Muthulakshmi Chandrasekaran muthulac@usc.edu

USC ID : 4486180242

April 29, 2018

Homework #4

Image Recognition with Convolutional Neural Network and SAAK

Problem 1: CNN Training and its Application to the MNIST Dataset

Abstract and Motivation :

Image Classification is a key factor in almost all of the emerging applications. There are various methods of image classification – like using neurons, using support vector machines etc. The most popular form of image classification is the use of Neural Nets. They are used extensively because of the high performance. Comparatively, SAAK is also another method, that provides more advantages than the CNN.

Part A – CNN Architecture and Training

Approach :

Neural networks make predictions by learning the relation between data features and the responses recorded. A convolutional neural network can be defined as a neural network that consists of multiple replicas of a single neuron. This provides the advantage of training a neuron once and using it in different test cases. CNNs have very wide applications in the field of image processing. CNN is used for image classification as it performs image classification by looking at a few low – level features (like curves and edges) and builds up more abstract layers through convolutions.

Question 1: Description and Function of each of the CNN components

CNN is implemented via different architectures – one such architecture is the LeNet – 5[1], shown in Figure 1.1.

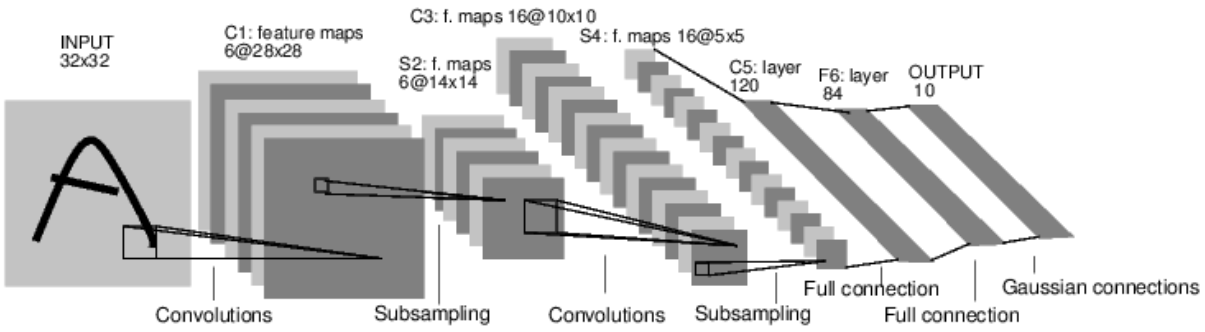


Figure 1.1 Lenet 5 Architecture

Every CNN has a series of layers – convolutional (non – linear) , pooling and fully – connected layers. Every input is passed through all these layers, and finally an output is obtained. LeNet5 consists of two convolutional layers, with two pooling layers, following each convolutional layer and two fully connected layers. This LeNet 5 architecture has 61706 parameters optimized over training. This architecture is a feed – forward Neural Network, meaning the output of every layer is obtained from the previous layer. The different layers are explained as follows.

Input Layer – The input layer provides input to the CNN. This architecture is fixed, i.e. it can accept only 32*32 sized input images. Different datasets can be validated in the input layer. The given data set MNIST has images of size 28*28.

Convolutional Layer – Basic layer of any CNN. This convolutional layer is specified by a set of kernels K and biases b per each kernel. These K kernels cover the entire image. This layer does convolution of each of the output (of the previous layer) with the kernels, and then adds bias to each convolved output. Each of the kernels is moved across the image, and the inner product of the kernel and the corresponding pixels is computed to find the value at each of the pixel location.

Each of the kernel here can be considered a feature identifier – that identifies features like simple straight lines, edges and curves. Convolutional Layer gives high values for features that are specified by the kernels, and almost zero values for features that are not specified by the kernels. The output of this convolutional layer is an activation map. For example, if the input to a convolutional layer is 32*32, and the filter of size 3*3 is applied, the resulting output will be of size 28*28.

The different parameters to define a single convolutional layer are:

i. Depth, or the number of different kernels and biases to be convolved with each output of the previous layer
ii. Height and width of each kernel
iii. Methods of boundary extension – typically zero padding, or padding by reflection.
iv. Overlap or shift between each of the individual output pixels, called stride

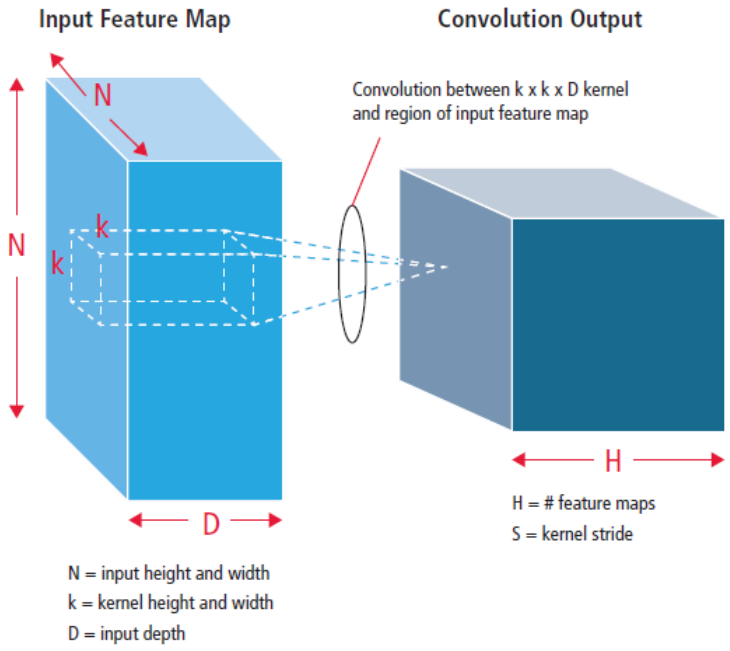


Figure 1.2 Convolutional Layer

Pooling Layer – A method of down sampling, wherein each part of the image is taken as a block, and replaced with a single value. This reduces the size of the spatial convolution layer. This pooling layer often succeeds the convolution layer. Here, a slice of the input (across width and height) is taken, and replaced by a single value. The generic form of pooling is Max Pooling, wherein the max value is replaced, and Average Pooling, wherein the average of that input slice is replaced. For example, a 2*2 kernel with an overlap of 2 between each output pixel, will get replaced by the max value of the corresponding data slice (in case of Max Pooling). Some parameters that define a pooling layer are

i. Size of the kernel
ii. Stride

The combination of Convolutional Layer followed by Pooling Layer considerably reduces the dimension – width as well as height of the input image.

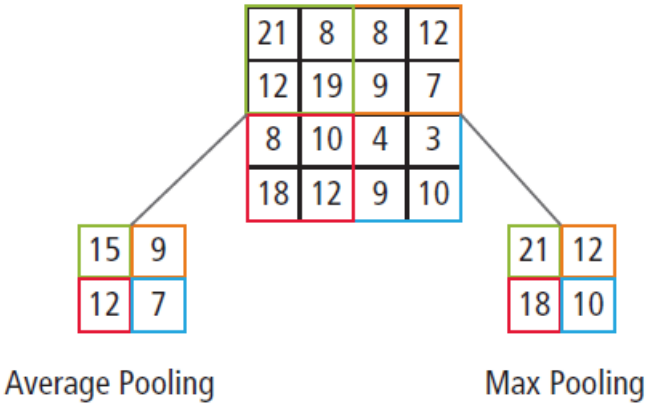


Figure 1.3 Pooling

Fully Connected Layer - A fully connected layer contains all the activations connected i.e. every neuron is connected to the output of the previous layer. This layer takes data (i.e. the layers preceding it) and outputs a vector of dimension N , where N is the number of output object classes. The output vector is a probability representation i.e. each value in the output vector represents the probability that the input belongs to that class, and these probabilities sum up to 1. A higher probability implies higher weights for that class. Their activation can be computed by matrix multiplication followed by a bias offset. This layer uses a SoftMax activation function in the output layer.

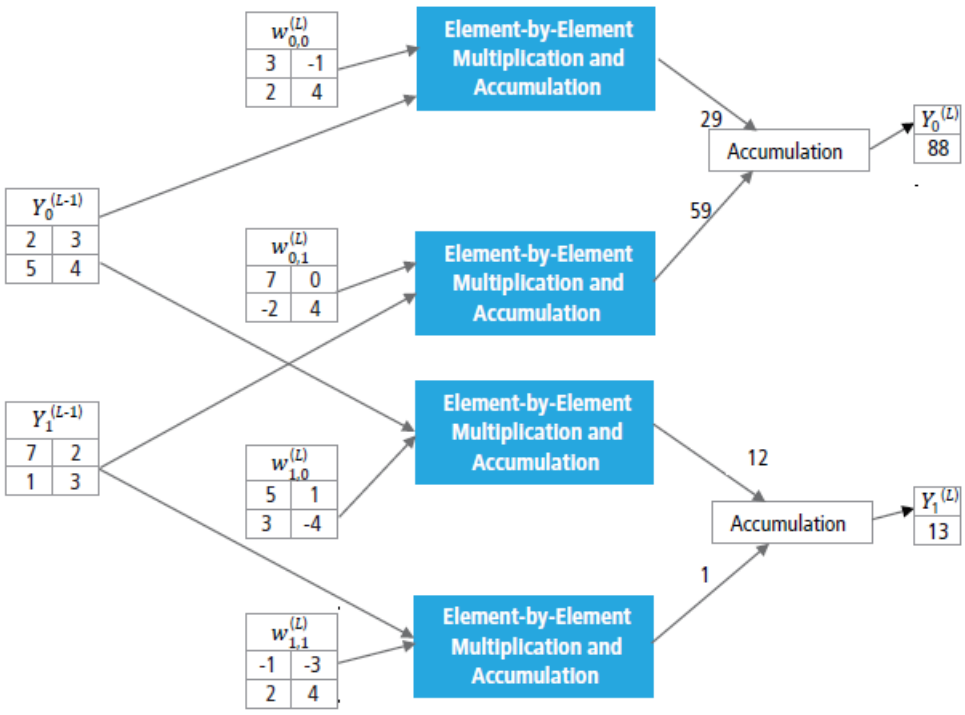


Figure 1.4 Fully Connected Layer

SoftMax Classification – A SoftMax classifier generalizes binary Logistic Regression classifier to many classes. The output of a Soft max classifier is normalized class probabilities. The function has two steps – first step is adding up the evidence of the input being in certain classes, and the second step is to convert the evidence into probabilities. The evidence is calculated in terms of weights and bias. This is mapping function takes an input data vector x and maps them to output class labels.

The mapping function can be given as

$$f(x_i, W) = W x_i$$

The SoftMax function can be expressed as [6]

$$p_i = \frac{\exp y_i}{\sum_{c=1}^{C} \exp y_c}$$

Where $i, c \in \{1,2, \ldots, C\}$ range over classes and $p_i, y_i \; and \; y_c$ refer to class probabilities.

Hence normalized probabilities are obtained at the output of the SoftMax classifier.

Activation function – An activation function is added at the output of any layer as a mapping function. They map the output of the neurons to any range. They could be either linear or non – linear. Linear activation functions do linear mapping of values, which are already linear. Hence linear activation functions are not popularly used. Non – linear functions introduce non – linearity into the network by some functions.

Some examples of the activation functions include Sigmoid, Tanh, ReLU and Leaky ReLU.

i. Sigmoid or Logistic Activation Function
- Function Range: [0,1]
- Useful for defining probability outputs
- Output – not centered at zero – causes gradient fluctuations and hence slow convergence
- Most commonly used Sigmoid: SoftMax

ii. Tanh or Hyperbolic Logistic Activation Function
- Function Range: [-1,1]
- Negative inputs → mapped negative and the zero inputs → mapped near zero.
- Mainly used for classification between two classes.
- Converges faster than sigmoid

iii. ReLU – Rectified Linear Unit Activation Function
- Range: [0, ∞]
- Converges faster than tanh and sigmoid
- Improper learning rate make the activation function die out
- All negative values →become zero – not able to fit or train the data properly.

iv. Leaky ReLU
- Improves ReLU by making the non – negative portion having a little slope value.
- Doesn’t Saturate
- Convergence faster than Sigmoid and Tanh
- Output is not zero centered – so doesn’t die out due to improper learning rate

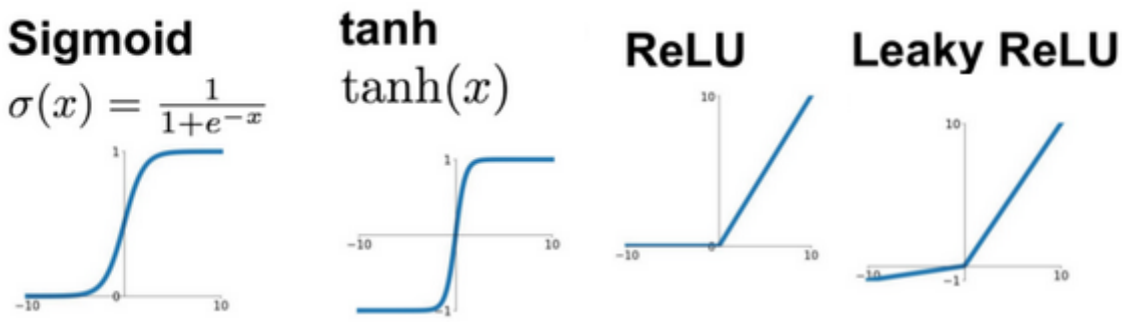


Figure 1.5 Activation Functions

Question 2: Overfitting Issue in Machine Learning

Overfitting corresponds to adapting the training data set to such an extent that the performance of the model to samples other than the training data set are degraded. That is, the model learns all the training data correctly, but it does not recognize the underlying process. In terms of neural networks, if the size of the training data set is small, only a few neurons will do most of the processing all times, making those neurons redundant, and the other neurons idle (comparatively). An example of overfitting is shown below [2].

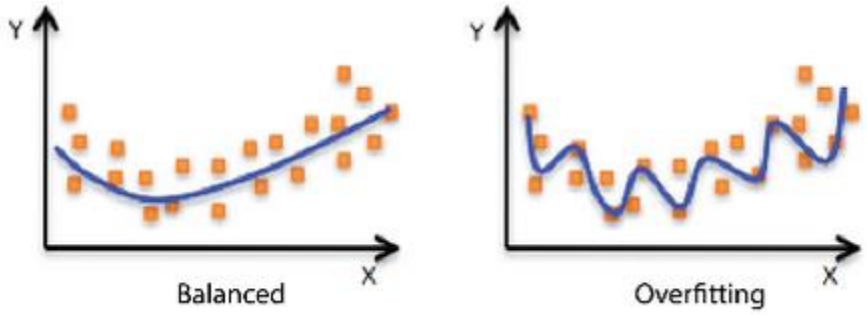


Figure 1.6 Overfitting

In simple terms, overfitting is where the model gives higher accuracy on the training data set only, and poor accuracy on any other data set.

Some techniques used in CNN to avoid overfitting are regularization. Here, instead of reducing the number of parameters, certain constraints are imposed during training. One commonly used method is Dropout – that helps to eliminate failure nodes. A dropout with a parameter c, in a single iteration, and drops the neurons from the network with a probability c3 throughout the

iteration. Hence the reliance on a set of neurons is eliminated. This incorporates relying on several neurons in a layer. A simple illustration of dropout is shown below [3].

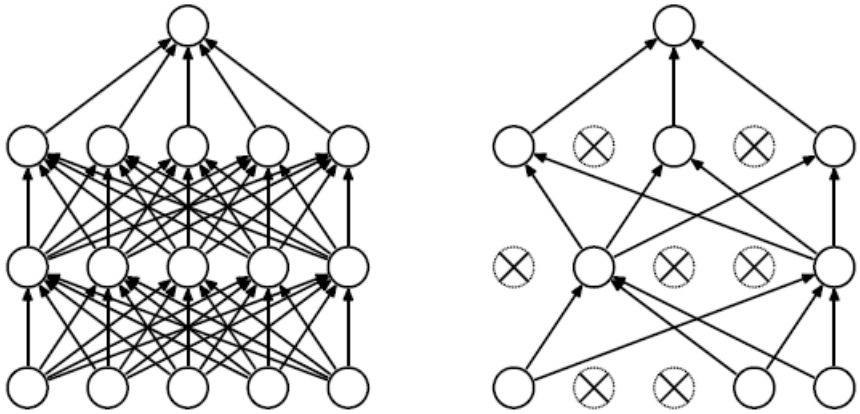


Figure 1.7 Dropout

Question 3 – CNN is much better than traditional methods for CV problems

CNN is widely employed in many of the Image Processing problems. Considering an example [4]

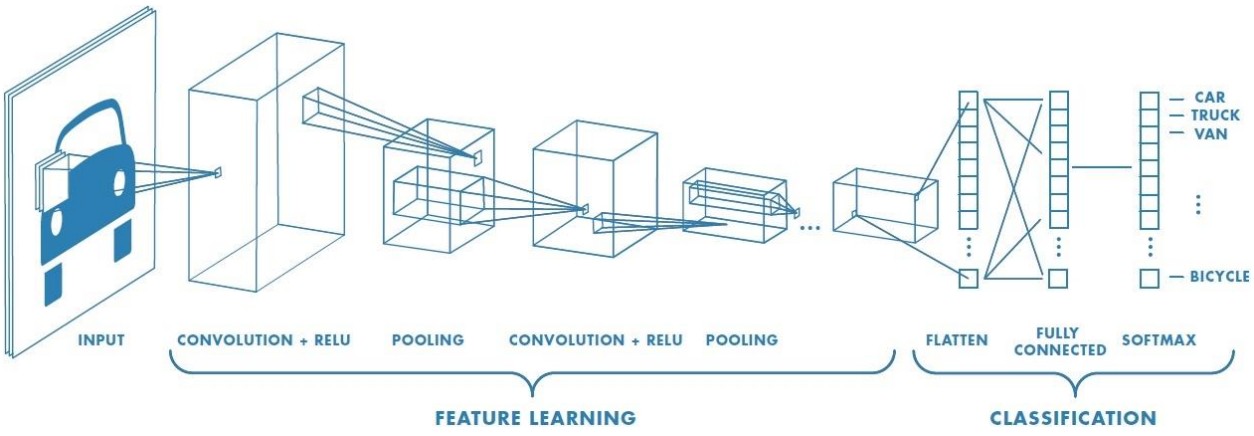


Figure 1.8 A Convolutional Neural Network

The objective is to get the input as an image and find which class the image belongs to. Some common methods of this image classification include the use of Artificial Neural Networks, Support Vector Machines, Random Forests, Decision Trees and Convolutional Neural Networks.

- Artificial Neural Networks are non – parametric classifiers used for classification of images. These classifiers are self – adaptive data driven, which classifies the images very effectively. The computation efficiency of this classifier is high. This has the disadvantage of overfitting the data, and the time for training is very high.
- Decision tree is also a non – parametric method, that does not require intensive training of data. This method has a deficient performance when there are more missing values, or when the different outcomes are correlated.
- Support Vector Machines are generalized non – linear classification techniques, which is widely used for two – class classification problems. The decision boundary is a hyperplane, that makes the decision rule simple, and hence the computational complexity is reduced.
- Convolutional Neural Network has a layered structure of neurons, wherein each layer performs a dedicated function. The different layers – convolutional layer followed by pooling and fully connected layers help in classification. The following are some advantages of CNN:

– Higher accuracy can be achieved. There are different architectures of CNN that help in achieving higher accuracy – like Lenet, AlexNet etc.
– CNN is also robust to distortions like change in shape due to lighting conditions, horizontal and vertical shifts etc.
– CNN requires lesser memory. The use of convolutional layer drastically reduces the memory requirement, where the operations are in – place.
– Considerable time for training. In a standard neural network, if the number of parameters increase, the time required for training will also increase proportionally. CNN uses a lesser number of features, and hence the time for training is also comparatively reduced.

This classifier suffers from the following disadvantages: CNN is a black box, wherein the actual process that happens is not explained. This method also requires huge amount of data. The time taken for training the data is also not very low. The parameter changes are based only on assumptions, and hence parameter changes for higher accuracies are based on previous works or trial – and error – methods.

Question 4 – Loss function and Back Propagation Architecture

Loss function:

A loss function optimizes the parameter values in a neural network model. This function maps a set of parameter values onto a scalar value that gives a relative measure of the performance of the network. Some examples of loss functions include Mean Squared Loss, Entropy Loss etc.

Data losses are usually average data losses. The common loss functions used are Mean Squared Error and Entropy Loss. Mean Squared Error is calculated as

$$MSE := \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

Where n is the number of classes and e is the difference between ground truth and inference result? This squared loss has the disadvantage that it might be dominated by the outliers.

Cross Entropy loss is based on maximum likelihood estimation. The formula of cross entropy is

$$H_{y'}(y) := -\sum_i y_i' \log(y_i)$$

Cross Entropy or log loss measures the performance of a classification model whose output is a probability in the range $[0,1]$. This increases as the predicted probability deviates from the original label value. Loss function continuously calculates the mismatch in the classification result during training, and tries to iteratively reduce it to a minimal value. Hence the loss will be high for a poor classification of data and low for a good classification of data.

Back Propagation:

Back propagation trains the neural networks, by the calculation of gradients that decide the weights of the connections. In back propagation, the gradient of the loss function is calculated, and this is used to calculate weights by gradient descent optimization. A known, definite value of output is needed for each input value. Backpropagation is used to find the direction of the steepest descent. An input vector is passed through the neural network, and the output is

compared with the desired output (or measured in terms of loss function), and an error value is calculated. These error values are then back – propagated, until each neuron in the network has its own error value contribution to the actual output, as in the figure below.

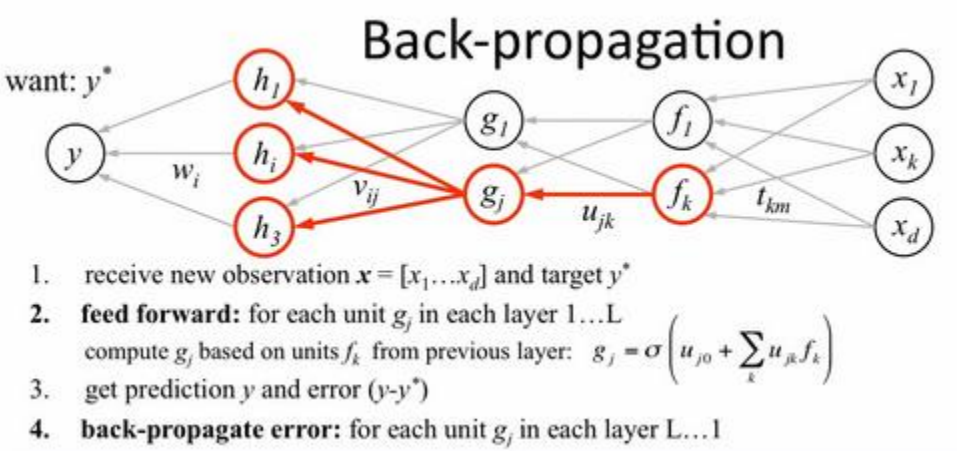


Figure 1.9 Back propagation – Simple illustration [5]

The following processes occur – The error values are used to calculate the gradient of the loss function. This gradient is fed into the gradient descent optimization. This updates the weights, and hence the loss function is minimized. This method of back propagation allows the neurons to learn all the characteristics of the input space. The goal of backpropagation is to compute the partial derivative of a loss function with respect to a weight.

Part B – Train LeNet – 5 on MNIST Dataset

LeNet 5 is one of the basic architectures of Convolutional Neural Networks. LeNet – 5 has 7 layers, excluding the output. The input is a 32*32 image, the receptive field is a 20*20 patch in the center.

The first layer C1 is the convolutional layer, having 6 feature maps, each of size 28*28. This layer has 156 trainable parameters.

Layer S2 is the sub – sampling layer, with 6 feature maps, each of size 14*14. The window size is chosen as 2*2. There are no weight calculations in this layer. This 2*2 block in C1 is connected to one unit in S2. These block co – efficients are added and multiplied using trainable

parameters, and the result is passed through the activation function. Hence S2 has half the size of C1. S2 has 12 trainable parameters. The activation function used here is Sigmoidal, which is non – linear. Sigmoid takes a longer time to converge.

Layer C3 is a convolutional layer, with inputs of size 14*14 and 16 feature maps, and each unit connected to several 5*5 blocks of S2's feature maps. This makes it non – symmetric and the number of connections are within limits. The output image is of size 10*10. C3 has 1516 trainable parameters.

Layer S4 does sub – sampling, having 16 feature maps, each of size 5*5. Like S2, each 2*2 unit in C3 is connected to one unit in S4. The dimensions of the specified hyper – parameters become 5*5 from the actual 10*10. This layer has 32 trainable parameters.

Layer C5 is again a convolutional, with 120 feature maps, each unit connected to several 5*5 blocks. Each feature map is 1*1. This layer is not fully – connected. This has 48120 trainable parameters. This layer has the largest receptive field i.e. each neuron has the largest information about the input image. Sigmoid activation introduces non – linearity.

Layer F6 is fully – connected to C5, contains 84 units, and has 10164 trainable parameters.

These six layers compute the inner product between the weight and input vector, and a bias is added. This is then passed through a sigmoid activation function. The final layer or the Output Layer is has Radial Basis Functions, one for each class. The output layer cascaded with a SoftMax layer, that provides the log probabilities of the output labels.

The other common parameters that are needed are as follows.

- Initialization of the Weight Kernel:
  The weights in each layer are initialized using some built – in initialization methods. Some examples are Random Normal, Random Uniform etc. The default initializer used is glorot_uniform, which draws samples within $[-limit, limit]$ where
  $limit = \sqrt{\frac{6}{fan-in+fan-out}}$, where $fan-in$ and $fan-out$ is the number of input and output units in the weight tensor respectively.

- Batch Size

  Batch size defines the number of samples taken per iteration through the network. This determines the time needed for training the given network. Batch size determines the convergence of the network. Ideally, batch size should divide the number of training samples without any remainders.

  In general, networks with smaller batch sizes converge faster since the weights are updated frequently. This also results in a slight error in the estimation of gradient. Larger batch sizes tend to make the weight vectors unstable, and hence longer time may be needed for convergence.

- Drop – out

  Drop – out is a form of regularization, that constraints the adaptation of the network during training, and hence prevents overfitting. Dropout drops certain neurons (according to some probability value), such that these neurons do not learn every detail of the previous instances in the training set. Thus, these neurons increase the efficiency of the hidden units to find patterns for unseen data. Dropout makes the weights spread over the input features instead of focusing on some features. For example, a dropout of 0.5 means 50% neurons are dropped.

- Data Shuffle

  Shuffling of the training data is important in a CNN. If the data is not shuffled, there are high chances of getting batches of highly correlated samples. Different random shuffling of the training set leads to different samples. The different neural network objective functions are non – convex, and hence using different ordering of training samples doesn't cause much difference in performance.

- Learning Rate

  Learning rate is a parameter that adjusts the weights of the network with respect to loss gradient. When the learning rate is low, smaller steps are taken along the downward slope, and hence more time to converge. When the learning rate is high, gradient descent goes beyond the maximum, and hence it may fail to converge. Learning rates are dependent on each application, and are chosen naively.

- Optimizer

Optimization Algorithms minimize the error function, which depends on the weights and bias of the model, and are useful in predicting the results from the inputs. An example is the Gradient Descent optimization algorithm – either stochastic or mini batch variants. A constraint on using the GD is the proper choice of the learning rate for every update, and the acceleration parameter called Momentum. There are different optimizer methods – like Adam, Adagrad, AdaDelta etc. Stochastic Gradient Descent takes longer time to converge. Adam stands for Adaptive Moment Estimation, wherein learning rates for each parameter are computed adaptively, and avoids high variance in updates. Adagrad optimizer modifies learning rate based on the values of all the past gradients, the only disadvantage being the learning rate is always decreasing. AdaDelta removes the decaying learning rate problem of Adagrad, by limiting the past gradient size to certain window. Of all the optimizers, Adam is found to give the best convergence for sparse data. SGD and other methods perform poorly.

- Number of epochs

  An epoch can be defined as the one complete cycle where the entire dataset is passed forward and backward through the neural network only once. Since Gradient Descent is an iterative process, multiple epochs are needed for updating weights. A single epoch leads to underfitting. As the number of epochs increase, there are more weight updates, and the model moves from underfitting to optimal fit. If the number of epochs is increased further, it leads to overfitting.

This LeNet – 5 architectures are implemented using Keras. This is tested using the MNIST dataset. This dataset consists of handwritten digits, from 0 to 9, and has 60000 training samples and test samples. The digits in this set are normalized by size and centered in an image. The code takes input from the Tensor flow version of MNIST – that has 55000 training samples, 5000 validation samples and 10000 test samples. Every MNIST sample has two parts – an image of the digit and a corresponding label.

Results :

This architecture is evaluated for different sets of parameters. The parameters that were modified are number of epochs. Dropout value, Kernel Initializer, Batch Size and Learning Rate.

The following table shows the comparison of different values for parameters that were modified, with the corresponding accuracies and losses.

Table 1 Different Parameter Settings for LeNet – 5 Architecture

S. No	Kernel Initializer	Learning Rate	Momentum	Batch Size	No. of Epochs	Dropout Value	Accuracy		Test Error	Test Loss
							Training	Testing		
1	Random Normal	0.1	0.5	128	200	0.25	99.88	99.191	0.809	0.024
2	**Default**	0.1	0.5	128	200	0.25	99.78	99.19	0.807	0.020
3	**Random uniform**	0.1	0.5	128	200	0.25	99.70	99.11	0.889	0.021
4	**Truncated Normal**	0.1	0.5	128	200	0.25	99.71	99.05	0.948	0.0264
5	Random_Normal	**0.2**	0.5	128	200	0.25	99.82	99.08	0.919	0.049
6	Random_Normal	**0.05**	0.5	128	200	0.25	99.69	99.04	0.959	0.037
7	Random_Normal	0.1	**0.8**	128	200	0.25	99.4	99.17	0.829	0.035
8	Random_Normal	0.1	**0.25**	128	200	0.25	99.89	99.10	0.899	0.061
9	Random Normal	0.1	0.5	**512**	200	0.25	99.82	97.34	2.65	0.08
10	Random Normal	0.2	0.5	**32**	200	0.25	99.73	99.19	0.81	0.026
11	Random Normal	0.2	0.5	128	**400**	0.25	99.67	99.17	0.829	0.028
12	Random Normal	0.2	0.5	128	**50**	0.25	99.73	98.38	1.62	0.051
13	Random Normal	0.2	0.5	128	200	**0.7**	99.71	99.14	0.859	0.042
14	Random Normal	0.2	0.5	128	200	**0.25**	99.83	99.17	0.829	0.028

Some accuracy and loss plots of the above values are shown in the following figures. Though dropout is not present in the actual architecture, it is added for optimization.

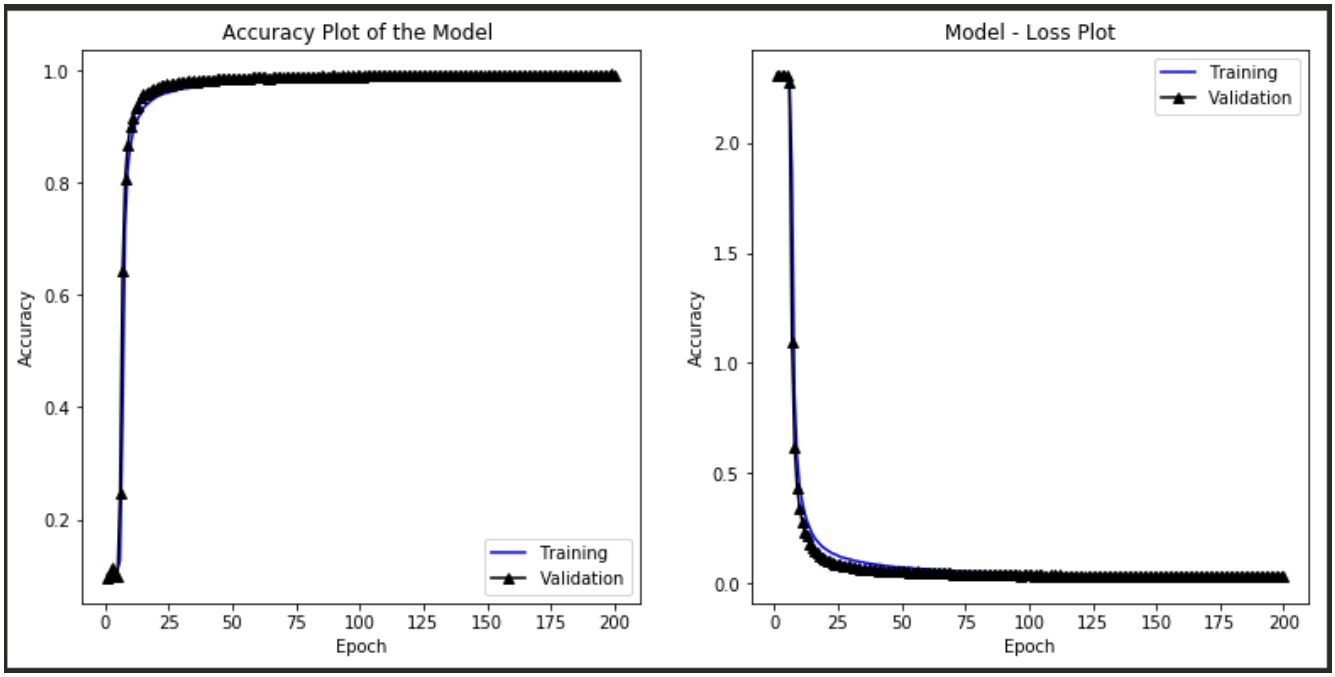


Figure 1.10 (a) Plot for Random Normal Kernel initializer, learning rate 0.1, momentum 0.5, batch size 128, 200 epochs and 0.25 dropout.

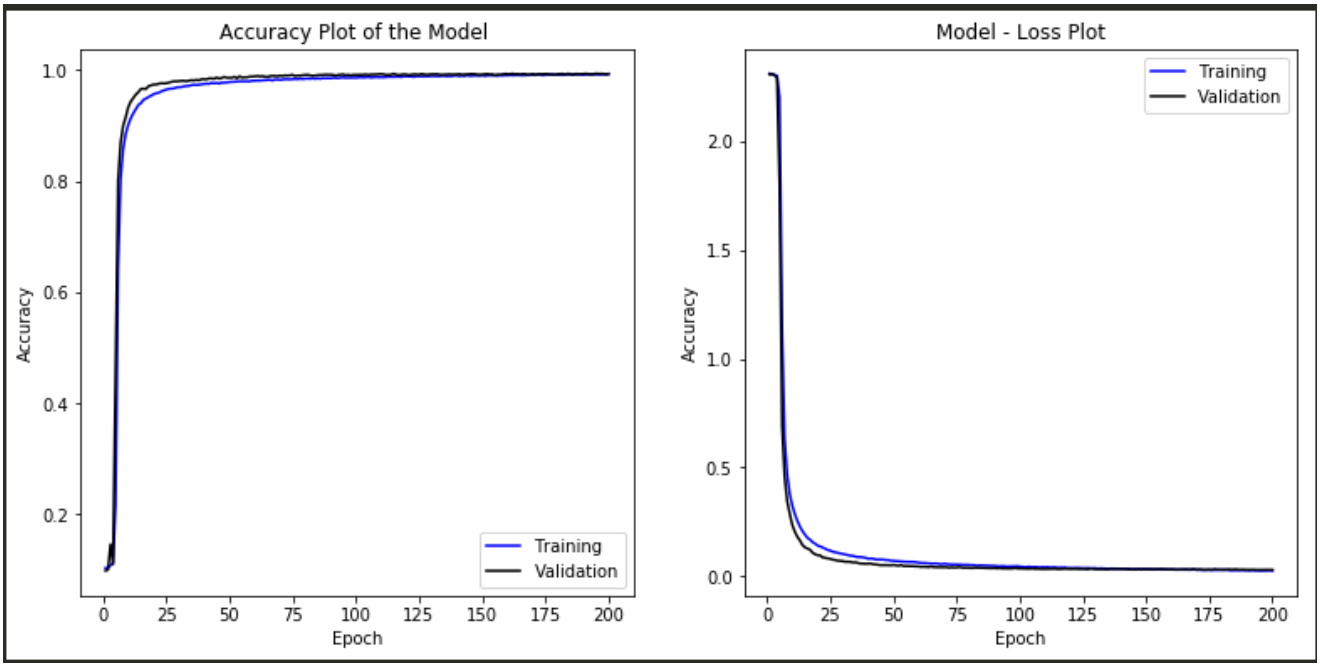


Figure 1.10(b) Plot for Variance Scaling Kernel initializer, learning rate 0.1, momentum 0.5, batch size 128, 200 epochs and 0.25 dropout.

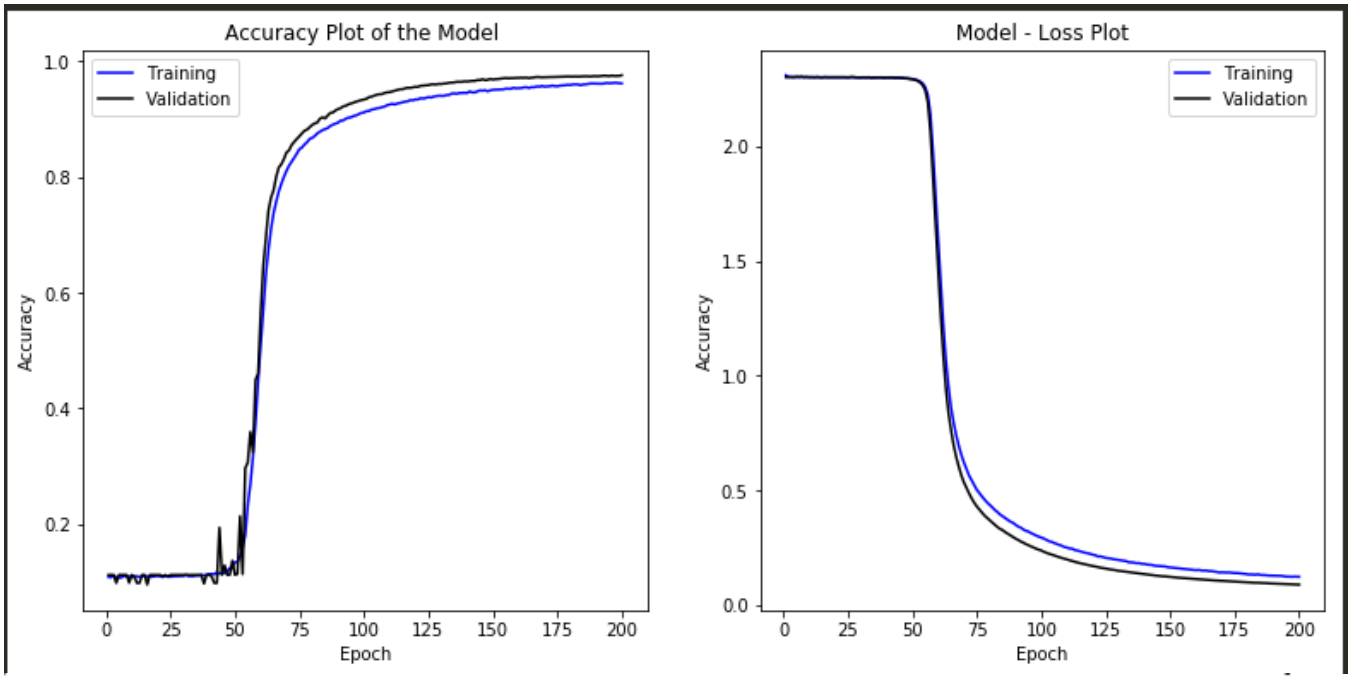


Figure 1.10(c) Plot for Random Normal Kernel initializer, learning rate 0.05, momentum 0.5, batch size 128, 200 epochs and 0.25 dropout.

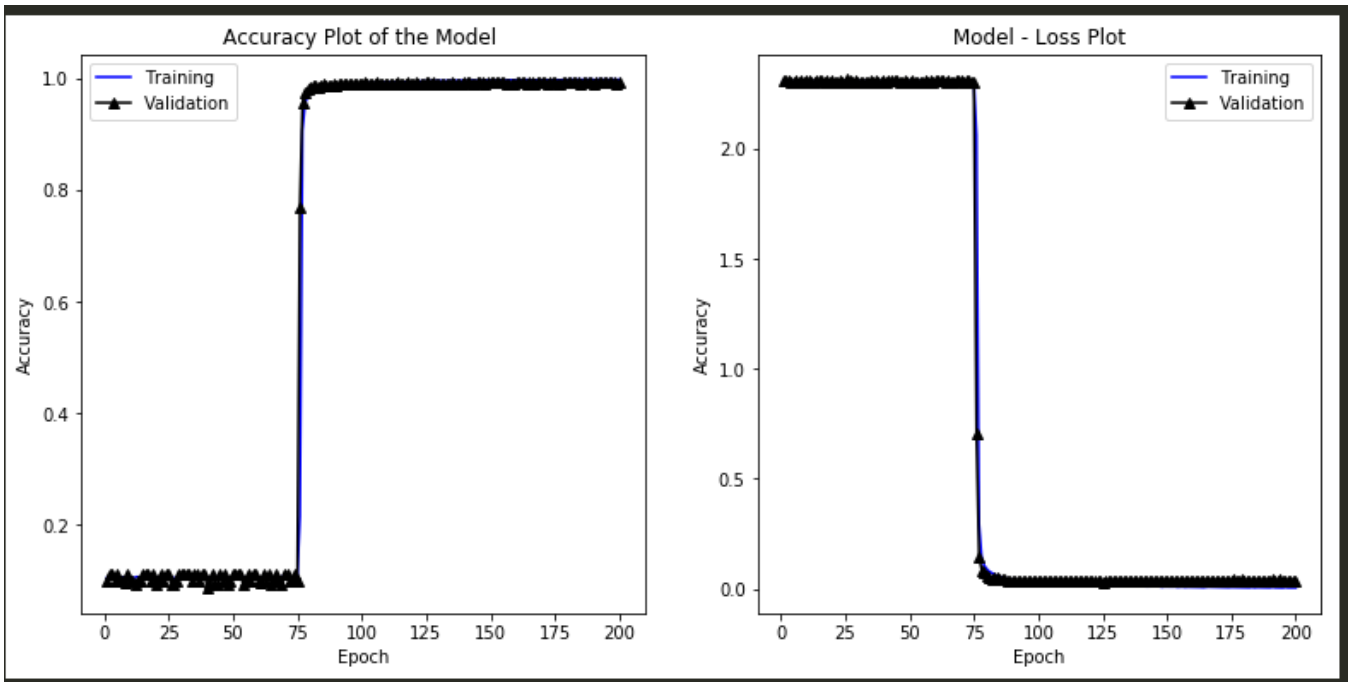


Figure 1.10(d) Plot for Random Normal Kernel initializer, learning rate 0.1, momentum 0.8, batch size 128, 200 epochs and 0.25 dropout.

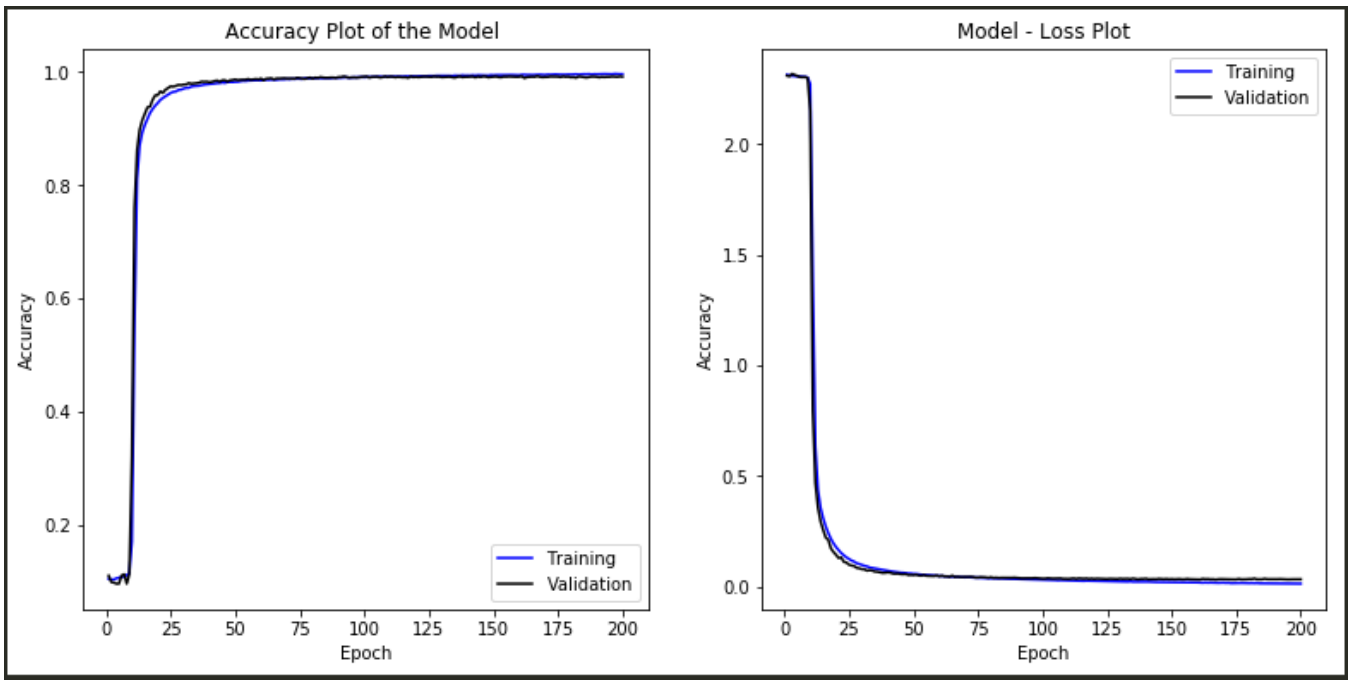


Figure 1.10(e) Plot for Random Normal Kernel initializer, learning rate 0.1, momentum 0.5, batch size 512, 200 epochs and 0.25 dropout.

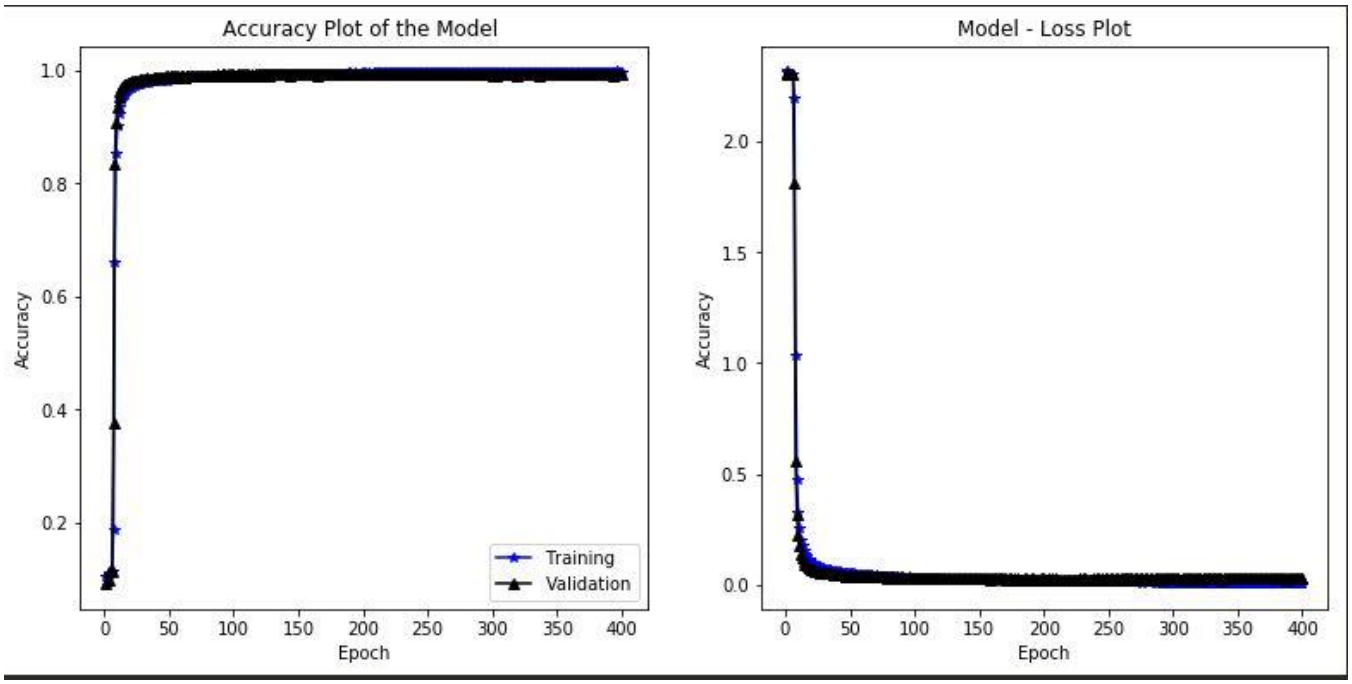


Figure 1.10(f) Plot for Random Normal Kernel initializer, learning rate 0.1, momentum 0.5, batch size 128, 400 epochs and 0.25 dropout.

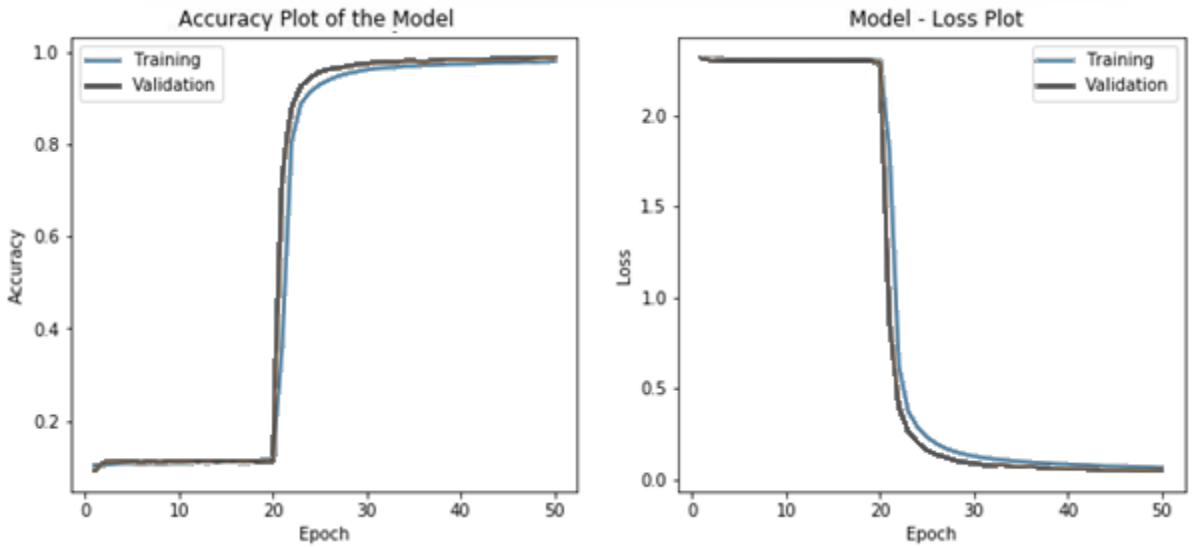


Figure 1.10(g) Plot for Random Normal Kernel initializer, learning rate 0.1, momentum 0.5, batch size 128, 50 epochs and 0.25 dropout.

The maximum accuracy obtained in this case is 99.191%. Hence the parameters were modified – and the architecture was modified. The parameters that were modified are :

- Activation Function: ReLU
- Filter in the first stage and second stage: 32 and 64 respectively
- Pooling is max pooling
- Dropout Values in two stages: 0.5 and 0.25
- Learning Rate = 0.1
- Momentum = 0.5
- Batch Size = 128
- Number of epochs = 200
- Optimizer : Adam Optimizer

The accuracy increased to 99.37% after applying these modifications. The results are plotted as in Figure 1.11.

Discussion:

Training LeNet – 5 on MNIST Dataset :

Different parameters were changed to find the best possible accuracy on the test set. Table 1 shows the different sets of parameters.

- There were no initial pre – processing steps. The images in the MNIST dataset are of size 28*28. Hence there was no need of any pre – processing.
- The architecture provided the information on the number of convolutional layers, pooling layers and fully connected layers. Hence a sequential model was built, and all layers are stacked onto the model.
- This architecture is applied to the MNIST dataset and its performance is evaluated by changing the different parameters via brute force methods.
- Loss and accuracy are plotted again the number of epochs, for both training and validation sets,. It is observed that the plots are complementary.
- The standard set that is implemented has Random Normal Kernel initializer, learning rate 0.1, momentum 0.5, batch size 128 and the model was run for 200 epochs. A dropout of 0.25 was added to optimize the model.
- The convolutional layer has the kernel initializer function, which is chosen either as Random Normal, Random Uniform or the default Glorot normal. Weights are initialized in each layer using these kernel initializers. The right choice of kernel initializer determines convergence rate. From Table 1(Observation Sets 1, 2, 3 and 4), it is observed that the best values of testing accuracy are obtained for Default kernel initializer and the Random Normal Kernel Initializer. The testing accuracies in these cases are close to 99.19%.
- The next parameter is the learning rate. Learning rate adjusts the weights of the network with respect to loss gradient. When the learning rate is low, the convergence time i.e. the time taken for the model to learn is higher. When the learning rate is high, it may diverge. Hence the learning rate should be optimum. The accuracy depends on the epochs as well as the learning rate, as in observation sets 5 and 6 in Table 1. This can be observed from Figure 1.10(a) and Figure 1.10(c). Figure 1.10(c), which uses a learning rate of 0.05, shows a slow convergence, compared to Figure 1.10(a), which has a learning rate of 0.1.

- An epoch can be defined as the one complete cycle where the entire dataset is passed forward and backward through the neural network only once. For a CNN, multiple epochs are needed for updating weights. As the number of epochs increase, there are more weight updates, and the model moves from underfitting to optimal fit. If the number of epochs is increased further, it leads to overfitting. The number of epochs, along with learning rate, contribute a major part to convergence. If the number of epochs is large, along with a large learning rate, converge is faster, and the model saturates. Figure 1.10(f) and Figure 1.10(g) show the effect on the number of epochs. Figure 1.10(f) shows the convergence for 400 iterations, wherein the accuracy seems to saturate after certain stage. Observation sets 11 and 12 in Table 1 also prove the same.

- Batch size defines the number of samples that propagate through the network. This determines the time needed for training the given network. Batch size determines the convergence of the network. Ideally, batch size should divide the number of training samples without any remainders. Figure 1.10(e) and Figure 1.10(a) can be used for comparing the batch size.

  From Table 1, observation sets 1, 9 and 10 compare the effect of batch size. It is observed that smaller batch size of 32 converges faster, since the weights are updated every 32 images, and the accuracy is also found to be higher. On the contrary, larger batch sizes (of say 512) tend to make the weight vectors unstable, and hence longer time may be needed for convergence, resulting in poor accuracy of the test set.
- Choice of the optimizer

  Basic CNN used Stochastic Gradient Descent, which takes a longer time to converge, and that happens only after large number of iterations. Hence an improvement could be made in the optimizer choice.
- Choice of dropout

  Dropout is used mainly to avoid overfitting issues. A dropout of p indicates neurons are dropped with a probability p before the next layer. The optimal choice of dropout is 0.5 – as in observation set 1. But different values of dropout does not matter, since it gives almost the same test accuracy.

- Choice of momentum Value

  Momentum defines how much percentage of the previous stage weights are used in the current stage. Different momentum values give different accuracies as in Observation Sets 7 and 8 of Table 1.

Using the Lenet 5 architecture, the best possible accuracy obtained was 99.191%. The choice of the parameters are Sigmoid activation function, Random Normal Kernel Initializer, Stochastic Gradient Descent Operation, Learning Rate of 0.1, Momentum of 0.5, Batch size of 128 and the number of epochs 200. The number of filters in each stage did not change. A dropout layer was added for optimization, with a value of 0.5. The corresponding accuracy – loss plots are given in Figure 1.10(a).

Part C – Improving LeNet – 5 for MNIST

Results :

Table 2 gives some of the trials for improving the LeNet 5 architecture.

Table 2 Improving LeNet 5

S. No	Parameter	Set 1	Set 2	Set 3	Set 4	Set 5
1	Activation Function	ReLU	ReLU	**ReLU**	ReLU	ReLU
2	Kernel Initializer	Random Normal	Random Normal	**Random Normal**	Random Normal	Random Normal
3	C1 : # Filters	6	6	**32**	32	120
4	S2 : Pooling Type	Max	Max	**Max**	Max	Max
5	Dropout	No	0.5	**0.5**	0.5	0.5
6	C3 : # Filters	16	16	**64**	120	120
7	S4 : Pooling Type	Max	Max	**Max**	Max	Max
8	Dropout	No	0.25	**0.25**	0.25	0.5

9	C5 : # Filters	120	120	**120**	120	120
10	F6 : # filters	84	84	**84**	84	84
11	Optimizer Choice	SGD	Adams	**Adams**	Adams	Adams
12	Learning Rate	0.1	0.1	**0.1**	0.1	0.1
13	Momentum	0.5	0.5	**0.5**	0.5	0.5
14	Batch Size	128	128	**128**	128	128
Training Accuracy		99.14	99.37	**99.52**	99.39	99.50
Test Accuracy		99.17	99.20	**99.37**	99.19	99.16
Test Error		0.8297	0.799	**0.629**	0.808	0.837
Training Loss		0.0535	0.0365	**0.0312**	0.0040	0.0521

The best set was Set 3, the accuracy vs epoch and the loss vs epoch graphs for both the training and validation sets are plotted.

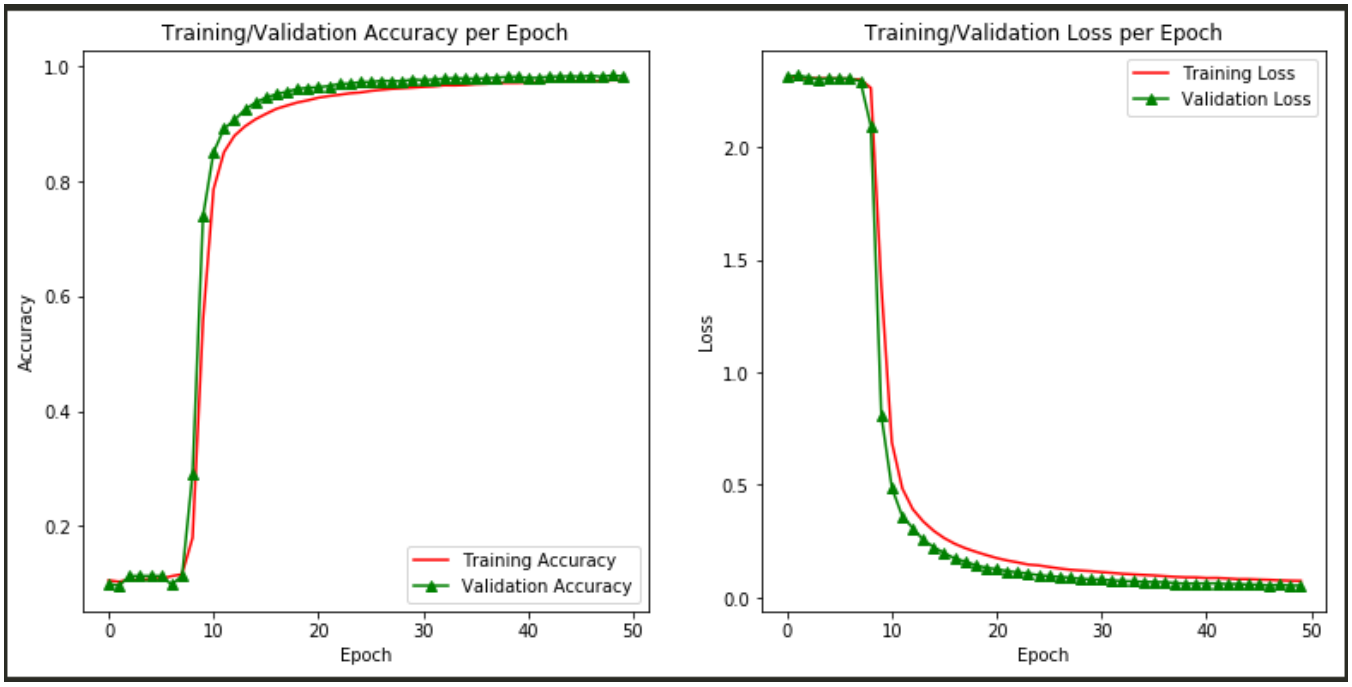


Figure 1.11 Plots for the improved Lenet – 5 Architecture

Discussion :

The training accuracy of the MNIST database is always high, as observed in Table 1. Only the test accuracy needs to be changed. Certain parameters are modified , and the results are obtained as in Table 2. The highest test accuracy obtained was 99.37%. Some of the parameters where the changes are made are discussed below.

- Choice of the Activation function :
  In the original LeNet5 architecture, Sigmoid is used as the activation function. The sigmoid function is not centered at zero, and hence it causes high fluctuations, so longer time to converge. Hence, to improve this, ReLU activation is used, which maps positive values onto linear scale, cancelling negative values. This leads to faster convergence.
- Number of filters :
  The number of filters or kernels in actual convolutional layers is 6 and 16. This is modified since larger window sizes bring in more information on the test data, but take longer time to converge.
- Kernel Initializer :
  The weights of each layer has to be chosen rightly. The choice of weights play a major role in convergence criteria, since they operate directly on the input data. For the actual data set , Random Normal was found to give the maximum accuracy, and hence the same is used.
- Optimizer Choice
  The original LeNet 5 used Stochastic Gradient Descent, this requires a large number of iterations to converge. Hence an optimum optimizer could be used, like the Adam Optimizer that calculates the momentum adaptively based on the previous layers. Adam is found to improve the accuracy.
- Pooling :
  The actual pooling mechanism in LeNet 5 is average pooling. Max Pooling is found to give better results, in many references. Hence max pooling is used.
- Learning Rate and Momentum :

Learning rate defines the pace of convergence of a model. The learning rate chosen is 0.1, and it is found to give better results. This could be modified. Momentum is chosen adaptively according to Adam Optimizer.

Hence the best parameters for achieving higher accuracy would be the Set 3 in Table 2.

Problem 2 – SAAK Transform and its Application to MNIST Dataset

Abstract and Motivation :

Convolutional Neural Networks(CNN) and Subspace Approximation with Augmented Kernels(SAAK) can be used for classification of images. Convolutional Neural networks make predictions by learning the relationship between data features and the responses recorded. CNN performs image classification by looking at a few low – level features (like curves and edges) and builds up more abstract layers through convolutions. SAAK also performs image classification using both spectral and spatial properties. Both these methods have similarities and differences.

Part A – Comparison between SAAK Transform and CNN Architecture

Some similarities between SAAK and CNN are

i. CNN and SAAK both employ convolution operation to generate outputs in a layer.
ii. The activation function used before each stage is the ReLU activation function.

Comparison between CNN and SAAK:

i. Architecture

CNN has a fixed architecture and an optimization function. In a CNN, weights are obtained by back propagation through iterative processes. CNN follows a complete end – to – end architecture. The inputs are properly passed through all layers – convolutional layer, pooling and fully connected layers. The weights are defined iteratively through back

propagation procedure. CNN based methods have weaknesses in terms of efficiency, scalability and robustness. Whenever the number of output object classes must be modified, or the number of features must be changed, the weights of the entire network must be changed i.e. the entire model must be retrained. The filter weights are not robust enough to handle changes in the number of object classes. Thus, CNN performs better for fixed number of object classes and fixed number of features.

On the other hand, SAAK has two modules – feature extraction and classification module. For extraction of features, data labels are not needed. SAAK is robust to changes in the number of object classes. SAAK uses dimensionality reduction, and hence only the most dominant features will be selected. If the covariance matrices do not change much, the SAAK coefficients should also not change much.

ii. Inverse Transform

CNN doesn't seem to have any inverse transform except using the RECOS Transform. Images can be obtained by two approaches – Generative Adversarial Network and Variational Auto Encoder. In both these approaches, the entire training of the dataset is required. Even though training is done on an intensive scale, performance of the network is completely random.

On the other hand, SAAK has orthonormal transformation kernels, on which inverse transform can be done easily. Forward SAAK Transform is done on the source image to obtain the SAAK co – efficients, from which SAAK co – efficients of the target image are obtained. Performing inverse SAAK on it gives the target image.

iii. Basic blocks

CNN is a neural network, which consists of pattern recognition techniques. It is just known that CNN contains convolutional layers, pooling layers and fully connected layers. CNN could be approximated as a Multi-Layer Perceptron, as in some research papers. Except that, the work of the CNN is a black – box, which is difficult to explain even with mathematical concepts.

In comparison, SAAK is fully based on orthonormal kernels, augmentation and spatial – spectral transformations, which are based on statistics and linear algebra.

iv. Determination of Filter Weights

In CNN, filter weights are determined by optimization functions that iteratively run on data and its corresponding labels. The weights are updated by backpropagation mechanism. The number of iterations are very large.

SAAK, on the other hand, determines filter weights by application of KLT in every stage. This does not require the use of any data labels, except in the final decision function.

v. Feature Extraction

In CNN, various features are learned through back propagation automatically from the data and its corresponding labels. These are done for many iterations.

In SAAK transform, neither the data labels nor the back-propagation mechanism is needed for feature extraction. SAAK Coefficients are extracted, and multiple SAAK coefficients having higher discriminant power from the same class are used to build features.

vi. Implementation Mechanism

CNN has convolutional layers, that perform convolution using multiple filters. These convolution operations are performed on overlapping blocks. Since overlapping blocks are used, dimension reduction is necessary. Hence spatial pooling is required. The common types of pooling used are average pooling and max pooling. Moreover, the filter size in CNN is generally odd like $3 * 3$, $5 * 5$ etc.

On the contrary, SAAK does not need extra stages of spatial dimension reduction. The KLT does spatial dimension reduction. Moreover, the dimension of SAAK in each stage could be even or odd.

Part B – Application of SAAK Transform to MNIST Dataset

Abstract and Motivation:

Both CNN and SAAK can be used for image classification. CNN has its own disadvantages like robustness, efficiency and scalability. CNN requires computationally intensive training. Even for slight changes in the number of classes, size of the training data set or the number of features, the entire model must be retrained. These CNN models are completely dependent on the end –

to – end optimization, which makes them less robust. SAAK Transform overcomes all these disadvantages.

Approach:

SAAK Transform is applied to the MNIST Dataset. SAAK Transform can be considered as a mapping from a real function on a three – dimensional cuboid to a one – dimensional spectral vector. It does spatial – spectral decomposition of the image. Figure 2.1 shows the SAAK Transform for multiple stages.

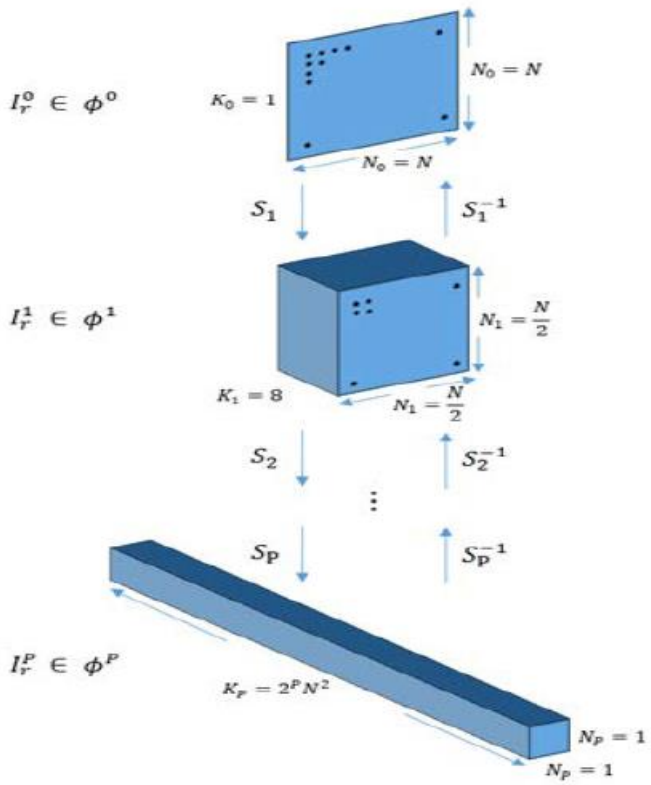


Figure 2.1 SAAK Transform for multiple stages

The forward SAAK Transform consists of three steps:

- Using KLT, an approximation of subspaces that has orthonormal bases, and is linear
- Kernel Augmentation with the negative of the kernel
- Application of ReLU activation function to the output

The overall process can be divided into three tasks – SAAK Coefficient Generation, Coefficient Selection and Dimensionality Reduction and Classification.

Step 1: SAAK Coefficient Generation

- SAAK Coefficients work on non – overlapping regions. Hence for each input image, a non-overlapping patch of spatial size 2x2 is selected.
- Variance of each patch is calculated and the small-variance patches are removed.
- Principle Component Analysis (PCA) is done on to the zero-mean patch data to get the PCA transform matrix.
- Transform all the input data patches by selecting the important spectral components in the PCA matrix.
- Augment transform kernels using sign – to – position format conversion.

This process is repeated for each SAAK transform stage. The number of important components in each stage are chosen as 3, 4, 7, 6, 8.

Step 2: Selection of SAAK Co – efficients

- After getting responses from all SAAK stages, a 1500-dimension feature vector is obtained.
- F-test is performed on it and features with larger F-test scores (around 1000 dimension) are chosen.
- Then, another round of PCA is done to reduce the feature dimension to 32, 64 and 128.

Step 3: Collection of features

- Class labels are utilized to collect features of training samples.

So in every stage of the transform, the process is like :

Load the input images (According to training set and validation set numbers) → Perform KLT by quad – tree process → Standardization → Apply PCA to extract the significant kernels → Sign to Position Conversion or simply Augmentation → SAAK Co – efficients.

There are five stages of this transform, that perform SAAK have different kernel sizes, and hence it reduces dimensions both in spatial and spectral ways.

The SAAK Kernel information can be given as follows

Stage	Kernel Size	Feature Multiplier	No.of important components	Number of features
1	16*16	256	3	768
2	8*8	64	4	256
3	4*4	16	7	112
4	2*2	4	6	24
5	1*1	1	8	8
Total Number of components				1168

This must be reduced to 1000 dimensions. This is further reduced to 32, 64 and 128 features using Principal Component Analysis. Then this model is tested for accuracy using Support Vector Machine and Random Forest Classifiers.

Results

The results of each stage are as follows – here a sample output is given for 10000 sample training set size.

```
Stage 1 start:
Extracted image batch shape: (256, 10000, 2, 2, 1)
Processed image batch shape: (2560000, 4)
Number of anchors to keep: 3
Anchor vector shape: (3, 4)
Augmented anchor vector shape: (6, 4)
Reshaped anchor shape: (6, 2, 2, 1)
Tensorflow formated anchor shape: (2, 2, 1, 6)
Saak coefficients shape: (10000, 16, 16, 6)
Stage 1 end

Stage 2 start:
Extracted image batch shape: (64, 10000, 2, 2, 6)
Processed image batch shape: (640000, 24)
Number of anchors to keep: 4
Anchor vector shape: (4, 24)
Augmented anchor vector shape: (8, 24)
Reshaped anchor shape: (8, 2, 2, 6)
Tensorflow formated anchor shape: (2, 2, 6, 8)
Saak coefficients shape: (10000, 8, 8, 8)
Stage 2 end
```

```
Stage 3 start:
Extracted image batch shape: (16, 10000, 2, 2, 8)
Processed image batch shape: (160000, 32)
Number of anchors to keep: 7
Anchor vector shape: (7, 32)
Augmented anchor vector shape: (14, 32)
Reshaped anchor shape: (14, 2, 2, 8)
Tensorflow formated anchor shape: (2, 2, 8, 14)
Saak coefficients shape: (10000, 4, 4, 14)
Stage 3 end

Stage 4 start:
Extracted image batch shape: (4, 10000, 2, 2, 14)
Processed image batch shape: (40000, 56)
Number of anchors to keep: 6
Anchor vector shape: (6, 56)
Augmented anchor vector shape: (12, 56)
Reshaped anchor shape: (12, 2, 2, 14)
Tensorflow formated anchor shape: (2, 2, 14, 12)
Saak coefficients shape: (10000, 2, 2, 12)
Stage 4 end
```

```
Stage 5 start:
Extracted image batch shape: (1, 10000, 2, 2, 12)
Processed image batch shape: (10000, 48)
Number of anchors to keep: 8
Anchor vector shape: (8, 48)
Augmented anchor vector shape: (16, 48)
Reshaped anchor shape: (16, 2, 2, 12)
Tensorflow formated anchor shape: (2, 2, 12, 16)
Saak coefficients shape: (10000, 1, 1, 16)
Stage 5 end
```

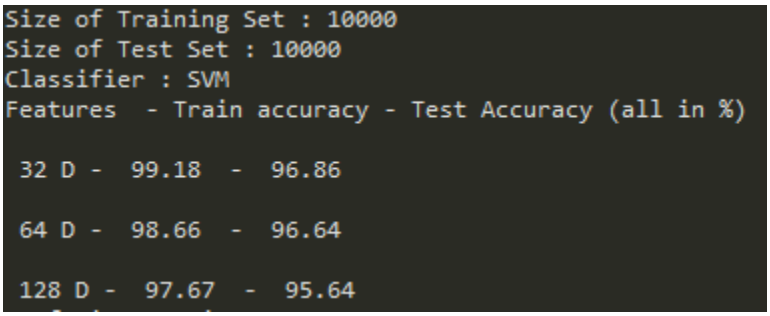
The test accuracy obtained for different trials are summarized in Table 3.

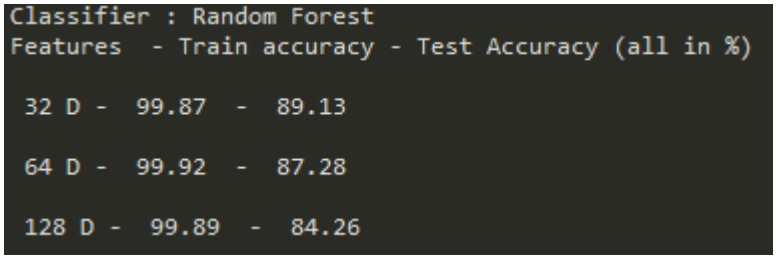
Table 3 Results obtained for different values

S.No	Training Set Size	Number of dimensions using PCA	Classifier	Training Accuracy	Test Accuracy
1	40000	32	SVM	99.20	**98.34**
			RF	99.71	93.4
		64	SVM	98.45	97.27
			RF	99.79	92.32
		128	SVM	97.68	97.54

			RF	99.81	91.29
2	50000	32	SVM	99.22	98.24
			RF	99.88	94.21
		64	SVM	99.80	93.41
			RF	99.87	92.37
		128	SVM	99.79	91.65
			RF	99.85	91.34
3	59000	32	SVM	99.07	98.27
			RF	99.78	94.07
		64	SVM	98.54	98.20
			RF	99.84	92.04
		128	SVM	98.21	97.85
			RF	99.88	90.74
4	10000	32	SVM	99.18	96.86
			RF	99.87	89.13
		64	SVM	98.66	96.64
			RF	99.92	87.28
		128	SVM	97.67	95.64
			RF	99.89	84.26

The screenshot of the output obtained – for training set 10000 is shown below.





Discussion:

From Table 3, it is observed that

- the number of features play a major role in determining the training and test accuracies. Training accuracy is always high – since almost the entire dataset is used.
- PCA does dimensionality reduction i.e the number of features are reduced from a massive 1000 to just 32, 64 or 128 features. It is observed that the accuracy is still more for 32 features (in most cases).
- There are two types of classifiers used – SVM and RF. In both the cases, training accuracy is obviously higher.
- Though the training accuracies of RF classifier are found to be higher than that of the SVM classifier, the test accuracies for RF classifier are low. This might lead to overfitting.
- RF Classifier doesn't work when there are a large number of values, and it fails. SVM can accommodate large number of values.
- PCA overall reduces the computation time, by reducing the number of dimensions. When the size of the training set is also reduced, the time for the entire program to run also decreases.
- PCA is found to give poor results as the number of features increases.
- When the size of the training set is small, the accuracy is also found to be lower, as observed from Table 3.

The best set of training accuracy obtained are close to 99.8% for all three sets of feature reduction methods, and the highest test accuracy is close to 98.3% for the three cases, which are highlighted in Table 3. Comparing with CNN, the accuracy for CNN is much higher, even though the size of the training set differs. CNN needs a large training set for higher accuracy. CNN is also found to take a very long time for computation, while SAAK takes comparatively lesser time.

Part C – Error Analysis

Abstract and Motivation :

Both SAAK Transform and CNN can be used for image classification. Though both these methods achieve classification, the accuracy factor differs for both. CNN has its own disadvantages of accommodating changes in classes, features or the size of the trainings set. SAAK overcomes all these by its unique transformation kernels. But this suffers from certain disadvantages.

Approach:

The error analysis of CNN and SAAK is performed. An error occurs when the input data is incorrectly classified. Classification errors could be due to

- Missing Data
- Imbalance in the datasets
- Wrong choice of classifiers
- Wrong choice of classification mechanism etc.
- One measure to compare the performance of both is by the use of a Confusion Matrix. A confusion matrix is written as

		prediction outcome		
		p	***n***	**total**
actual value	***p'***	True Positive	False Negative	P'
	n'	False Positive	True Negative	N'
	total	P	N	

True Positive and True Negative give the values of correctly classified samples, and False positive and false negative denote misclassified samples. Hence for any model, for higher classification accuracy, the model should have higher values along the main diagonal.

SAAK is compared in terms of this confusion matrix.

Results:

The confusion matrix for CNN is given below.

```
[[ 976    0    0    0    0    0    1    2    1    1]
 [   0 1027    0    0    0    1    1    0    0    0]
 [   0    0 1133    1    1    0    0    1    0    0]
 [   0    0    0 1007    0    3    0    0    0    0]
 [   0    0    1    0  958    0    2    1    0    1]
 [   1    0    0    6    0  884    1    0    0    0]
 [   6    2    0    0    1    7  939    0    2    0]
 [   0    3    3    0    0    0    0 1019    2    1]
 [   3    0    2    4    2    3    0    1  978    2]
 [   0    2    0    2    8    5    0    3    2  987]]
```

The confusion Matrix for SAAK is given below.

```
Confusion Matrix:
 [[ 973    0    4    0    0    2    6    0    2    4]
 [   0 1126    1    0    0    0    2    9    0    4]
 [   1    3 1007    3    5    0    0   13    2    1]
 [   0    1    3  988    0    9    0    2    5    6]
 [   0    0    1    0  957    1    2    3    5   14]
 [   2    1    0    4    0  866    6    0    3    2]
 [   1    2    1    0    3    5  939    0    3    1]
 [   1    0    9    7    1    1    0  988    3    7]
 [   2    2    6    6    1    6    3    0  949    5]
 [   0    0    0    2   15    2    0   13    2  965]]
```

The confusion matrix is a 10*10 structure, that has actual values on one side, and the predicted values on the other side. It is observed that all the principal diagonal elements have higher values.

Discussion

From the confusion matrix of SAAK and CNN, it is observed that CNN has slightly higher values on the main diagonal – which means that there are not too many off – diagonal elements in the confusion matrix as SAAK. Moreover comparing the efficiency from the Tables 1, 2, and 3 , CNN is found to perform better. But CNN fails when the metric is compared based on the size of training set.

SAAK performs better even when a small training set is used, and gives higher accuracy comparatively. The error is also found to be low, and hence both SAAK and CNN can be used for all image classification applications, and based on the desired requirement, the model can be chosen accordingly.

To improve the performance of CNN, the following could be done :

1. ***Improvising the data set*** – The dataset given as input to CNN is the most important factor. The data should be free of any unknowns, and even if there are unknowns, there shoukd be proper methods of imputing the data for correct classification.
2. ***Architecture of CNN*** - Since CNN is a black box, the inner architecture of CNN also matters. Though there are different architectures that give error rates as low as 1%, the time for execution of the code should also be considered.
3. Changes in the arrangement of layers – The arrangement of layers could be modified. The number of each layers could be improved.

To improve the performance of SAAK, the following could be done :

1. SAAK operates good even if there is a lesser training data set. Hence the quality of the input data set needs to be high. There should be no missing data, the dataset should be balanced.
2. SAAK co – efficients are extracted using F score as a feature reduction measure. There are simple methods of dimensionality reduction used, such as PCA. If these methods are improved – like using covariance or correlation metrics, the performance could be improved.

Thus CNN and SAAK can be improved.

References:

1. Discussion Notes and Lectures

2.AWS Document: model overfitting vs Underfitting

https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

3. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research 2014, pp1929 – 1958.

4. https://www.mathworks.com/discovery/convolutional-neural-network.html

5. https://www.youtube.com/watch?v=An5z8IR8asY

6. http://knet.readthedocs.io/en/latest/softmax.html

7. SAAK Transform https://arxiv.org/abs/1710.04176