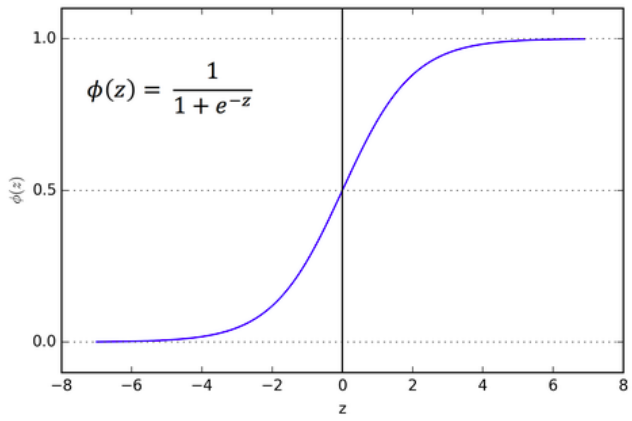



- Non-linear activation function

  Many images have features which cannot be extracted using linear functions, hence non-linear functions were introduced. Non-linear complex mapping functions enables the network to map and approximate any function.

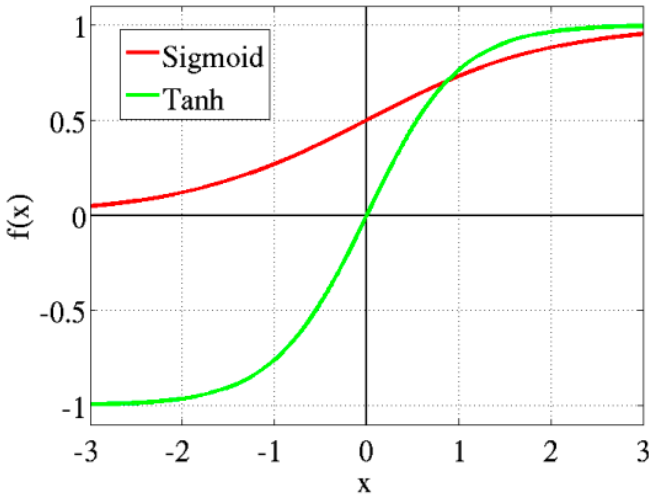
  There are four major non-linear activation functions:

  - Sigmoid or Logistic Activation function :




    As we can see, the function is non-linear. It maps the input to output values between 0 and 1. This models a good function for a problems which needs output to be in terms of probability distributions. The function is monotonic and differentiable. The softmax function is more generalized function for a probabilistic output for a multiclass classification.
    It is not very popular anymore because it saturates and kills gradients (Gradient Vanishing Problem) and the outputs are not zero centered causing slow convergence.

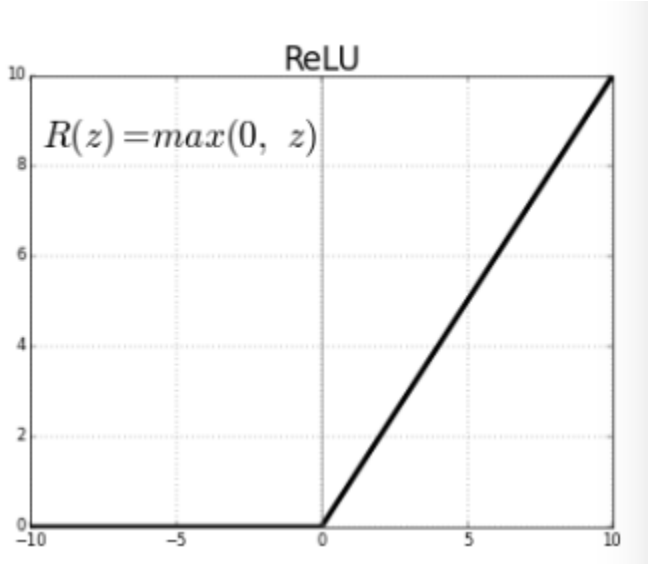
- Tanh function : This maps the input range to output range between -1 and +1.




This is a better activation function as it preserves the negativity of input functions. It also suffers from the vanishing gradient problem but converges much faster than sigmoid as the outputs are zero centered.

The function is not monotonic and differentiable. This is mainly used in two-class classification problems.

- ReLU (Rectified Linear unit)



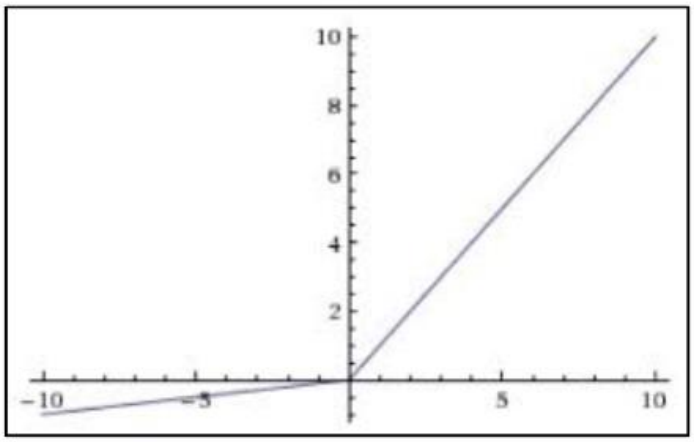

ReLU replaces all negative values in the feature map by 0. Any function can be approximated with combinations of ReLU. Range of ReLU is [0,inf). ReLU is implemented using just a thresholding operation, hence the operations involved is inexpensive.

It converges much faster than sigmoid/tanh and output are not zero centered.

- LeakyReLU function

  To avoid completely setting any value less than zero tobe zero, leakyReLU introduces a small negative slope. The function is given by :

  $f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x >= 0)(x)$ where $\alpha$ is a small constant.




5. The softmax function

Softmax function is a probabilistic function which squishes the obtained values to a value ranging between 0 and 1. It is usually the final layer in multiclass problems, where each value represents the probability with which the given test image belongs to a class. It is also a generalization of logistic regression and are trained under a cross entropy loss.

- **Overfitting issue in model learning**

Overfitting occurs when the model is fitting the training data too well and hence it ends up misclassifying or badly classifying the test data.
Overfitting is a major issue in CNN where the train accuracy is 100% and sometimes the testing accuracy drops to 50%. Hence, care should be taken to make sure that the model does not overfit and give errors.
Steps for reducing overfitting:

1. Use more data

One of the obvious alternatives would be to collect more data but this is not feasible and we would have to work with the data we already have.

2. Cross-validation of the training set

This sets aside a separate subset of the training set as a validation set and trains the model with the rest of the sets. Then it iterates over the obtained values and sets the network weights with the most optimum ones.

3. Use data augmentation

This introduces randomness to the model, we can take an image rotate it, scale it and perform affine operations on it and then feed it to the model. Care should be taken to perform data augmentation only on the training set.

4. Add pooling layers, regularizations

One of the major methods to curb overfitting is by adding a max pooling layers after a convolutional layer. If it is a 2x2 max-pooling operation only 1 maximum value is selected to represent the entire 2x2 block thus leaving considerable amount of dubiousness for the model to train on.
Another popular method is regularization which introduces extra cost terms to minimize over and adding a penalty for misclassification. Hence, it pushes coefficients of many variables to 0 to reduce the cost function.

5. Use architectures that generalize well

We can use general architecture instead of building an architecture that classify a specific object. For example , instead of building a network which classifies a cat in the image with reasonable illumination, it would be efficient to build a network which can recognize a cat even when it is occluded. Instead of obtaining a 100% accuracy on train set, it would be better to aim for a higher testing accuracy.

- **CNN works better than traditional methods**

Neural networks learns on the way during image classification whereas traditional methods are not so flexible and are constrained to learning certain parameters like an edge in the image. It might not necessarily know how to differentiate between an eye and a nose in an image, whereas a neural network can learn the difference when it is classifying by updating its filter coefficients.

CNN is involved deeply in feature extraction when classifying images as compared to traditional image processing algorithms. In CNN the image is first broken into important features like shapes, edges etc. These are then fed into a fully-connected or a dense layer.

The advantage of using a CNN is it is robust compared to traditional image processing algorithms, that is it can be trained to classify several type of objects in an image with reasonable accuracy. The pro is that the filters are learned as and when it is applied to extract a feature map. It has the advantage of using other layers like max-pooling to reduce redundancy

of features. It also takes into account the local region around it during classification as and when filter convolutions are applied.

The downside of using CNN would be that you need considerable amount of data to train the model to recognize and classify relevant features. The downside of using classical image processing for image classification would be to manually design filters which can detect and classify an image with reasonable accuracy.

Problems in image classification -

- Viewpoint variation

An object in an image to classify, for example, a cat as shown above can be taken in different angles, but it fundamentally still remains a cat.

- Bad illumination

Our computer might not be able to identify the cat in the image above as it is not highlighted as the main object.

- Occlusions

Algorithms should be robust enough to classify an object using only the main features of it.

- Background clutter

There is a person in front of the logs but our computer fails to identify it because of too much background noise.

CNN's are typically used to solve problems mentioned above and have better accuracies as compared to traditional image processing algorithms (like parameteric linear classifier, KNN). The convolutional layers of CNN form the heart of it as it is robust and flexible enough to learn and update its kernel weights on the fly.

- **Loss function and Back propagation algorithm**

Error is usually defined to be the difference between the expected output and the predicted output. It is given by :

$$\mathcal{L}(\mathcal{Y}) = \mathcal{Y} - \hat{\mathcal{Y}}$$

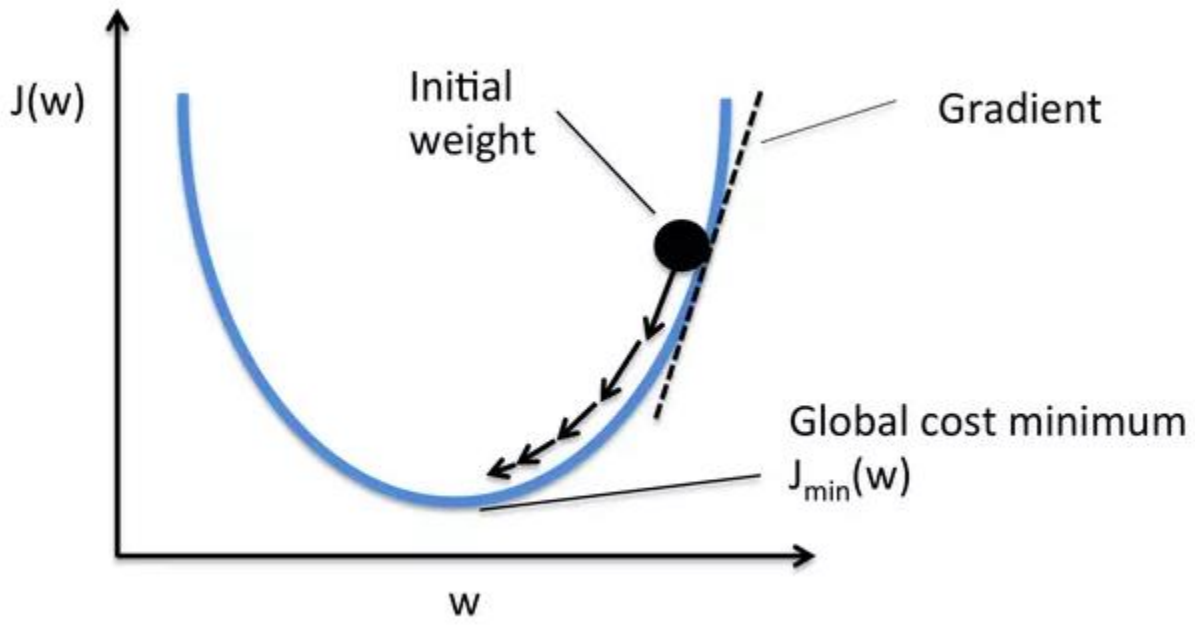



Fig : Minimizing Loss function approach(Gradient descent)

J(w) is the Loss function which we need to optimize. There are various loss functions which produce given errors for the same output and thus affects the performance and the operation of the model. The different loss functions are : Mean square error, cross entropy, hinge loss, L1 loss etc. The most popular one is the mean square error loss which calculates the squared difference between expected and obtained output.

The loss function or the cost function needs to be minimized and the most popular approach is the gradient descent algorithm which take steps to find the global minima of a convex optimization problem. The gradient calculated while performing gradient descent is used to update the weights that minimize the cost function.
\

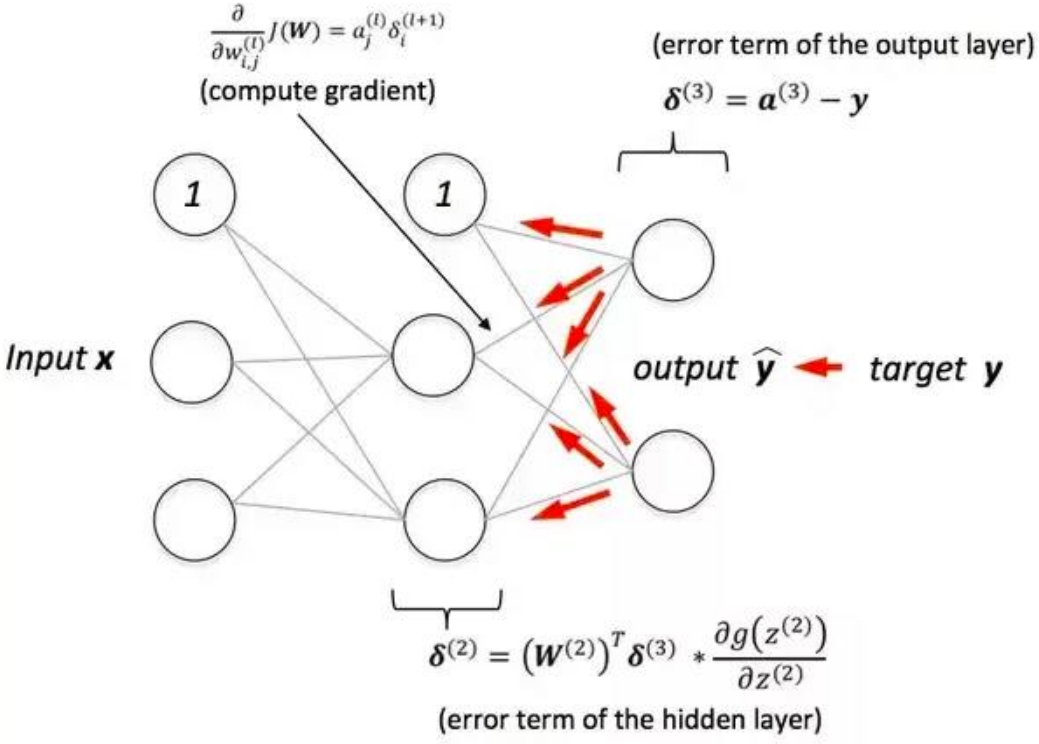



Fig : Backpropagation algorithm

Back propagation as the name suggests, minimizes the loss function and back propagates through the network to update the new weights. Our weight initializations might be completely off, and if we don't propagate back in the network and update, we will end up with huge errors. Hence, we need back propagation. Back propagation needs a known target output.
The error is calculated with the obtained and the target outputs and calculates the gradient of loss function considering the weights. It is usually used in supervised learning algorithms. The gradient is calculated in the last layer and back propagated to update weights.

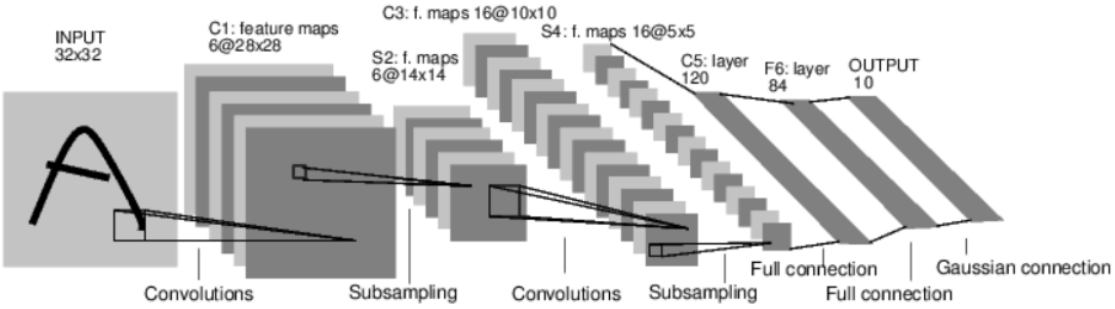
MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

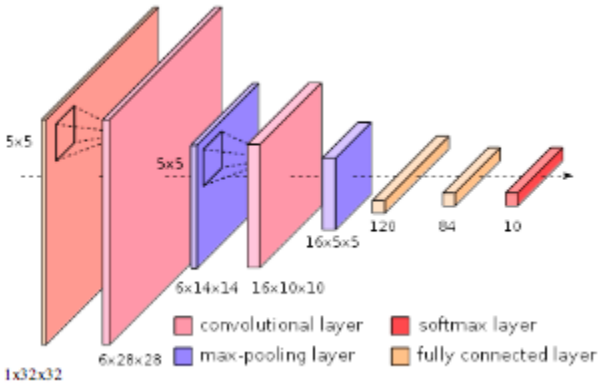
**PROBLEM 1B) Train LeNet-5 on MNIST dataset**

**1.2.B. APPROACH:**

The LeNet architecture[2] was introduced by LeCun in 1998. It is one of the basic deep learning structures. It has two conv layers followed by a max pooling layer to avoid overfitting. It has three fully connected layers. The kernel size for the convolution operations is (5,5). It has specified number of filters with the numbers being 6 and 16. The last fully connected layer reduced to 10 dimensions which is equivalent to the number of classes present in the mnist dataset.

The LeNet-5 structure is given below :




It can be summarized as :

```
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

The provided database is MNIST. It contains a training set of 60000 images and a testing set of 10000 images. The handwritten digits in the dataset is normalized by size and centered to fit the image dimension. This pre-processing is already done.

The parameters considered for varying and initializing is given in the table below :

OPTIMIZERS

1. SGD - Stochastic Gradient Descent Optimizer :
   SGD is very susceptible to variance and thus fluctuates between different minimas, but this ensures that it converges to the global minima. But the convergence is thus slowed down and sometimes overshoots.
2. Adadelta :
   Well suited for sparse data as it make larger updates for infrequent parameters and smaller ones for the frequent ones. We also don't have to mention the learning rate.
3. Adam :
   Converges faster and is rid of vanishing gradient problem.

4. Adagrad :
   No manual training of learning rate is required. It modifies the rate at each step.
5. RMSprop :
   Updates per-parameter based on the weight gradients and does well on noisy datasets.

- DROPOUT
  It is a regularization technique used to avoid overfitting of data. Here, neurons are randomly dropped. The parameter to be passed is the fraction of the input data to be dropped.
- KERNEL INITIALIZER
  This defines the way in which weights are being set to the different network layers. I have considered two ways -
  1. Glorot_normal : it draws samples from normal distribution based on the number of input and output parameters
  2. RandomUniform : generates samples in uniform distribution.
- EPOCHS : The number of iterations for the entire of the dataset.
- LEARNING RATE : A larger learning rate takes bigger steps towards the global minimum in view of converging faster, but it might get stuck with bigger steps and never reach it. A smaller learning rate takes small steps towards the minima, but the convergence is very slow. Hence, the learning rate should be chosen midground.
- DECAY : Amount by which the learning rate should decay over each update.
- MOMENTUM : Amount by which gradient descent is accelerated to reach the global minima and converge.
- BATCH SIZE: Number of samples considered for every update. Default is 32.

| | |
|---|---|
| ACTIVATION FUNCTIONS | Sigmoid, relu, tanh,linear, LeakyRelu |
| OPTIMIZERS | SGD, Adadelta, Adam, Adagrad, RMSprop |
| DROPOUT LIST | 0.2,0.5 |
| KERNEL INITIALIZATION LIST | Glorot_uniform, RandomUniform |
| EPOCHS | 100 |
| Learning rate decay | 0.01,0.1,0.5 |

I have performed this on Google Collab as the dataset size is huge and a lot of iterations were to be done. Here it took only 4 seconds to run for every epoch.

```
[ ] !python3 drive/CNN/hw_4_prob_1_b_manasa_tanh_400.py

/usr/local/lib/python3.6/dist-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` i
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
WARNING:tensorflow:Variable *= will be deprecated. Use variable.assign_mul if you want assignment to the variable value or 'x = x * y' if you want a new p
Train on 60000 samples, validate on 10000 samples
Epoch 1/100
52736/60000 [=========================>....] - ETA: 4s - loss: 1.1769 - acc: 0.649060000/60000 [==============================] - 42s 697us/step - loss: 1
Epoch 2/100
60000/60000 [==============================] - 41s 690us/step - loss: 0.4354 - acc: 0.8790 - val_loss: 0.3089 - val_acc: 0.9180
Epoch 3/100
60000/60000 [==============================] - 41s 689us/step - loss: 0.3574 - acc: 0.8976 - val_loss: 0.2662 - val_acc: 0.9245
Epoch 4/100
28160/60000 [=============>................] - ETA: 20s - loss: 0.3261 - acc: 0.906560000/60000 [==============================] - 41s 691us/step - loss:
Epoch 5/100
60000/60000 [==============================] - 42s 696us/step - loss: 0.2934 - acc: 0.9146 - val_loss: 0.2236 - val_acc: 0.9352
Epoch 6/100
60000/60000 [==============================] - 42s 697us/step - loss: 0.2775 - acc: 0.9196 - val_loss: 0.2109 - val_acc: 0.9378
Epoch 7/100
24576/60000 [===========>..................] - ETA: 22s - loss: 0.2688 - acc: 0.919960000/60000 [==============================] - 42s 700us/step - loss:
Epoch 8/100
60000/60000 [==============================] - 42s 698us/step - loss: 0.2543 - acc: 0.9250 - val_loss: 0.1949 - val_acc: 0.9416
Epoch 9/100
60000/60000 [==============================] - 42s 698us/step - loss: 0.2467 - acc: 0.9280 - val_loss: 0.1889 - val_acc: 0.9434
Epoch 10/100
22528/60000 [==========>...................] - ETA: 24s - loss: 0.2408 - acc: 0.929360000/60000 [==============================] - 42s 698us/step - loss:
Epoch 11/100
60000/60000 [==============================] - 42s 697us/step - loss: 0.2347 - acc: 0.9298 - val_loss: 0.1795 - val_acc: 0.9463
Epoch 12/100
60000/60000 [==============================] - 42s 697us/step - loss: 0.2309 - acc: 0.9324 - val_loss: 0.1756 - val_acc: 0.9475
Epoch 13/100
21504/60000 [=========>....................] - ETA: 24s - loss: 0.2258 - acc: 0.934560000/60000 [==============================] - 42s 697us/step - loss:
Epoch 14/100
60000/60000 [==============================] - 42s 698us/step - loss: 0.2236 - acc: 0.9334 - val_loss: 0.1693 - val_acc: 0.9489
```

Every iteration would save two files - model history and then the parameters with it. There are 60 iterations, so there was 120 files at the end of running the code for a single activation function.

I have considered activation function as the deciding criteria for the efficiency of a model. I have kept the activation function constant and have varied the remaining parameters given above.So there were 60 combinations of the above parameters and the best ones obtained for the particular activation function is reported below. Therefore, I have 5 cases :

Case 1: ACTIVATION FUNCTION : LINEAR
The maximum test accuracy was reached for :

| Activation function | Dropout | Optimizer | lr_decay_val | Weight initialization | Train loss | Train accuracy | Test loss | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| Linear 113 | 0.2 | Adam | 0.01 | Random Uniform | 0.028303033 | 0.991583345 | 0.034228269 | 0.98891 |

Case 2: ACTIVATION FUNCTION : RELU
The maximum test accuracy was reached for :

| Activati | Dropout | Optimiz | lr_decay | Weight | Train | Train | Test | Test |
|---|---|---|---|---|---|---|---|---|

| on function | | er | _val | initialization | loss | accuracy | loss | accuracy |
|---|---|---|---|---|---|---|---|---|
| ReLU 347 | 0.2 | RMSprop | 0.01 | Random Uniform | 0.008637069 | 0.997666567 | 0.020744673 | 0.9933 |

Case 3: ACTIVATION FUNCTION : TANH

The maximum test accuracy was reached for :

| Activation function | Dropout | Optimizer | lr_decay_val | Weight initialization | Train loss | Train accuracy | Test loss | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| tanh 413 | 0.2 | Adam | 0.01 | Random Uniform | 0.001738894 | 0.999766667 | 0.027259979 | 0.9906 |

Case 4: ACTIVATION FUNCTION : SIGMOID

The maximum test accuracy was reached for :

| Activation function | Dropout | Optimizer | lr_decay_val | Weight initialization | Train loss | Train accuracy | Test loss | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| Sigmoid 512 | 0.2 | Adam | 0.01 | glorot_uniform | 0.024876834 | 0.99265 | 0.032449454 | 0.9890 |

Case 5: ACTIVATION FUNCTION : LeakyReLU

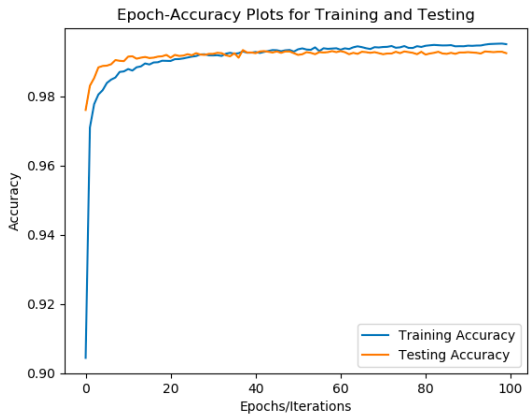
The maximum test accuracy was reached for :

| Activation function | Dropout | Optimizer | lr_decay_val | Weight initialization | Train loss | Train accuracy | Test loss | Test accuracy |
|---|---|---|---|---|---|---|---|---|
| LeakyReLU13 | 0.2 | Adam | 0.01 | Random Uniform | 0.00491711 | 0.998916652 | 0.019719298 | 0.99356 |

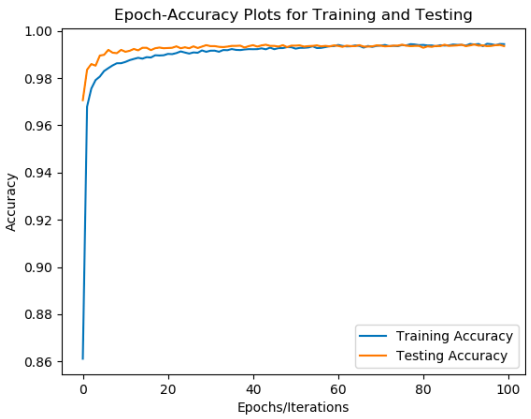
Below are the best testing accuracy (>0.99) plots obtained for the LeNet 5 with the network parameters varied. Refer to the index in the .csv image to find the parameters.

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

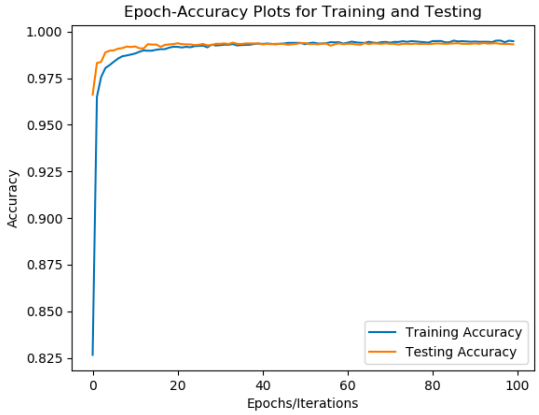
## 1.3.B RESULTS : EPOCH-ACCURACY PLOTS



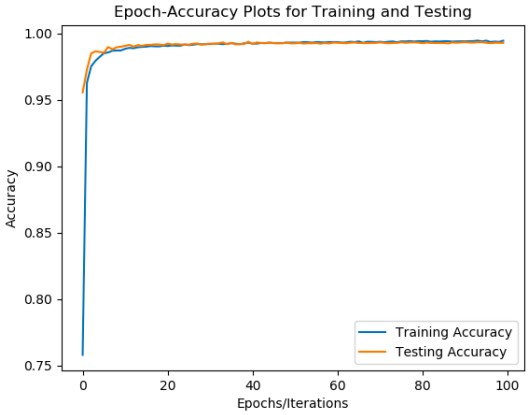

12 - Index



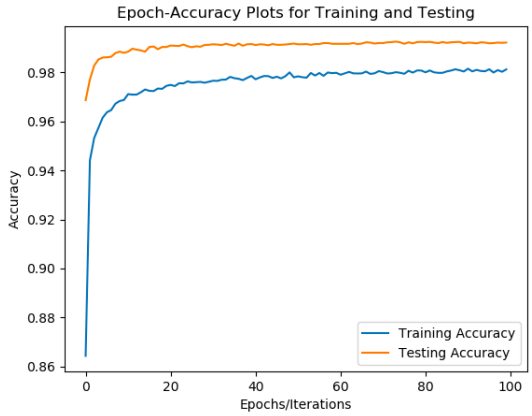

13 - Index

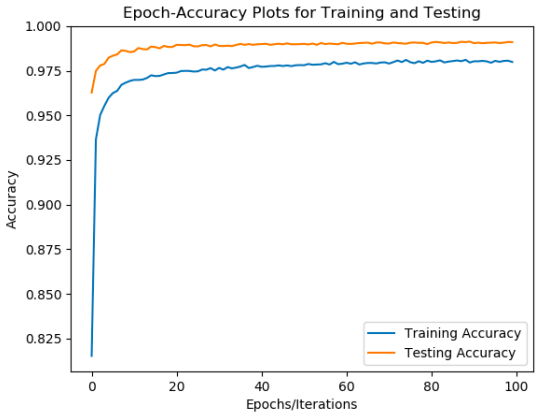



24- Index

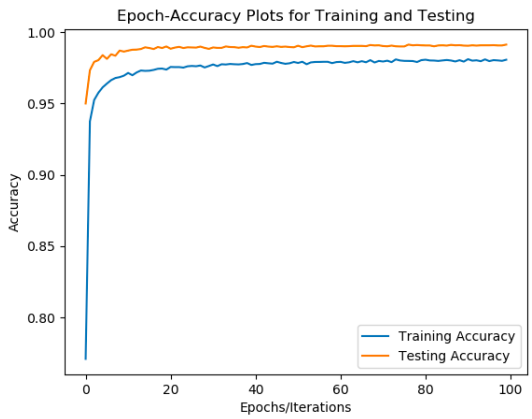
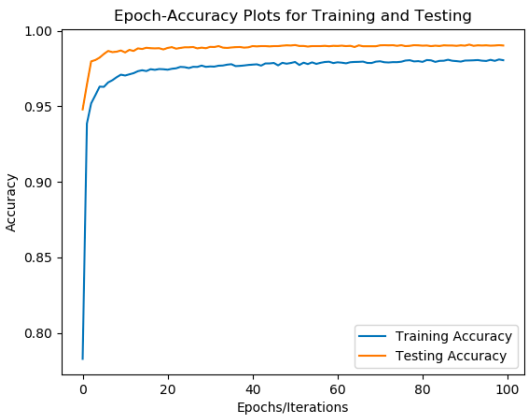
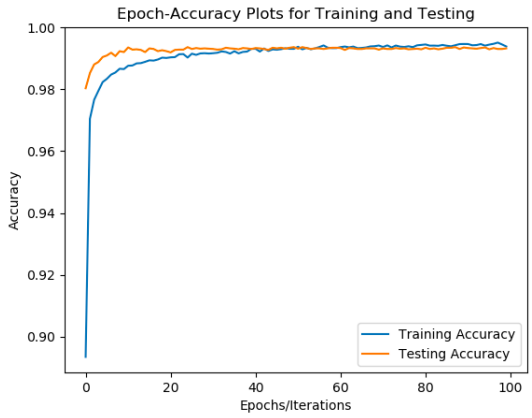
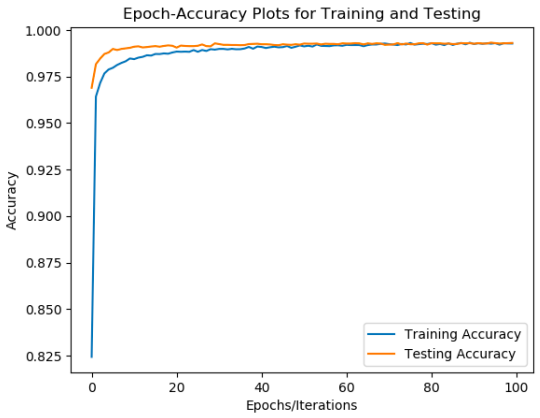
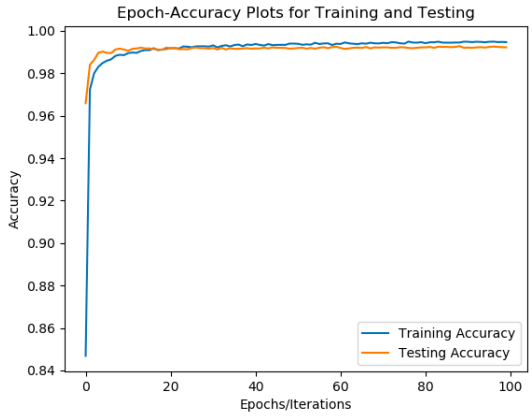
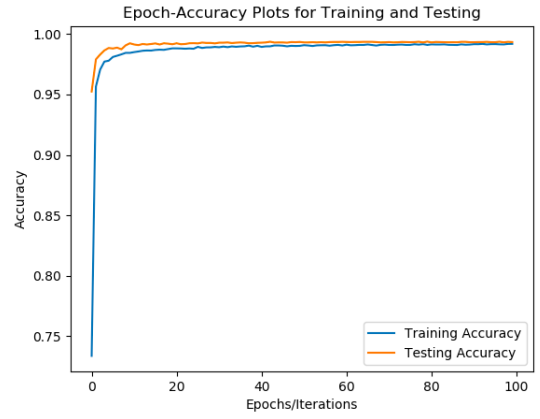



25- Index

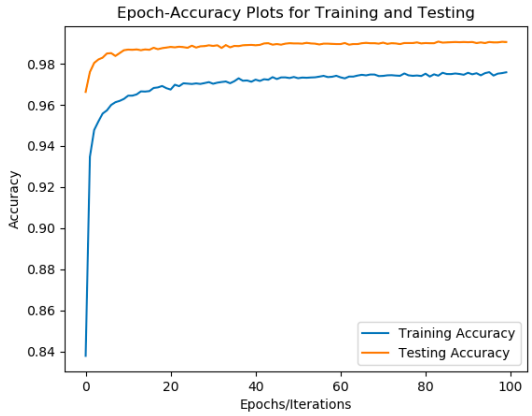


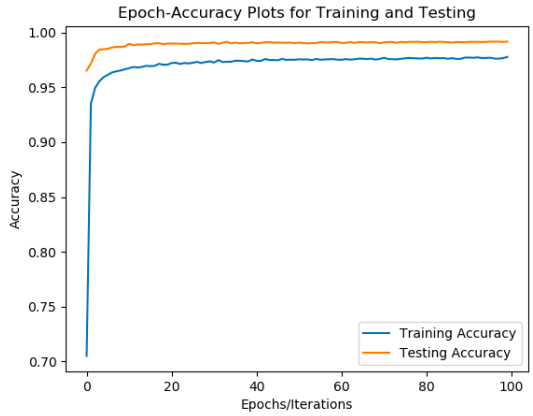
MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

| 42- Index | 43- Index |
|---|---|
|  |  |
| 54- Index | 55- Index |
|  |  |
| 312- Index | 313- Index |
|  |  |
| 324 - Index | 325- Index |

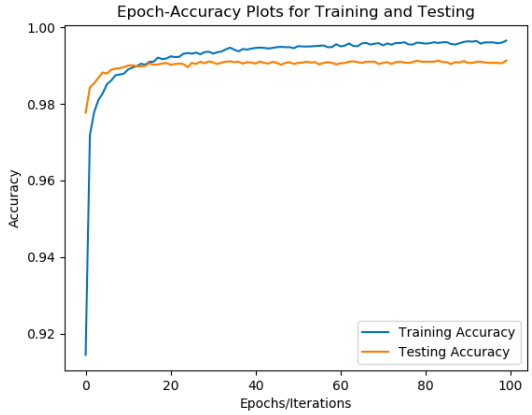
MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu



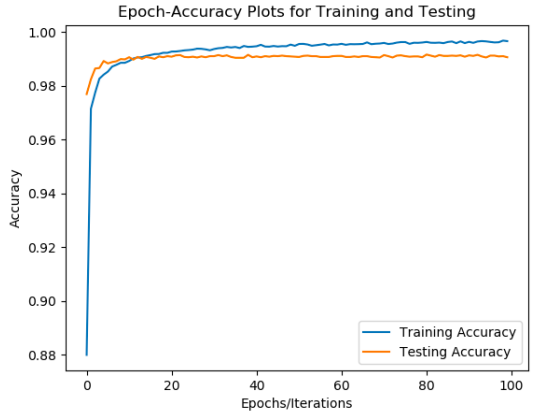

342- Index



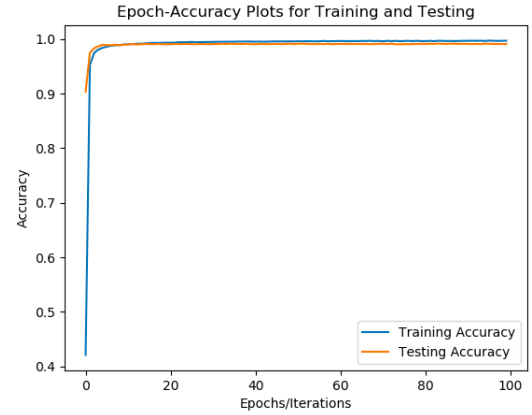

354- Index



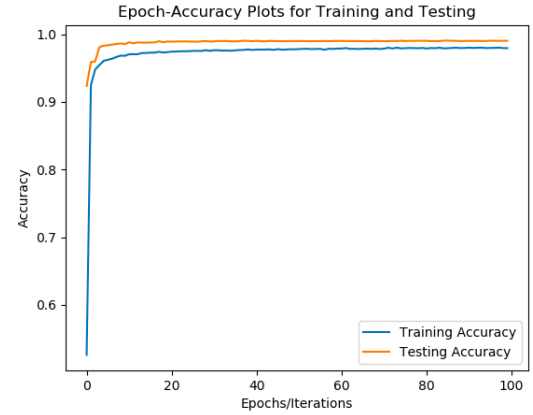

412- Index




413- Index




424- Index




454- Index

The 5 different parameter settings are shown below :

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

| Index | Iteration | Activation | Dropout | Optimizer | lr_decay | Weight_initilizations | Train_loss | Train_Accuracy | Test_loss | Test_Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 12 | LeakyReLU | 0.2 | Adam | 0.01 | glorot_uniform | 0.004073934 | 0.999066666 | 0.024203963 | 0.9924 |
| 13 | 13 | LeakyReLU | 0.2 | Adam | 0.01 | RandomUniform | 0.0049171 | 0.998916652 | 0.019719298 | 0.99356 |
| 24 | 24 | LeakyReLU | 0.2 | RMSprop | 0.01 | glorot_uniform | 0.004041373 | 0.9991 | 0.02181558 | 0.9931 |
| 25 | 25 | LeakyReLU | 0.2 | RMSprop | 0.01 | RandomUniform | 0.00521661 | 0.9986 | 0.021575079 | 0.993 |
| 42 | 42 | LeakyReLU | 0.5 | Adam | 0.01 | glorot_uniform | 0.019005346 | 0.9943 | 0.023402063 | 0.9922 |
| 43 | 43 | LeakyReLU | 0.5 | Adam | 0.01 | RandomUniform | 0.021045933 | 0.993516667 | 0.026352141 | 0.9912 |
| 54 | 54 | LeakyReLU | 0.5 | RMSprop | 0.01 | glorot_uniform | 0.020891074 | 0.99373 | 0.026997171 | 0.9913 |
| 55 | 55 | LeakyReLU | 0.5 | RMSprop | 0.01 | RandomUniform | 0.022677565 | 0.9929 | 0.028585653 | 0.9902 |
| 312 | 12 | relu | 0.2 | Adam | 0.01 | glorot_uniform | 0.003741682 | 0.999183333 | 0.021457492 | 0.9933 |
| 313 | 13 | relu | 0.2 | Adam | 0.01 | RandomUniform | 0.006532256 | 0.99845 | 0.019687753 | 0.9931 |
| 324 | 24 | relu | 0.2 | RMSprop | 0.01 | glorot_uniform | 0.003930898 | 0.99904 | 0.022094381 | 0.9922 |
| 325 | 25 | relu | 0.2 | RMSprop | 0.01 | RandomUniform | 0.008637019 | 0.997666667 | 0.020744773 | 0.9934 |
| 342 | 42 | relu | 0.5 | Adam | 0.01 | glorot_uniform | 0.025488103 | 0.992833333 | 0.02981473 | 0.9906 |
| 354 | 54 | relu | 0.5 | RMSprop | 0.01 | glorot_uniform | 0.023194224 | 0.99341 | 0.027684475 | 0.9919 |
| 412 | 12 | tanh | 0.2 | Adam | 0.01 | glorot_uniform | 0.001865573 | 0.999716667 | 0.028873648 | 0.9914 |
| 413 | 13 | tanh | 0.2 | Adam | 0.01 | RandomUniform | 0.001738893 | 0.999766665 | 0.027259979 | 0.9906 |
| 424 | 24 | tanh | 0.2 | RMSprop | 0.01 | glorot_uniform | 0.001350198 | 0.99975 | 0.029994731 | 0.9908 |
| 454 | 54 | tanh | 0.5 | RMSprop | 0.01 | glorot_uniform | 0.018427967 | 0.9946 | 0.031867354 | 0.9902 |

Check the indexes above with the selected network parameters for the accuracy.

**1.4.B. DISCUSSION :**

From the above plots and tables, we can see that the maximum testing accuracy obtained was 0.99356 (99.356%) for the parameters - LeakyReLU(Activation), 0.2(Dropout), Adam(Optimizer), 0.01(lr_decay), RandomUniform(Weight initialization),0.0049171(Train loss), 0.998916652(Train accuracy),0.019719298(Test loss),0.99356(Test accuracy).

We can see from the above plots that for testing accuracy > 0.99, only LeakyReLU, relu, tanh activations achieved it. The best two optimizers was Adam and RMSprop, and the optimum learning rate and decay rate was 0.01. We can also see from the plots and the table that accuracy > 0.95 was achieved in very few iterations. The iteration number is also provided in the table. This tells us that our model with the network parameters work efficiently.

We can observe from the plot that testing accuracy is above the training accuracy. This tells us that the model is not overfitting.

**PROBLEM 1 C )Improve the LeNet-5 for MNIST dataset**

**1.2.C. APPROACH [6]:**

The different combinations of activation functions, optimizers were dealt exhaustively in 1b). Hence, for 1c, the activation function remains constant - LeakyReLU, with the changes introduced in the structure.

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

Test 1 :

The structure is changed as follows -

The dense layer now has the expected output shape(units ) = 32, hence the network is growing in the third dimension.

As we are using a Conv2D filter of size (3,3), the input shape (28,28) is now changed to (26,26).

The structure is given as follows:

```
/Users/mansi/tensorflow/lib/python2.7/site-packages/h5py/__init__.py:36: FutureW
reated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               589952
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 730,602
Trainable params: 730,346
Non-trainable params: 256
```

| | |
|---|---|
| Activation | LeakyReLU |
| Optimization | RMSprop |
| Number of Classes | 10 |
| epoch | 20 |
| Input_shape | (28,28,1) |

| | |
|---|---|
| Dropout | 0.2, 0.25 |
| Convolution kernel size | (3,3) |
| Fully connected Layer Activation | Softmax |
| loss | categorical_crossentropy |
| metrics | accuracy |
| Test Accuracy | 99.689% |

The network has grown in depth. I have two convolutional layers with kernel size = (3,3) with activation = 'relu', as opposed to LeNet which has a single Convolutional Layer with filter size =(5,5). Max-pooling layer is introduced right after two convolutional layers to downsample the obtained activation points. An introduction of Dropout, assures that the model is not overfitting. Dropout drop the weights of the nodes with a probability of 1-p. This is randomized, hence helps in reducing overfitting. Then there are two more convolutional layers, followed by a max-pooling, and a dropout. It has been proven that max-pooling and dropout together increases the testing accuracy. Then it is followed by a convolutional layer and a dropout layer. Then the output is flattened and applied to a Fully-connected (Dense) network with activation as a relu function. The batch_normalization is applied and this reduces the parameters to 512. This is followed by the last dropout layer and then the fully connected with the activation function of softmax.
The obtained test accuracy was 99.689%

Test 2:

```
/Users/mansi/tensorflow/lib/python2.7/site-packages/h5py/__init__.py:36: FutureW
reated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 32)        25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               803072
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 887,530
Trainable params: 887,530
Non-trainable params: 0
_________________________________________________________________
```

| | |
|---|---|
| Activation | ReLU |
| Optimization | RMSprop |
| Number of Classes | 10 |
| epoch | 30 |
| Input_shape | (28,28,1) |
| Dropout | 0.25,0.5 |
| Convolution kernel size | (5,5) |

| | |
|---|---|
| Fully connected Layer Activation | Softmax |
| loss | categorical_crossentropy |
| metrics | accuracy |
| Test Accuracy | 99.611% |

The network has grown in depth. I have two convolutional layers with kernel size = (5,5) with activation = 'relu', as opposed to LeNet which has a single Convolutional Layer with filter size =(5,5). Max-pooling layer is introduced right after two convolutional layers to downsample the obtained activation points. An introduction of Dropout, assures that the model is not overfitting. Dropout drop the weights of the nodes with a probability of 1-p. This is randomized, hence helps in reducing overfitting. Then there are two more convolutional layers, followed by a max-pooling, and a dropout. It has been proven that max-pooling and dropout together increases the testing accuracy. Then the output is flattened and applied to a Fully-connected (Dense) network with activation as a relu function. This is followed by the last dropout layer and then the fully connected with the activation function of softmax.
The obtained test accuracy was 99.611%

Test 3 :

```
/Users/mansi/tensorflow/lib/python2.7/site-packages/h5py/__init__.py:36: Fu
reated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 594,922
Trainable params: 594,922
Non-trainable params: 0
```

| | |
|---|---|
| Activation | ReLU |
| Optimization | Adam |
| Number of Classes | 10 |

| | |
|---|---|
| epoch | 5 |
| Input_shape | (28,28,1) |
| Dropout | 0.2 |
| Convolution kernel size | (3,3) |
| Fully connected Layer Activation | Softmax |
| loss | categorical_crossentropy |
| metrics | accuracy |
| Test Accuracy | 99.467% |

The network has grown in depth. I have two convolutional layers with kernel size = (3,3) with activation = 'relu', as opposed to LeNet which has a single Convolutional Layer with filter size =(5,5). Max-pooling layer is introduced right after two convolutional layers to downsample the obtained activation points. Two other Convolutional Layer with activation 'relu' follows it. Then another max-pooling layer is introduced.Then the output is flattened and applied to a Fully-connected (Dense) network with activation as a relu function. An introduction of Dropout, assures that the model is not overfitting. Dropout drop the weights of the nodes with a probability of 1-p. This is randomized, hence helps in reducing overfitting. This is followed by the fully connected with the activation function of softmax.
The obtained test accuracy was 99.467%

Test 4 :

```
(tensorflow) Manasas-MacBook-Air:models mansi$ python model.py
/Users/mansi/tensorflow/lib/python2.7/site-packages/h5py/__init__.py
reated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
2018-04-28 13:04:47.371565: I tensorflow/core/platform/cpu_feature_g
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 16)        64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 16)        2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 32)        4640
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 32)        128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 32)          9248
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10250
=================================================================
Total params: 814,970
Trainable params: 814,778
Non-trainable params: 192
_________________________________________________________________
```

| | |
|---|---|
| Activation | ReLU |
| Optimization | Adam |
| Number of Classes | 10 |
| epoch | 20 |
| Input_shape | (28,28,1) |
| Dropout | 0.25,0.5 |
| Convolution kernel size | (3,3) |
| Fully connected Layer Activation | Softmax |
| loss | categorical_crossentropy |
| metrics | accuracy |
| Test Accuracy | 99.319% |

The network has grown in depth. I have two convolutional layers with kernel size = (3,3) with activation = 'relu', as opposed to LeNet which has a single Convolutional Layer with filter size =(3,3). Every convolutional layer is followed by a batch-normalization layer. Max-pooling layer is introduced right after them to downsample the obtained activation points. An introduction of Dropout, assures that the model is not overfitting. Dropout drop the weights of the nodes with a probability of 1-p. This is randomized, hence helps in reducing overfitting. Then there are two more convolutional layers, followed by a max-pooling, and a dropout. It has been proven that max-pooling and dropout together increases the testing accuracy. Then it is followed by two convolutional layers with batch normalization layer. Then there is a max-pooling layer introduced. Then the output is flattened and applied to a Fully-connected (Dense) network with activation as a relu function. It is again followed by a dropout and one more fully connected and a dropout layer. This is followed by the fully connected with the activation function of softmax.
The obtained test accuracy was 99.319%

Test 5 :

```
  model.add(Dense(10, activation = 'softmax', init='he_normal')) #Last
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 12)        312
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 12)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 25)          7525
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 25)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 180)               72180
_________________________________________________________________
dropout_1 (Dropout)          (None, 180)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               18100
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 99,127
Trainable params: 99,127
Non-trainable params: 0
_________________________________________________________________
```

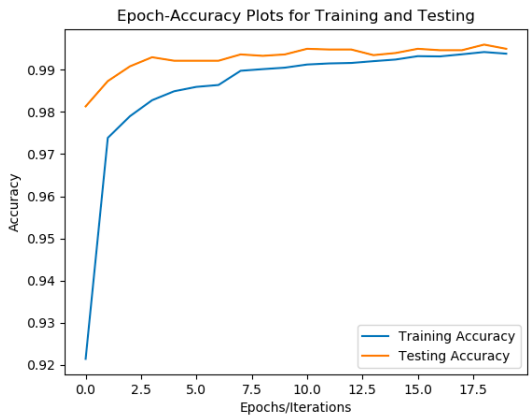
| | |
|---|---|
| Activation | ReLU |
| Optimization | Adam |
| Number of Classes | 10 |
| epoch | 20 |
| Input_shape | (28,28,1) |
| Dropout | 0.5 |
| Convolution kernel size | (5,5) |
| Fully connected Layer Activation | Softmax |
| loss | categorical_crossentropy |
| metrics | accuracy |

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

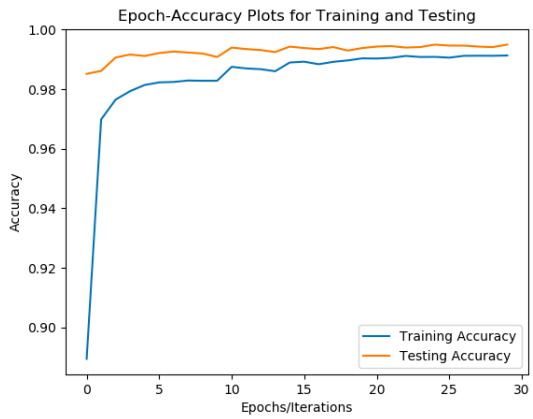
| | |
|---|---|
| Test Accuracy | 98.677% |

The network has grown in depth. I have two convolutional layers with kernel size = (3,3) with activation = 'relu', as opposed to LeNet which has a single Convolutional Layer with filter size =(5,5). Every convolutional layer is followed by a max-pooling layer. Max-pooling layer is introduced right after them to downsample the obtained activation points. Then the output is flattened and applied to a Fully-connected (Dense) network with activation as a relu function. It is again followed by a dropout and one more fully connected and a dropout layer. An introduction of Dropout, assures that the model is not overfitting. Dropout drop the weights of the nodes with a probability of 1-p. This is randomized, hence helps in reducing overfitting. This is followed by the fully connected with the activation function of softmax.
The obtained test accuracy was 98.68%

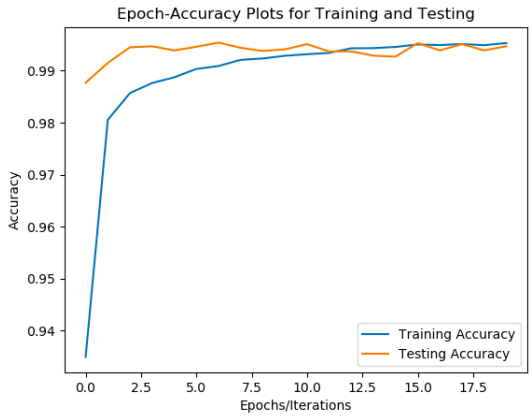
**1.4.C. DISCUSSION :**



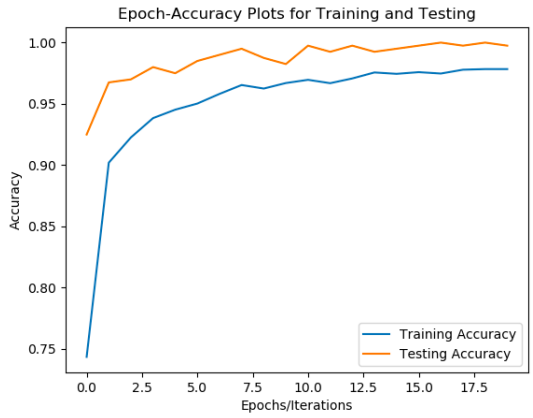

Test 1



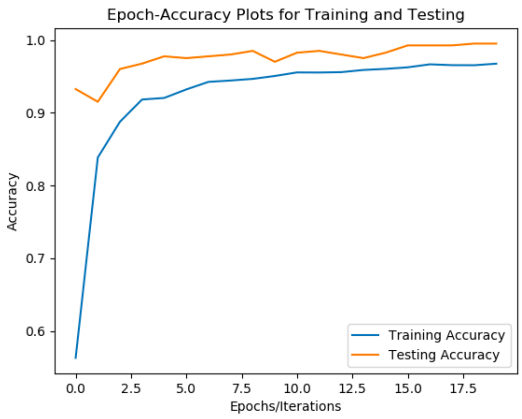

Test 2




Test 3




Test 4

| | |
|---|---|
|  <br>Test 5 | |

| | |
|---|---|
| Test 1 - test accuracy | 99.689% |
| Test 2 - test accuracy | 99.611% |
| Test 3 - test accuracy | 99.467% |
| Test 4 - test accuracy | 99.319% |
| Test 5 - test accuracy | 98.677% |

Alternative variations of LeNet-5 produced the maximum test accuracy of 99.356%. But our test 1, test 2, and test 3 structures produced better outputs than the best combination of LeNet. The maximum test accuracy obtained was from the test 1 structure which achieved the accuracy of 99.689%.

As we can see, the structure got deeper with the addition of multiple convolutional layers and succeeded with a batch normalization layer. Dropout and Max-pooling was also introduced to make sure that the model doesn't overfit, and from the accuracies we can see that the model works better. The fully connected layer was also followed by a batch-normalization and dropout layers to make sure that the model does not produce a lot of errors.

We can also observe from the above plots for test1 that the testing accuracy is much better than the training accuracy over 20 epochs and also the training accuracy is not dropping over the epochs. Hence the model is a better fit.

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

## PROBLEM 2 - APPLICATION OF SAAK TRANSFORM TO MNIST DATASET

### 2.1. ABSTRACT :

Introduction of CNN for image classification was a breakthrough for image classification problems. Saak transforms, even though it uses neural networks, have a strong statistical foundation. Using the multi-stage Saak transform, we can obtain multiple representations of a larger image in both pure spatial and spectral domain. Multi-stage Saak coefficients are computed and are used to build the feature vectors of the representation class. These feature vectors typically have higher discriminant power. The Saak transform kernels are derived from second order statistics and hence are different from the CNN learned filters.

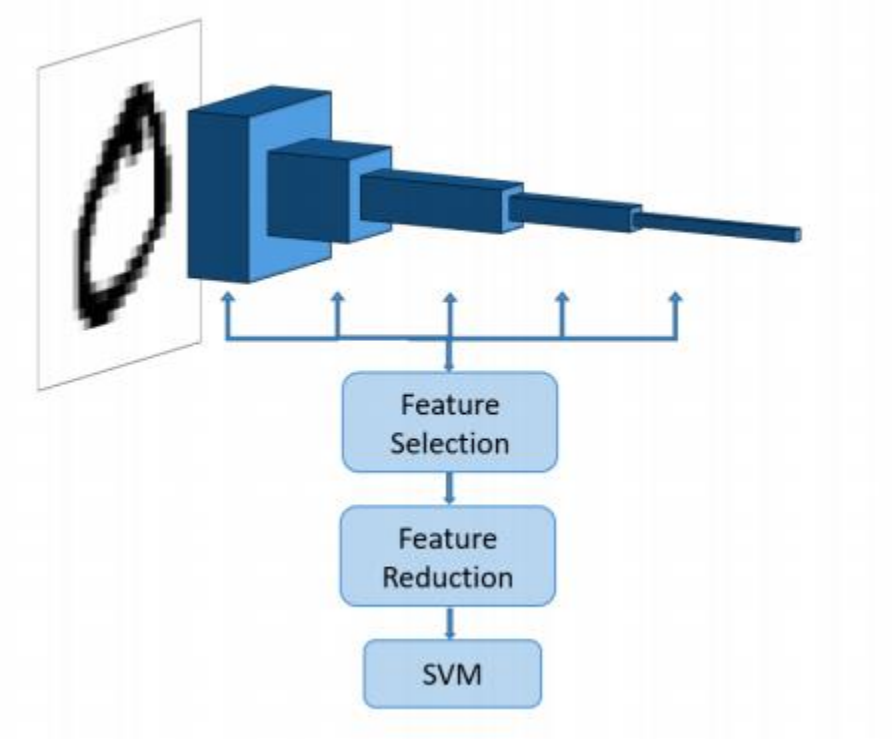



Fig: Saak transform working

It is primarily made up of 3 steps:

1. Saak coefficients generation :
    - Select the non-overlapping blocks spatially with a size of 2x2. As we want to retain features which represent the class and have a higher discriminant power, low variance patches are eliminated.
    - PCA or KLT is performed on the obtained patch to get the PCA matrix. The advantage of Saak Transform is that all the calculations and transforms occur in the spatial domain and an added advantage is that is that it taken place in the

orthonormal domain. Thus, this reduces the dimensions by half in both horizontal and vertical directions.

- The next step would be to augment the transform kernels. This is done by projecting onto the augmented kernel set.
- Here, the signed KLT is converted to a positional KLT. Thus this results in doubling of the spectral dimensions.
- Apply the non-linear activation function ReLU to the feature vector.
- Repeat the same for all the 5 stages of the multi stage Saak transform.
- Resulting vector dimension will be reduced from 2048 to 1500.
- At the end of this, we would obtain a feature vector well representing the class with the important components.

2. Saak coefficients selection :
   Here, f-test is performed to obtain the best (approx.1000)features based on the f-test scores.
   The F-test score is calculated using :

$$F = \frac{\text{between-group variability (BGV)}}{\text{within-group variability (WGB)}}.$$

3. Classification :
   Classify the obtained features using SVM and RandomForestClassifer

In the multi-stage Saak transform, a given image is split into 4 quadrants such that the root is the original node. All the decomposed nodes become the leaf. This is a recursive process and it keeps breaking down until the quadtree reaches a size of 1x1.In every stage, the Saak coefficients ar generated. The best features are selected using the maximum eigen values, and dimensionality reduction and compression techniques like PCA and then classified using SVM and Random Forest.

## PROBLEM 2.A) Comparison between Saak Transform and CNN architecture

### 2.2.A) APPROACH :

CNN and SAAK[4] -
Both CNN and Saak are made up of Neural networks and are applied with a similar activation function like ReLU to introduce non-linearities in the network.Working of a CNN and Saak transform is on the same foundation, but their approach is fundamentally different. It can be compared with one another in 4 different aspects -

- End-to-end Optimization versus Modular Design

It is mainly divided into two steps - feature extraction module and the classification module.

CNN exhibits 3 main weaknesses - robustness against perturbation, portability among different datasets, and scalability against class number . Every addition or deletion of a class affects the performance of CNN significantly. Saak is more robust and is desensitized towards variation in classes. As PCA is applied, the prominent features are retained and thus Saak coefficients are not affected majorly.

- Generative Model versus Inverse Transform

No official inverse transform has been presented for CNN. We can resynthesize images based on GAN and VAE.

However, for Saak transform, it is backed up by traditional mathematical and signal processing ideas. The kernels of Saak are orthonormal and hence the inverse can be resynthesized easily. Multi-stage Saak transform pipeline can be broken down for low resolution and high resolution stages. Multiple combinations of the stages can be tried to result high resolution images.

- Theoretical foundation

Most commonly the working of a CNN is referred to as a "Black box" operation. There is significant research in the field to visualize the filters at every stage. Many empirical experiments have been done and many scientists are diving into the field, but the field is largely driven by intuition. Whereas Saak is completely founded on Linear Algebra and statistics, hence it is mathematically elegant and every step and result is accompanied with a proof.

- Filter weight determination, feature selection and others

The filter weights are learned when the network is trained in CNN, and these weights are updated in the backpropagation step iteratively, but the operation of the Saak transform adopts a primarily different approach. The multi-stage Saak transform automates the process of determining the weights of the kernels. It is a one pass feedforward algorithm. They are determined by the second order statistics of input vectors at every stage of the network.

Another fundamental difference is in implementation. CNN applies convolution on overlapping blocks and then apply spatially pool to reduce dimension. Whereas, Saak transform is applied to non-overlapping blocks and no spatial pooling is needed.

Saak also needs a comparatively less amount of training data as compared to CNN to resolve at almost the same classification accuracy. It also does not need labels to train, and as it is spatially better, we can change the number of dimensions we need as thus is more robust. A big advantage is that Saak transform does not have to compute its coefficients again if new datasets are added, whereas CNN would have to train all over again.

**PROBLEM 2.B AND C) Application of Saak transform to MNIST dataset and Error Analysis**

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

**2.2.B AND C) APPROACH :**

Subspace Approximation with augmented Kernels(Saak) is a data driven transform built on concepts of a CNN. As the transform is centered to orthonormal spaces, forward and inverse transform can be applied for image analysis and synthesis respectively. The inverse process is still not explored completely in the Neural Network domain. It is a multi-stage transform where at every stage only significant feature vectors are preserved.

I have used the online available Saak-tensorflow code. [5]
Procedure followed to tweak the code for Saak -

- Saak transforms calculates the anchor vectors and decides on the direction of maximum variance, but it is provided in the question paper to consider keeping the number of components for the 5 stages of Saak to be -

| Ith stage | components_to_keep |
|---|---|
| 0 | 3 |
| 1 | 4 |
| 2 | 7 |
| 3 | 6 |
| 4 | 8 |

So, hardcode these values in the util.py file inside the network folder. Pass 'i' as a parameter to function fit_anchor_vectors.

- Write the functions for calculating f_test and classifying using SVM or RandomForestClassifier
- I have used sklearns library function for f_test (f_classif). This is done on the training set only using the train-coef [from Saak transform] and mnist.train.labels. I am computing the number of selected features to be around 1000 dimension by setting the threshold. This gives us the best selected training features.
- Then PCA is applied to transform the selected features to the required n = 32,64, and 128 dimensions.
- To calculate the training accuracy, pass the reduced train features and mnist.train.labels to SVM and RandomForestClassifier. I have reported the scores I have obtained.
- To calculate the testing accuracy, pass the test features and mnist.test.labels to SVM and RandomForestClassifier. I have reported the scores I have obtained.

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

**2.2.B) DISCUSSION :**

Saak transform presents good outputs even when the training size is small. For example, for the first iteration, I considered the validation size to be 59000 and the train size = 1000 and pca dimension = 128. I got the testing accuracy about 90% which is a very good score for such a small training size. CNN would not have given me a better accuracy.
The best experience with Saak transform was that it could run on my CPU and I didn't have to use a GPU. The computation was also extremely fast.

The given number of components to use for PCA and then for classification is given in the question to be : 32,64,128 after reduction of original 2048 to 1500 and then to 1000.Validation size = 5000

| Training size | Number of components | Training accuracy using SVM | Testing accuracy using SVM | Training accuracy using RFC | Testing accuracy using RFC |
|---|---|---|---|---|---|
| 55000 | 32 | 0.992363636364 | 0.983 | 0.998709090909 | 0.930 |
| 55000 | 64 | 0.988890909091 | 0.982 | 0.998927272727 | 0.923 |
| 55000 | 128 | 0.980927272727 | 0.977 | 0.998981818182 | 0.899 |

The best classification accuracy is obtained by Number of components =32, and using SVM, the testing accuracy was 98.3%

I have used the online code [5] available which was already calculating all the 5 stages of Multi-stage Saak Transform with augmentation of kernels. I added the code for computing PCA again and then classifying using SVM and RandomForestClassifier.

To consider the trade-off between number of components used for computation vs accuracy, Saak proposes an easier approach for lesser number of computations but for a compensated accuracy. The best accuracy from problem 1 is 99.69% with the number of trainable parameters to be close to 9000000. The number of parameters considered for calculation in Saak is much more lesser given the accuracy is 98.3%.

Saak does not need backpropagation and labels for training and testing as it is a pure spatial-spectral algorithm and is hence faster and computationally effective. Also, Saak is invariant to

the change of data of data, whereas in CNN we would have to train the entire network again. Hence , Saak is more computationally efficient and robust than CNN.

**2.2.C) DISCUSSION :**

To analyze the performance of a network or a model, it is necessary to understand why the network might fail to classify a testing image. Here, we analyze the classification error rates arising from CNN and the Saak transform -
Considered classifiers are :
Classifier SVM with number of pca components = 32 with accuracy = 98.3%
Test 1 structure( from 1c) - CNN - accuracy = 99.62%

```
------------------
Count that Both CNN and SAAK misclassified:  18
Percentage of the MNIST DATASET that both missed: 0.18%
Image Labels of this category:  [ 660 1015 1040 1394 1902 2131 2183 2463 2655 3423 4177 4202 5938 6577
 6598 6626 8528 9730]

------------------
```

The percentage of testing data which both CNN and Saak classified correctly is 82%.
We can see that both CNN and Saak misclassified 18 out of the 10000 testing samples. The indexes are also shown. Hence, I decided to check the images which were misclassified :

| | |
|---|---|
|  Correct 9 - misclassified to 8 |  Correct 6 - Misclassified to 5 |

By looking at the misclassified outputs, it seems that some pre-processing technique should be applied to reduce the misclassification. There is also presence of noise seen in the images and hence a denoising before feeding to the classifier would present a better result.

MANASA MANOHARA
3722587777
HW#4- 04/29
manasama@usc.edu

```
----------
Count of CNN Missed:  38
Count of CNN Missed that SAAK Predicted Correctly:  20
About 52.6315789474% of CNN missed, SAAK predicted correctly
Image Labels of this category:  [ 413  675  717  939 1404 1439 1460 1879 2036 2448 2455 2598 3535 3763
 4861 5655 6559 8280 9643 9694]

----------
Count of SAAK Missed:  168
Count of SAAK Missed that CNN Predicted Correctly:  150
About 89.2857142857% of SAAK missed, CNN predicted correctly
Image Labels of this category:  [ 152  248  321  322  341  382  446  448  449  450  496  583  629  690
 708  721  741  883  925  948  952  960 1045 1063 1113 1195 1227 1233
1243 1248 1261 1300 1320 1329 1501 1523 1531 1550 1554 1582 1682 1710
1755 1791 2045 2071 2099 2110 2136 2190 2294 2300 2372 2388 2407 2415
2423 2489 2608 2649 2772 2811 2864 2897 2928 2940 2954 2996 3061 3074
3118 3334 3370 3476 3491 3504 3521 3559 3560 3598 3619 3768 3777 3797
3809 3812 3854 3903 3942 3969 3986 4066 4076 4079 4164 4225 4249 4290
4307 4370 4498 4501 4640 4741 4808 4824 4880 4881 4887 5458 5601 5643
5735 5836 5937 5956 5973 5974 5983 5998 6036 6167 6506 6561 6572 6599
6652 7217 7435 8095 8326 8340 8409 9010 9016 9025 9588 9635 9665 9680
9699 9746 9750 9769 9771 9793 9809 9840 9923 9945]
```

We can see that CNN missed 38 out of 10000 testing samples, and Saak predicted 20 of those misclassified samples correctly. Whereas, Saak missed 168 of testing samples in which CNN predicted 150 of them right. We can also see that CNN classified 89% of Saak misclassified samples.

The huge number of misclassified points in Saak can be attributed to it training on only 50000 samples whereas CNN trained for 60000 samples. Possibly, if we trained our Saak model for all the 60000 samples, we would have achieved a lesser classification error with respect to Saak.

Data augmentation can also be performed to obtain better testing accuracy. We can see that the digits direction vary a lot and if this rotation factor is also considered by augmentation, our models would have performed better. We can also consider our data to consider other affine transformations but to be normalized to the center of the image. We can see from the above image that a '9' was misclassified as 8, maybe the network found another hole with corresponded to '8'. So as another pre-processing step, it would be better to thin/shrink the image and feed the number of holes, bay, lakes also as a feature with the image.

Saak performs efficiently with minimal data compared to CNN as it selects the best feature dimensions using the F-test. We have reduced the feature size to 32 dimensions. Perhaps, if we had classified using all of the best dimensions from the f-test, our Saak transform would have performed better.
For the CNN model from test 1(from 1c), we can probably increase the depth of the network and add more non-linearities to encompass the misclassified samples.

We can also check different types of ANOVA tests instead of f-test like Tukey's Honestly Significant Different Test, Newman-Keuls Test, Bonferroni's Test, Dunn's Test to obtain the best discriminative feature dimensions.

But for a smaller train size, Saak works impressively faster than CNN and has a good accuracy.

**REFERENCES :**

1. CS231n : http://cs231n.github.io/
2. LeNet5 : http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
3. Saak Transform : https://arxiv.org/pdf/1710.10714.pdf
4. Saak Transform : https://arxiv.org/abs/1710.04176
5. Saak- Tensorflow online source code: https://github.com/morningsyj/Saak-tensorflow
6. Kaggle competitions : https://www.kaggle.com/c/
7. Keras documentation : keras.io