# EE569 Digital Image Processing

Spring 2018

Assignment 4

Name: Thiyagarajan Ramanathan

USC ID: 4973341255

Issue Date: 3/25/2018 Due Date: 4/29/2018

## Problem 1: CNN Training and Its Application to the MNIST Dataset (50 %)

You will learn to train one simple convolutional neural network (CNN) derived from the LeNet-5 introduced by LeCun et al. [1]. Furthermore, you need to apply it to the MNIST dataset [2]. The MNIST dataset of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. Figure 1 shows some exemplary images from the MNIST dataset.

**Figure 1: MNIST dataset**

The CNN architecture for this assignment is given in Figure 2. This network has two *conv* layers, and three *fc* layers. Each *conv* layer is followed by a *max pooling* layer. Both *conv* layers accept an input

receptive field of spatial size *5x5*. The filter numbers of the first and the second *conv* layers are 6 and 16 respectively. The stride parameter is 1 and no padding is used. The two *max pooling* layers take an input window size of *2x2*, reduce the window size to *1x1* by choosing the maximum value of the four responses. The first two *fc* layers have 120 and 80 filters, respectively. The last fc layer, the output layer, has size of 10 to match the number of object classes in the MNIST dataset. Use the popular ReLU activation function [3] for all *conv* and all *fc* layers except for the output layer, which uses softmax [4] to compute the probabilities.

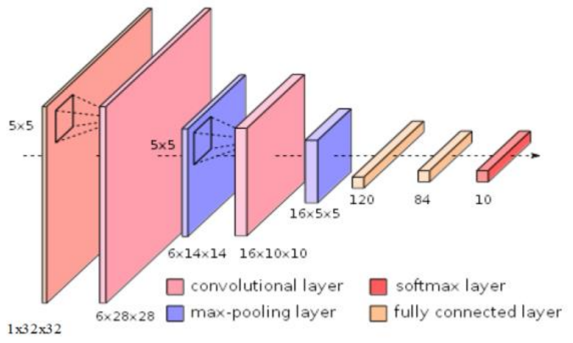


**Figure 2: A CNN architecture derived from LeNet-5.**

## (a) CNN Architecture and Training (20%)

Explain the architecture and operational mechanism of convolutional neural networks by performing the following tasks.

- Describe CNN components in your own words: 1) the fully connected layer, 2) the convolutional layer, 3) the max pooling layer, 4) the activation function, and 5) the softmax function. What are the functions of these components?
- What is the over-fitting issue in model learning? Explain any technique that has been used in CNN training to avoid the over-fitting.
- Why CNNs work much better than other traditional methods in many computer vision problems? You can use the image classification problem as an example to elaborate your points.
- Explain the loss function and the classical backpropagation (BP) optimization procedure to train such a convolutional neural network.

Show your understanding as much as possible in words in your report.

## (b) Train LeNet-5 on MNIST Dataset (15%)

Train the CNN given in Fig. 2 using the 60,000 training images from the MNIST dataset.

- Explain how you initialize the network parameters (filter weights, learning rate, decay and etc.) and discuss the effect of different settings.
- Compute the accuracy performance curves using the epoch-accuracy (or iteration-accuracy) plot on training and test datasets separately. Plot the performance curves under 5 different yet representative parameter settings. Discuss your observations.
- Find the best parameter setting to achieve the highest accuracy on the test set. Then, plot the performance curves for the test set and the training set under this setting.

## (c) Improve the LeNet-5 for MINST dataset (15%)

This is an open part for you to understand the CNN architecture. Please feel free in modifying the baseline CNN in Part (b) to improve the classification accuracy. For example, you can increase the depth of the network by adding more layers, or/and change the number of filters in some layers. You can also try different activation functions or optimization algorithms. Going deeper, you can even refer to any modern techniques in the literature. Report the best accuracy that you can achieve and describe your network architecture and the training parameter setting to reach this result. Discuss the sources of performance improvement. Your grading in this part will be based on your obtained performance in comparison with other students in the same class.

# Problem 1:

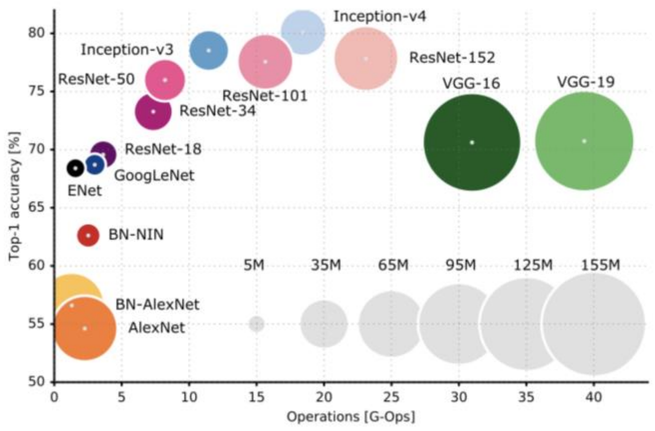
# CNN training and its application to the MNIST dataset

## Abstract and Motivation:

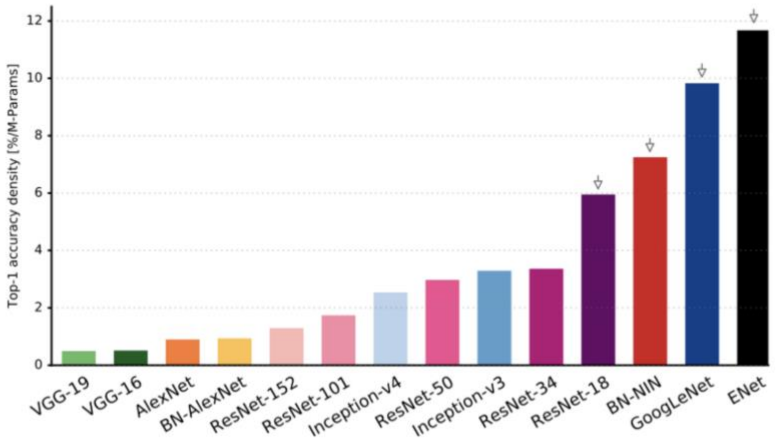
Convolutional Neural Networks have revolutionized the Image recognition tasks in the field of Computer Vision and Machine Learning. Neural Network classifiers have existed for a long time, but their popularity reduced because of lack of computational power. Now, due to the discovery of powerful processors and good computational power, they have risen again and are proving to be one of the best classifiers. Convolutional Neural Networks are specific to Image Processing and they have the same structure as the traditional neural networks with the difference being in how the features are extracted. For a typical classification task, the features are chosen based on the mean, standard deviation, a polynomial relation and so on. But for Convolutional Neural networks, the features are extracted using a filter. This is the where the part of convolution comes in. Convolution is specific to signal processing application, it is the inner product between two matrices, one being the filter and another being the block in the image itself. To extract different features from the image, different filters are designed and then fed to the traditional neural network classifier for classification and segmentation purposes.

In any classifier, the tasks can be divided into two stages namely, training and testing. For an artificial neural network, the training task consists of the learning part in which the weights for each feature are decided. In Convolutional Neural network, the learning part consists of the stage in which the filters that are applied to the image are tuned to extract the right features. This is performed by the back-propagation algorithm. The main part in any convolutional network is the learning phase. This is the most computational part. The CNN network named Alex-Net takes almost 2 weeks for the training stage, even when using two GPU's.

Le-Net 5 is one of the very early stage CNNs. Though it was designed in 1998, the algorithm did not gain popularity due to the lack of computational power. After 2010 due to the increase in computational power available, and because of the ImageNet database, the CNNs have evolved again. Though there is no mathematical background to these CNNs, they proved very good results as they are trained in a way very similar to how the human brain is trained for a child. The accuracies of the CNNs with the number of parameters used for each of them are given below in the following figure. All the accuracies are for the ImageNet database.



As we can see, Inception-v4 has got an accuracy of 80% while it has almost 35M parameters. Though VGG-19 has got an accuracy of around 70%, the training parameters are almost equal to 155M parameters which is a very large number with current computational powers. Another idea we can understand is that, the number of parameters does not decide the classification accuracy. Even the architecture of the network plays a major role in the deciding the accuracy as well as the training time. Thus, Inception-v4 is a very good architecture when compared to the other architectures. Inception-v4 has both very high accuracy and less number of training parameters conveying that it is very efficient when compared to the other CNN architectures. The following figure demonstrates the ratio of accuracy and number of parameters trained.

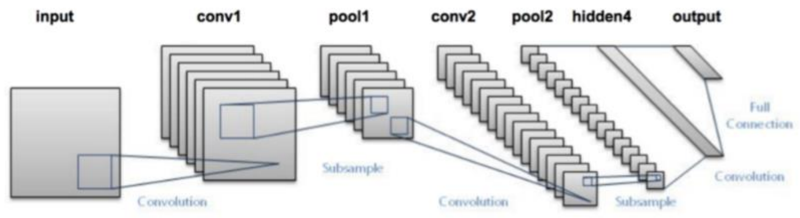


In the above figure, we can see that E-Net has a got a very good accuracy density. But the accuracy is just around 60% on the ImageNet database. Thus, accuracy and the training parameters are a trade-off. Hence, we can conclude that the accuracy of any Convolutional Neural Network depends mainly on the number of parameters and the architecture of the Neural Network.

The objective of the first problem, is to design the Le-Net 5 architecture for the classification of MNIST dataset which is a dataset of Hand-written digits. The first part of the problem asks us to understand the structure of the Convolutional Neural Network, and the functions of different layers in a neural network. The second part asks us to train the Le-Net 5 architecture using the original parameters as well as different parameters. The third part asks us to change the architecture of the neural network to get higher accuracies.

Approach and Procedure:

Each Convolutional Neural Network can be considered as a set of blocks, blocks being called as layers. Layers help us define the number of parameters in a neural network architecture. The traditional Le-Net 5 architecture is given below:



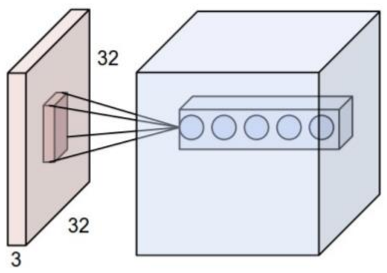
The architecture consists of 2 convolution layers, 2 pooling layers and 2 dense layers.

The input image is first fed to the Convolution layer, which finds the convolution between the image the filters. There are 6 filters used in the first convolution layer. The output of the first convolution layer is fed to the second layer which is the sampling layer in which the dimension of the image is reduced. The third layer is the second convolution layer in which 16 filters are applied to the image. The fourth layer is the sampling layer, where the dimension of the image is further reduced based on the pooling criteria. The fifth layer is the fully connected layer having 120 neurons, where the image is converted to a 1D vector converting it to a traditional neural network structure. The sixth layer is again a fully connected layer having 84 neurons and the last layer is the soft-max layer having 10 outputs, due to the number of classes for the MNIST dataset ranging from 0-9, the number of digits. The different layers are explained below:

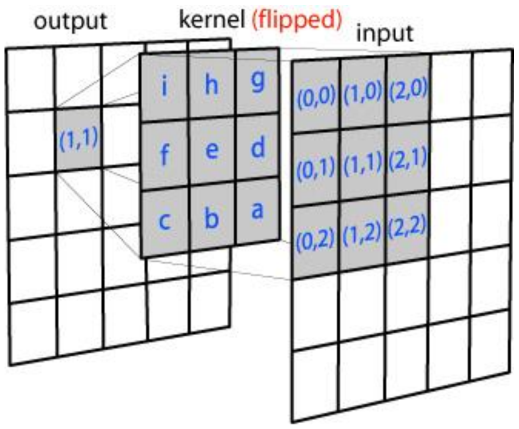
## Convolution Layer:

This is the very first layer of the architecture. The image is fed block by block to this layer. This layer implements the matrix multiplication with a block of the image. In the case of LeNet the dimension of filter is 5x5. Hence, this filter is multiplied with a 5x5 block of the image. There three main parameters controlling the convolution layer. The number of filters used, the spatial dimension of each filter, padding and strides. The number of filter define the number of feature images extracted for each image. In LeNet 5, the number of filters used for the first convolution layer are 6. The number of filters in the second convolution layer are 16. The spatial dimension used is 5x5 for both these stages. The second parameter as mentioned above is the filter dimension. This decides the number of pixels to be included for the pixel in consideration. In other words, this parameter specifies the number of neighboring pixels to be considered surrounding the pixel in consideration. In LeNet 5, both the convolutional layers have a spatial dimension of 5x5. The third parameter is the padding factor. The padding can be zero padding or reflective padding. When we consider the border pixels, there will be no pixels beside them as they are present in the boundary. When reflective padding is used, the pixels present on the other side of the boundary will be reflected and be considered as a boundary surrounding the border pixel. The only difference between zero padding and reflective padding is that, in zero padding zeroes will be added outside the boundary region and then convolved with the given filter. The last parameter is the stride parameter which defines the interval between the number of pixels to be considered for the convolution. When the stride is 1, each pixel of the image is considered and then convolution is applied. When the stride is 2, every alternate pixel is considered and convolution is applied. Generally, stride 1 is used. When the stride is 2, the output pixel number is drastically reduced. To consider every detail in the image, the stride must be 1.

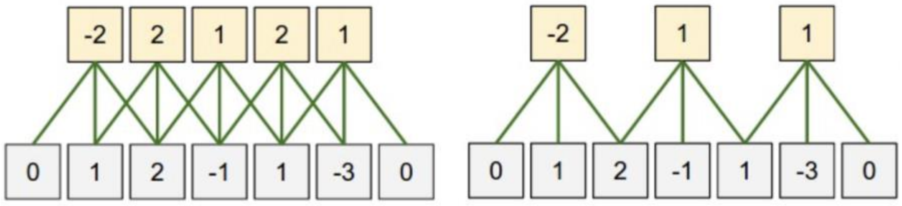
The convolution operation gives us a cuboid of pixels with the width depending on the number of filters used, and breadth and length depending upon the spatial dimension and strides. The following image gives an idea about the convolution operation.



In this case, the dimension of the input image is 32x32x3, 3 represents the number of channels in the image representing an RGB image. The convolution image can be easily visualized with the following image.



A pixel along with its neighboring pixels are considered and then multiplied with the kernel function, otherwise called as filter. The influence of strides can be understood with the following image which uses a 3x3 kernel.

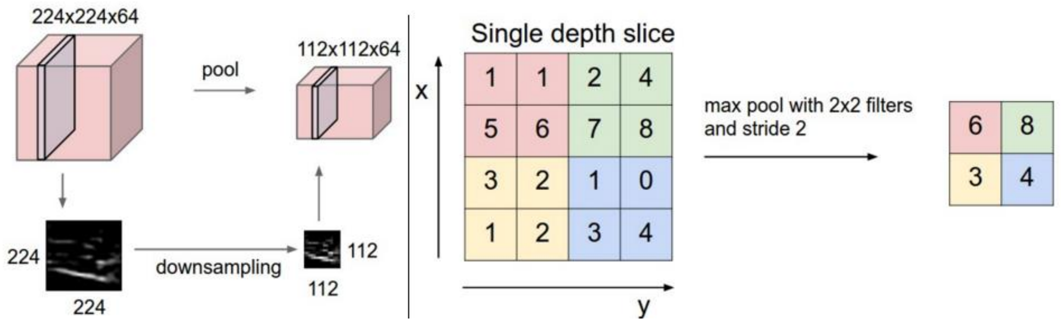


Stride: 1 Stride: 2

When the stride is 1, every pixel is considered during the scanning stage of the convolution layer and when the stride is 2, alternating pixel is considered. The stride mainly depends on the spatial dimension of the filter as well. When the spatial dimension of the filter is very large, stride can be considerably high. When the filter dimension is small, the stride has to be 1 otherwise, important details from the image can be lost. The output of the each of the filter is then applied to an activation function. The activation function will be explained later.

Pooling Layer:

Pooling Layer down-samples the image. For example, if the input image is 28x28 and if 2x2 pooling is used, the output image dimension is 14x14. This considers every 2x2 block of the image and assigns only one output for each 2x2 block based on different criteria such as maximum, average etc. This helps reduce the complexity of the problem before the image is passed for the next layer. The only parameter for the pooling layer is the window size, which decides the region to be reduced. This is decided by the stride factor for the pooling layer. If the stride is 2, then one pixel for every 2x2 block is generated. No matter what the window size is, the output is just one pixel. The LeNet 5 architecture uses two pooling layers each after the convolution layers. Pooling layer is a very important layer as it reduces the complexity of the neural network before the fully connected layers. The pooling layer can be understood with the following image.



In this image, max pooling is used and the stride is 2. Another type of pooling is average, which takes the average of the four pixels and gives one output. The main use of this layer is to reduce the number of neurons in the fully connected layer ultimately. The output of the pooling layer also will be applied to an activation function.

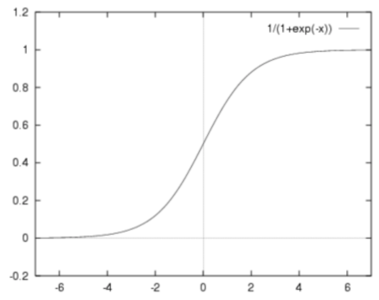
Fully Connected Layer:

The outputs from the max pooling layer after applying them to an activation function are converted to a one-dimensional vector. For example, in case of the LeNet 5 architecture the output of the second pooling layer is 16*5*5 which stretches out to become 400*1. This is the input to the fully connected layer. The main difference between the fully connected layer and the convolution layer is that, all the input nodes are connected to all the neurons. This is leads to the factor of non-linearity in the classifier. With the help of the fully connected layers we can created non-linear combinations of the different input features in that particular feature space. The main use of the fully connected layer is to reduce the number of inputs to the soft-max layer. They are usually present in the end of the neural networks for 2 main reasons. The first reason is that, if the fully connected layer is present in the beginning, the number of neurons will be extremely high and the computation is very high. Also, it has been proved that the presence of fully connected layer in the end improves the classification accuracy and also a faster convergence to the parameters. The second reason is that, fully connected layers do not preserve the spatial structure in the image and hence there is no point in applying the convolution layer after the fully connected layer. These are the reasons why the fully connected layers are in the end. Thus, the functionality of the fully connected layers is to reduce the number of inputs to the soft max layer, by creating linear combinations of all the input nodes with the help of a full connection and neuron weight and bias. The outputs of the fully connected layers are also fed to the activation functions to give the non-linearity to the neural network.

Activation Function:

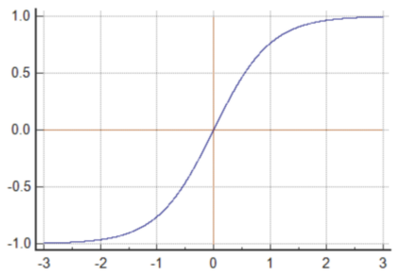
Activation function is one of the important features present in a neural network, and one of the main reasons why neural networks are able to perform very well for the Image/Audio classification and Object Recognition tasks. The main use of the activation function is to introduce non-linearity to the neural network. The activation function gives a non-linear mapping between the input and gives the output which is fed as the input to the next layer. Without these activation functions, neural networks will just behave as a linear regressor and the performance will go down. With the help of these activation functions we can learn any given function and the power of the neural networks depend on these activation functions. One of the important features of the activation functions is that they should be differentiable. This is very much needed during the back-propagation stage. Algorithms such as Gradient descent need these functions to be differentiable, otherwise they do not give any information to the learning algorithm and make it difficult for the algorithm to learn the weights, and sometimes the weights may not converge at all.

There are three popular activation functions namely Rectified Linear Units (ReLu), Sigmoid and Tanh (Hyperbolic Tangent). The graph of the sigmoid activation function is given below:



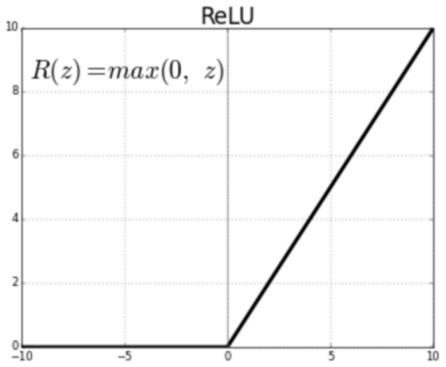
The mathematical formula for the sigmoid activation function is, $\frac{1}{1+e^{-x}}$. Even though the sigmoid activation function is differentiable, it is not preferred these days because it kills the negative components and makes them zero and it is not centered at zero. Also, they give a very slow convergence rates for the learning algorithm since they saturate very easily.

The second activation function is the tanh activation function. The mathematical formula for the tanh function is, $1 - \frac{e^{-2x}}{1+e^{-2x}}$ . The plot of the function is given below:



The main advantage with tanh when compared to the sigmoid activation function is that, the output is centered at 0. But it still saturates easily and hence the convergence rate is reduced for the learning algorithm. Hence, the tanh functions are not widely used even though they are better than the sigmoid activation function.

The most popular activation function right now is Rectified Linear Units (ReLu).



ReLu overcomes the problem of gradient vanishing when compared to the sigmoid and the tanh activation functions as there is mapping for all the values in the input range. Also, the function never saturates as it is always present for any value of input, whereas in Sigmoid and Tanh, the outputs saturate at +1 or -1 unlike in ReLu. There is a one-to-one mapping between the input and output in terms of ReLu and at the same time, the function is differentiable through its entire range. The only disadvantage with ReLu is that, it kills the negative components and makes them zero. To overcome this issue, leaky ReLu was discovered to not kill the negative values. ReLu is used in almost all the present-day CNN architectures.

Softmax Function:

One of the most important things to note is that, the activation function can be used only with the hidden layers. The problem is that, the outputs of the activation function can vary between any ranges and does stay between 0 and 1. If we consider the final layer which has nodes equal to the number of neurons, it conveys the output classes. If we use the Softmax function, it gives us outputs in the range of 0 and 1 which effectively conveys the probability with which the given input belongs to each of the class outputs. It is very similar to a linear regressor, just that the values lie between 0 and 1 and the sum must always be equal to 1. The output of the Softmax layer is the likelihood ratio of the output, giving us the value of probability that it belongs to each of the classes. We can assign the object to the class which has the highest probability. This is how the Softmax layer functions. The Softmax function can be used in the hidden layers as well, unlike the activation functions which can be used only in the hidden layers. Thus, the main function for the Softmax layer is that, it can limit the values between 0 and 1 and the sum of all values is equal to

1. This is mainly used as the output layer which conveys the information of the class to which the object belongs to.

Loss Function:

Loss function is used to calculate the difference between the true labels and predicted labels. The loss function is fed to the Back-Propagation algorithm to minimize the loss during the next iteration. The loss function converts the categorical values into a numerical value that an optimizer can work on to minimize the loss function by tuning the network parameters. There are two main loss functions depending on problem. They are Mean squared error loss and the cross-entropy loss function.

Mean Squared Error (MSE):

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

MSE is mainly used for regression type of problems where the output is a number instead of a category. In the case of MSE for CNN, the function may not be convex, and hence the global minimum of the loss function cannot be reached. And, since the true values are one-hot vectors and the output of the Softmax function is a probability value, the error function cannot help us to find the optimum parameters. For categorical output problems, categorical cross-entropy is the widely used loss function.

Categorical Cross-Entropy:

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

Cross Entropy was initially used for finding the signal losses in a binary signal transmission. Now it is widely used for these purposes for a multi-class classification problem. Since it considers the log of the predicted labels $y_i$, it converts the probabilities to logarithm scale and then compares with the true value which is a vector with only value being 1 and others being 0. Categorical Cross-Entropy has proved to be an important loss function and state of the art optimization techniques such as Adam Optimizer seem to work very well and give a fast convergence when categorical cross entropy is used. Thus, the main function of the loss function is to provide a relation between the true labels and the labels predicted by the classifier. This is the cost function for a CNN classifier. This is done for every iteration and fed to the back-propagation stage.

## Back Propagation:

Back propagation is the important step for the learning process for the supervised learning process. This mainly depends on the error function generated. The error function gives us the value based on the predicted and the target values. If the target values are very different from the true values, then the error function value will be very high. The main purpose of the learning phase is to decrease the error function by tuning the filter weights and the bias values. By doing this in an iterative process, the right filter weights and the biases can be adjusted in such a way that they minimize the loss function. The power of the learning algorithm depends on the differentiability of the learning function. The differential of the error function is taken and the gradient descent algorithm is trained on the given loss function.

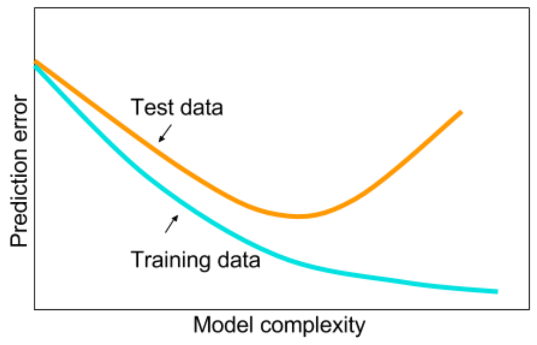
There are two main steps in the back-propagation algorithm:

1) Error Propagation
2) Weight Update

In the error propagation step, the error produced by the loss function is distributed in such a way, that it affects the weights of all the stages in the neural network. The optimization algorithm works to reduce the loss function for the next iteration, by modifying the weights in all the layers. All the layers, starting from the first layer until the output layer, the weights are changed in order to reduce the error function by a maximum value. A typical optimization algorithm is the Gradient Descent, which reduces the error in the direction of the steepest descent of the loss function. The weight update step of the back-propagation algorithm essentially consists of the part in which the weights of all the layers are modified to reduce the error during the next iteration. The back-propagation algorithm puts the constraint on the activation function that, it must be differentiable.

## Over Fitting Issue:

Over-Fitting is a very important problem in supervised learning algorithm. When the model is over fit, it becomes very specific for the given set of training data points and does not perform very well when unknown test data is provided to the system. The training data has some outliers and noise present in it. When the classification model becomes very specific to the given training data points, then the model is over-fit. The model will give very high accuracy for training data and the accuracy will be very low for test data. The following graph can explain the problem of over-fitting.



Preventing Over-Fitting:

Thus, if the model is over tuned, it will become very specific to the given data points and will give very low accuracy to the test data. Hence, this issue must be avoided in all cases.

The following steps can be used to reduce the problem of over-fitting:

1) Add more training data (if it is possible)
2) Data Augmentation
3) Add regularization (Dropout)
4) Reduce the architecture complexity

Adding more training images, increases the training time but reduces the problem of over fitting. When more training images are added, the model becomes more diverse and under the assumption that the model cannot become very specific for a system with a larger number of images, over-fitting can be reduced. The second idea is to augment the data. In terms of images, this means feeding different orientations of the same image and increasing the variety of images available to the training model. Data can be augmented by rotating the images, zooming in/zooming out, etc. The third idea is to regularize the model. The most common regularization used in CNN is dropout. In dropout, the output of neurons which give a very low output value are made to be 0. This is because, if a very low output is fed as input to the next layer, it does not affect the outputs of the next layer. And by giving importance for these low outputs, the model may become over-fit. Hence, by using the dropout value, all the outputs less than the dropout value are killed. One important thing to note it, dropout causes information loss and hence it is safer to use it only in the final stages of the network. If it is used in the initial stages, then there is a lot of chance that important features will be killed even before they are fed to the final layers. Hence it is a good practice to gradually increase the dropout value from the initial layers to the final layers. The fourth idea is to reduce the architecture complexity which implies to reduce the number training parameters which may reduce the issue of over-fitting by not training the model specific to given set of input images and tuning the large set of parameters.

Why CNNs are better than traditional other traditional methods:

CNNs have revolutionized the field of computer vision. CNNs are widely used in Object detection, Image classification, Semantic Segmentation, Scene Classification, Object tracking and so on. They have revolutionized the self-driving car markets which is mainly because of this new subject called 'Deep Learning'.

The main difference between CNN and traditional image classification algorithms is that, in traditional algorithms we need to specify the algorithm to look for specific features designed by us. But in CNN, the algorithm itself decides the best features to be extracted. The algorithm can learn what type of features to look for in the images and then extract them and use it for the classification procedure. For example, in Law's Filters for texture segmentation we need to specify the kernels needed for feature extraction. The filters are decided in such a way, that they can identify any texture when the input image is fed to them. After the features are extracted, traditional machine learning algorithms are used to train the classifier. In CNN, the kernels are identified by the Deep Learning system. The system knows what type of features to look for, to classify them correctly with minimum error rate. This is the main difference between CNN and traditional Image classification algorithms. The CNNs can learn to extract the features specific for an image and then extract the features and then train it using a traditional neural network classifier. Thus, the main reason why CNNs have a higher accuracy when compared to the traditional Image classification algorithms is that, the CNNs know what type of features to look for in the images by tuning the filter kernels by itself but in traditional image classification techniques, we need to specify the classification algorithm what type of features to look for.

Implementation:

The Le-Net 5 architecture is implemented using Keras. Keras is a package running on Tensorflow backend. A sequential model can be created and the different layers can be added by specifying the parameters for each of the layer such as number of filters, kernel size, kernel initializer, convolution padding, convolution stride, pooling stride and type, dropout, activation, loss function, the optimizer and its parameters. We have to specify the batch size per epoch and the number of epochs as well.

We can train the model with these parameters batch by batch for every epoch and the optimizer will change the weights according to the loss function obtained. Once the model is trained, we can use the trained model to find the test accuracy. The training accuracy will be higher than the test accuracy generally. Also, we can track the accuracy change during every epoch by testing the model on the validation set.

The architecture of LeNet-5 is summarized below:

First Convolution Layer: 6 filters with dimension 5x5

Second Pooling Layer: Stride is 2 and average pooling is used

Third Convolution Layer: 16 filters with dimension 5x5

Fourth Pooling Layer: Stride is 2 and average pooling is used

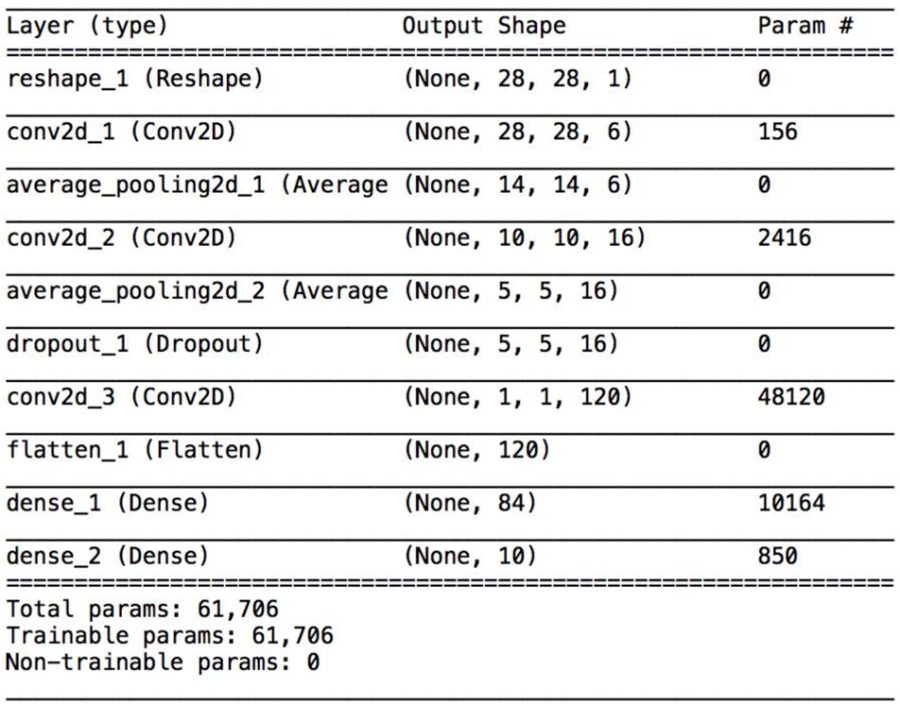
Fifth Fully connected layer: 120 neurons are used

Sixth Fully connected layer: 84 neurons are used

Seventh Fully connected layer: Has 10 output nodes, Softmax function

Optimizer: LeNet-5 was originally optimized with Stochastic Gradient Descent

The Summary of the LeNet-5 model before training is given below:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 16)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 120)         48120
_________________________________________________________________
flatten_1 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

The number of training parameters are 61,706 for the built LeNet 5 model. In the next section, we will see the results for different model parameters and their effect on the overall accuracy.

Experimental Results:

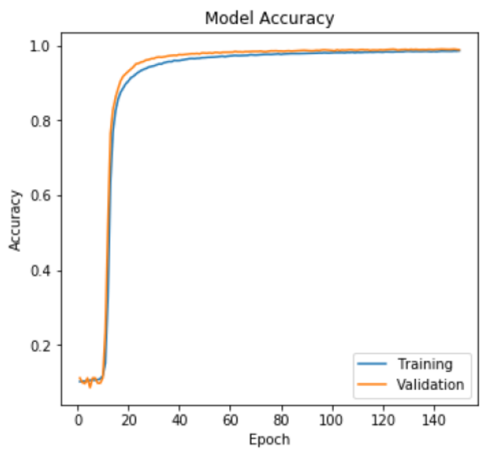
For implementing the original architecture of LeNet5, the activation function used is 'Sigmoid' and the optimizer used is Stochastic Gradient Descent, 6 filters for the first convolution layer, 16 for the second convolution layer and the pooling used was 'Average'. The decay parameter used is 'default' from the Keras package.

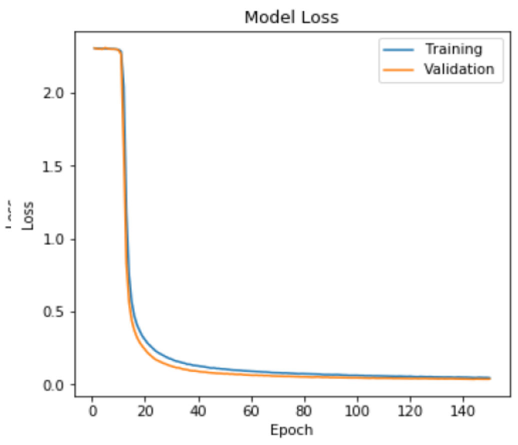
| Kernel Initializer | Learning Rate (SGD) | Momentum (SGD) | Batch Size | Train accuracy (%) | Test accuracy (%) |
|---|---|---|---|---|---|
| **Random Normal** | 0.1 | 0.5 | 128 | 99.91 | 99.2 |
| **Random Normal** | 0.3 | 0.3 | 128 | 99.85 | 99.14 |
| Random Normal | 0.3 | 0.5 | 128 | 99.98 | 99.25 |
| Random Normal | 0.1 | 0.25 | 64 | 99.38 | 99.08 |
| **Random Normal** | 0.1 | 0.5 | 32 | 99.74 | 99.26 |
| **Glorot Normal** | 0.1 | 0.5 | 128 | 99.48 | 99.16 |
| Glorot Normal | 0.3 | 0.5 | 128 | 99.83 | 99.17 |
| **Glorot Normal** | 0.3 | 0.7 | 200 | 99.90 | 99.22 |
| **Glorot Normal** | 0.5 | 0.5 | 128 | 99.85 | 99.25 |
| Glorot Normal | 0.1 | 0.25 | 64 | 99.3 | 99.04 |

| | | | | | |
|---|---|---|---|---|---|
| **Glorot Normal** | 0.1 | 0.5 | 32 | 99.7 | 99.22 |
| Random Uniform | 0.1 | 0.5 | 128 | 99.13 | 98.99 |
| Random Uniform | 0.3 | 0.5 | 128 | 99.27 | 99.01 |
| **Random Uniform** | 0.5 | 0.5 | 128 | 99.81 | 99.17 |
| **Random Uniform** | 0.1 | 0.25 | 64 | 99.69 | 99.19 |
| Random Uniform | 0.1 | 0.5 | 32 | 99.65 | 99.12 |
| **Truncated Normal** | 0.1 | 0.5 | 128 | 99.9 | 99.14 |

Case (i):

Initializer: Random normal, LR: 0.1, Momentum: 0.5, Batch size: 128

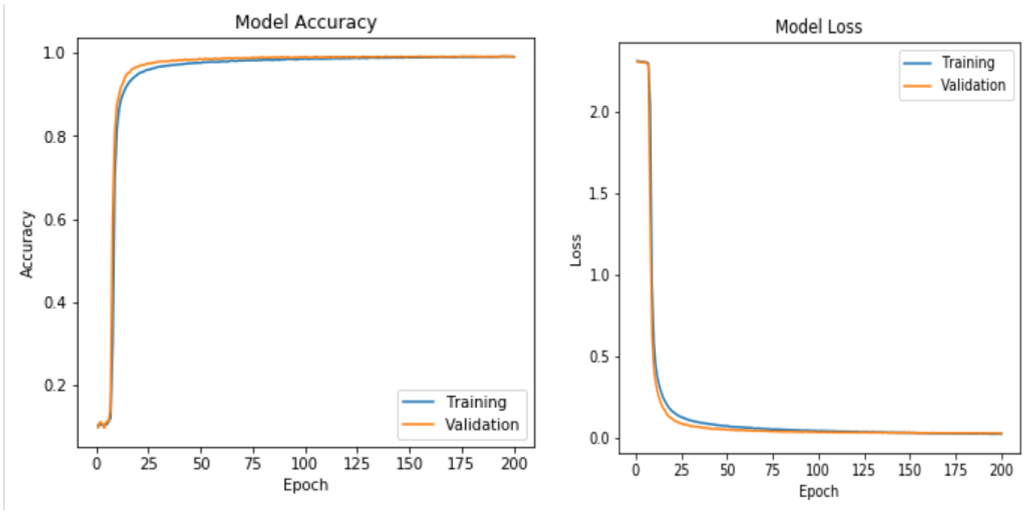




Training Accuracy: 99.91%, Testing Accuracy: 99.2%

Case (ii):

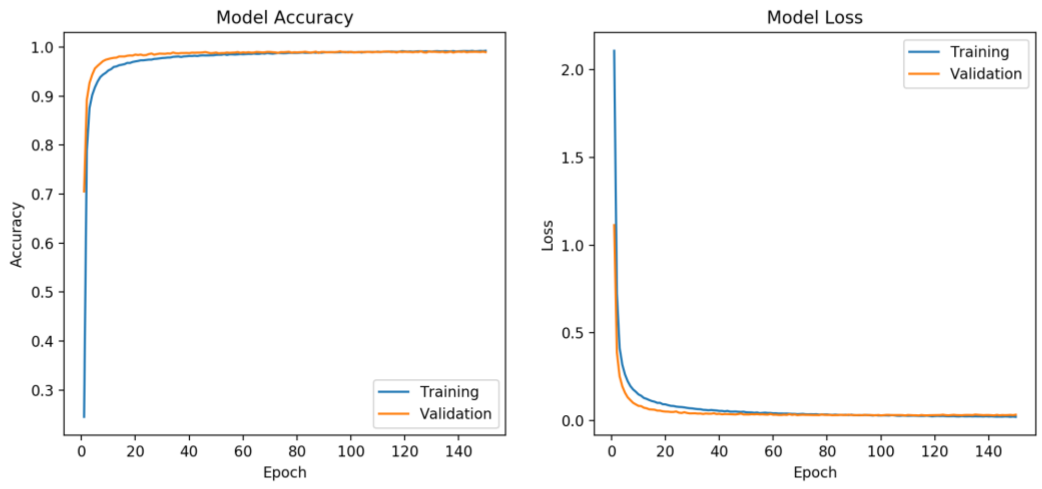
Initializer: Random normal, LR: 0.3, Momentum: 0.3, Batch size: 128



Training Accuracy: 99.85, Testing Accuracy: 99.14%

Case (iii):

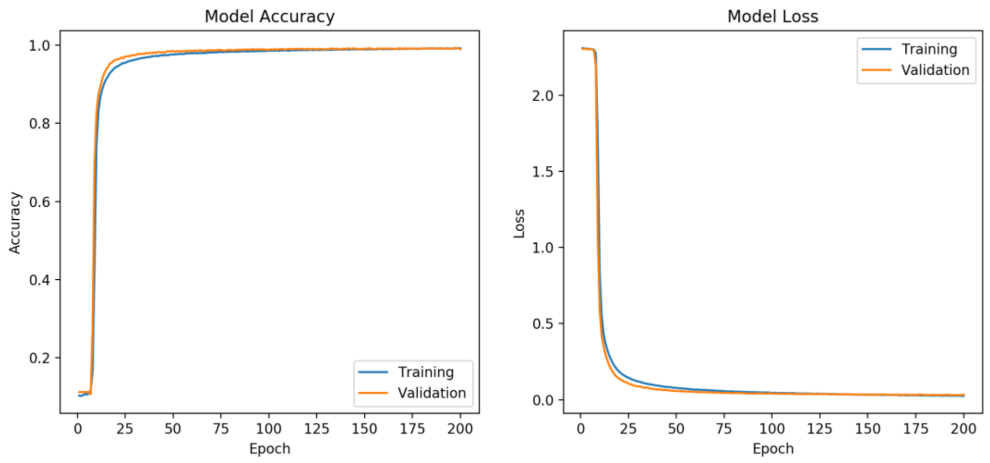
Initializer: Glorot normal, LR: 0.3, Momentum: 0.7, Batch size: 256



Training Accuracy: 99.48%, Test Accuracy: 99.16%

Case (iv):

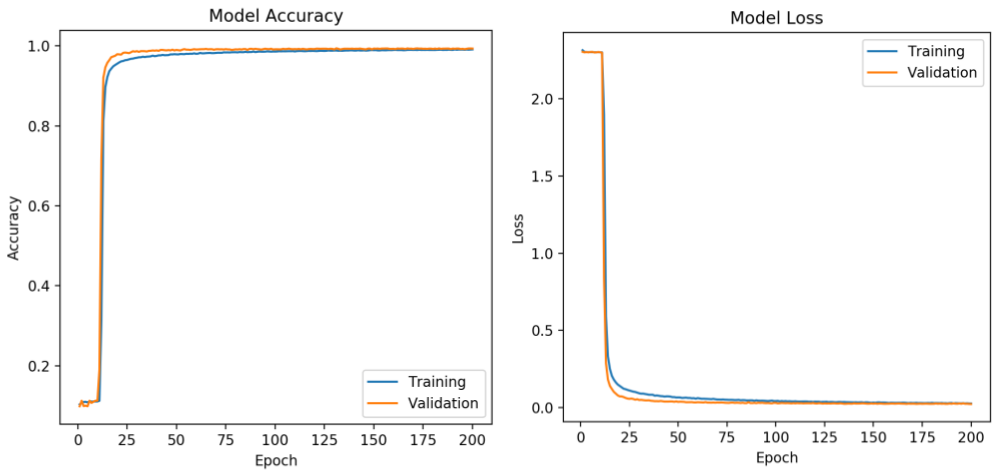
Initializer: Truncated normal, LR: 0.1, Momentum: 0.5, Batch size: 256



Training Accuracy: 99.9%, Test Accuracy: 99.14%

Case (v):

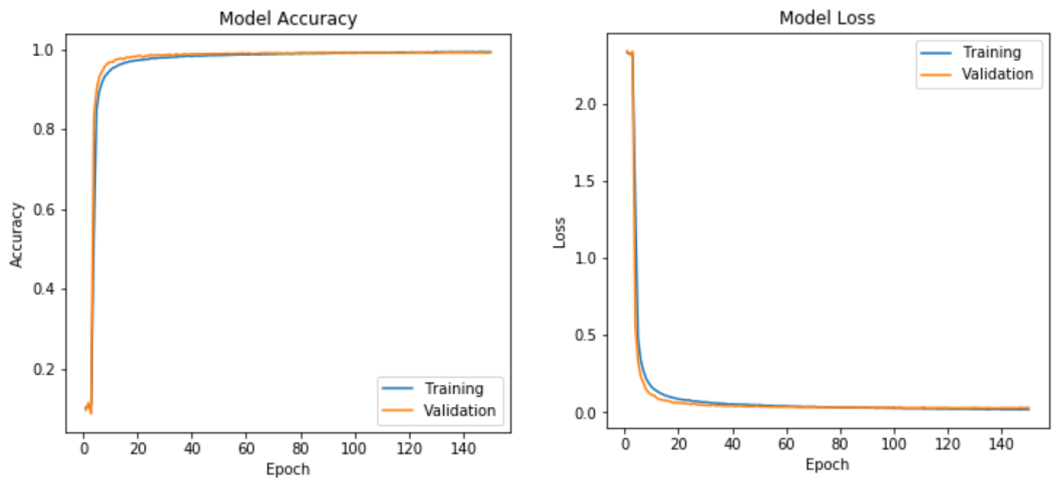
Initializer: Random Normal, LR: 0.1, Momentum: 0.5, Batch size: 32



Training Accuracy: 99.74%, Test Accuracy: 99.26%

Case (vi):

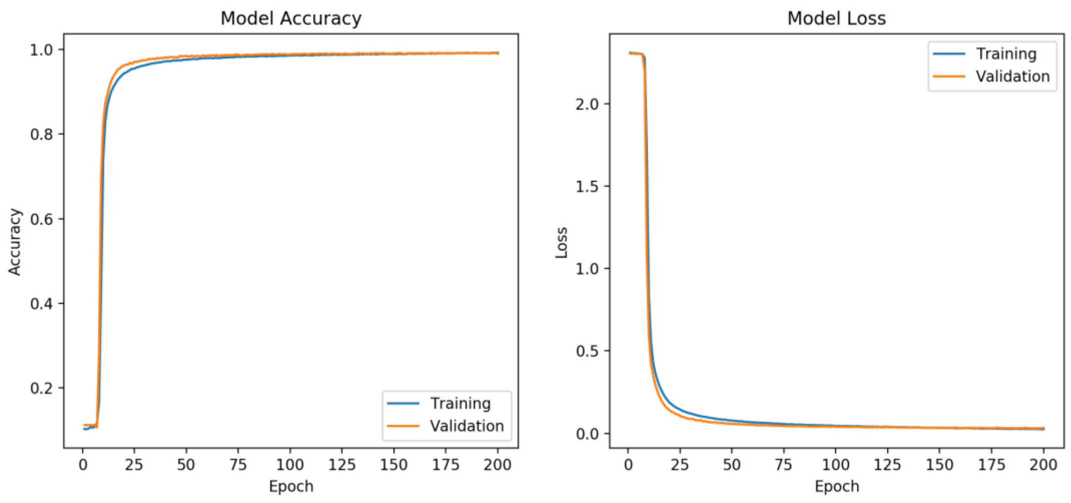
Initializer: Glorot Normal, LR: 0.5, Momentum: 0.5, Batch size: 128



Training Accuracy: 99.95%, Test Accuracy: 99.25%

Case (vii):

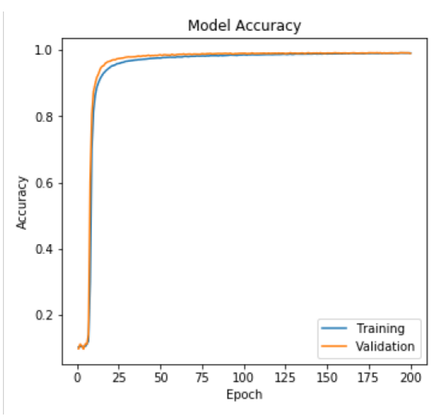
Initializer: Glorot Normal, LR: 0.1, Momentum: 0.5, Batch size: 32

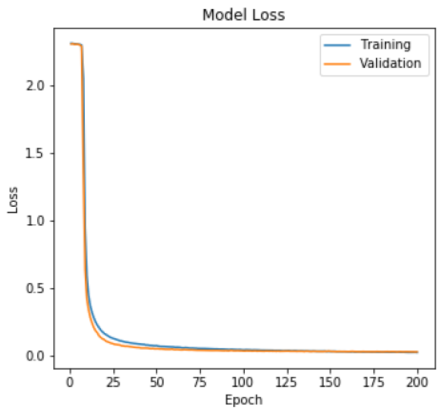


Training Accuracy: 99.7%, Test Accuracy: 99.22%

Case (viii):

Initializer: Random Uniform, LR: 0.5, Momentum: 0.5, Batch size: 128

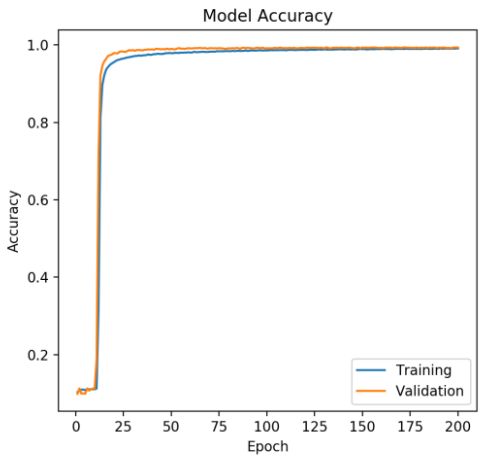


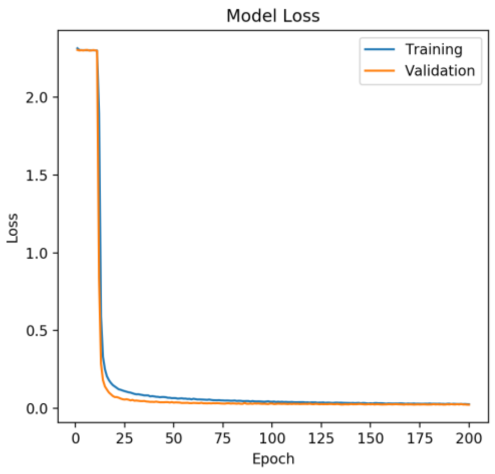


Training Accuracy: 99.81%, Test Accuracy: 99.17%

Case (ix):

Initializer: Random Uniform, LR: 0.1, Momentum: 0.25, Batch size: 64

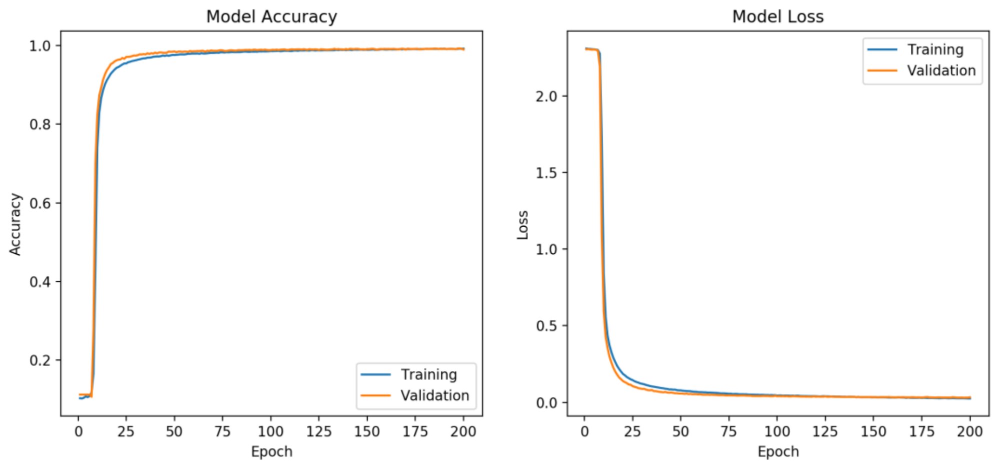




Training Accuracy: 99.69%, Test Accuracy: 99.19%

Case(x):

Initializer: Glorot Normal, LR: 0.1, Momentum: 0.5, Batch size: 128



Training Accuracy: 99.48%, Test Accuracy: 99.16%

<u>For Developing the LeNet-5 Architecture:</u>

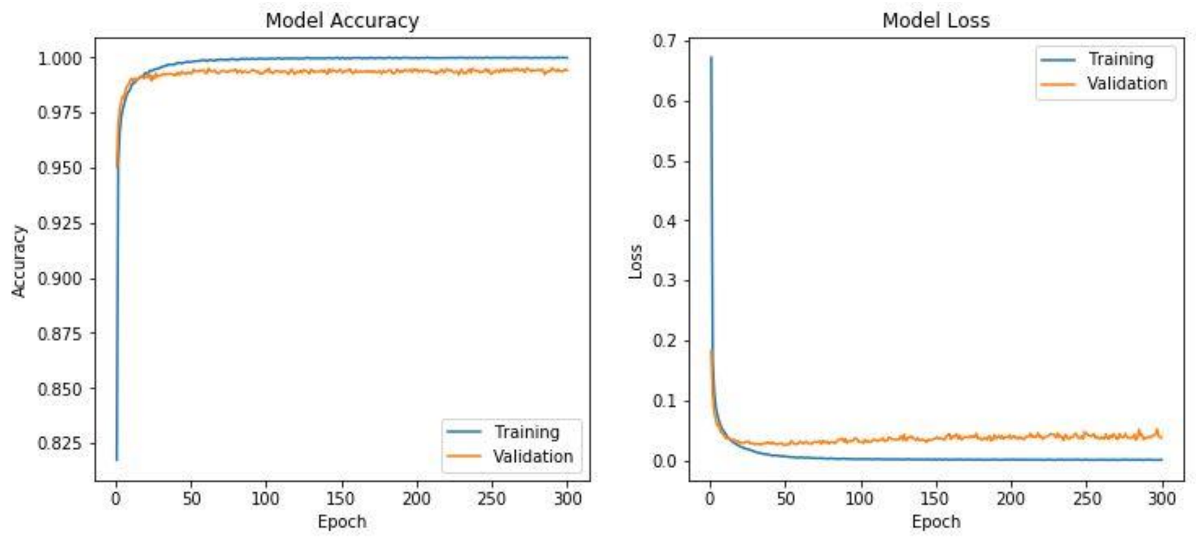
Some changes were made to the original LeNet-5 architecture to improve the accuracy. There are three main changes in the following models when compared to the original LeNet-5 architecture. The first difference was changing the activation function from sigmoid to ReLu. The second change was changing the pooling from average to max and the third change is, to introduce a dropout parameter before the first fully connected layer. The optimizer was also changed from SGD to Adam optimizer. A dropout of 0.25 has been used in all the cases before the fully connected layer. The following table summarizes the changes in the LeNet-5 architecture and the results.

| Conv1 Filters | Conv2 Filters | Dense Layers | Activation Function | Optimizer | Train Accuracy | Test Accuracy |
|---|---|---|---|---|---|---|
| 32 | 64 | 120, 84 | ReLu | Adam (lr: 0.001) | 99.95% | 99.32% |
| 32 | 64 | 120, 84, 42 | ReLu | Adam (lr:0.001) | 99.97% | 99.37% |
| 32 | 64 | 120, 84 | ReLu | Adam (lr:0.0001) | 99.975% | 99.39% |

| | | | | | | |
|---|---|---|---|---|---|---|
| 50 | 100 | 120, 84 | ReLu | Adam (lr:0.0001) | 99.985% | 99.44% |
| 32 | 64 | 120, 84 | ReLu | Adam (lr:0.0001) | 99.998% | 99.54% |

Case (i):

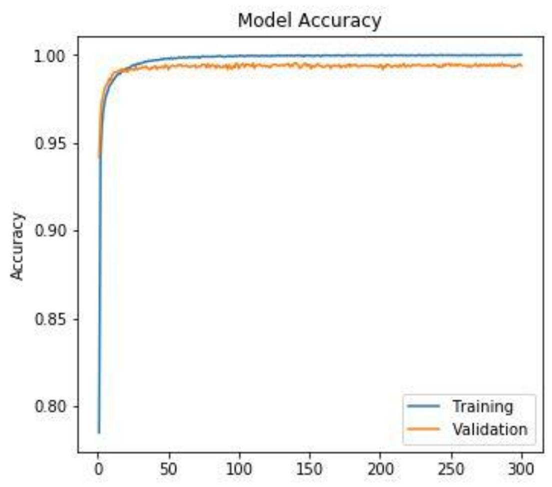
Conv1 Filters: 32, Conv2 Filters: 64, Dense Layers: 120, 84 Activation Function: ReLu, Optimizer: Adam (lr:0.001)

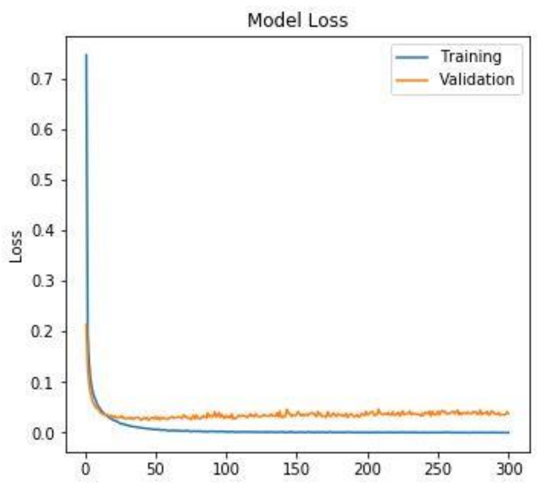


Training accuracy: 99.95%, Testing accuracy: 99.32%

Case (ii):

Conv1 Filters: 32, Conv2 Filters: 64, Dense Layers: 120, 84, 42 Activation Function: ReLu, Optimizer: Adam (lr:0.001)

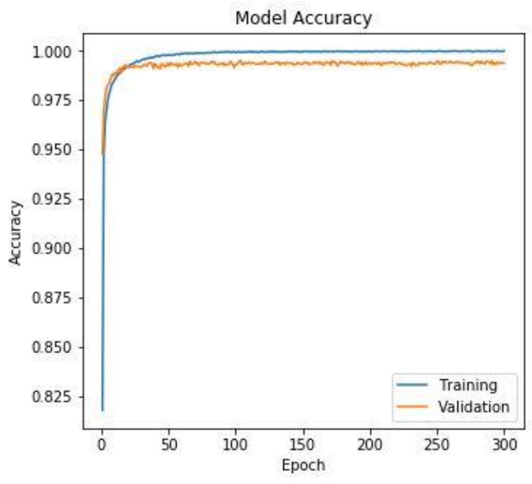


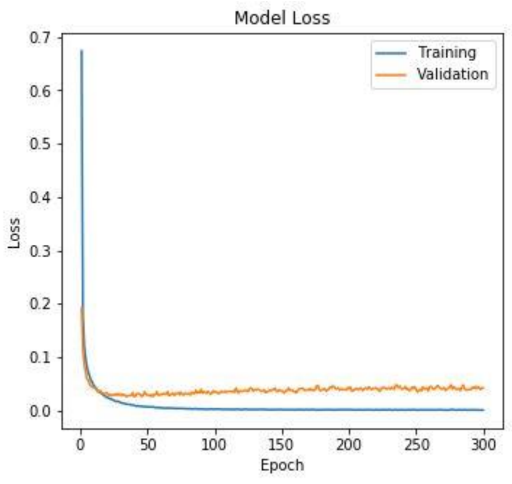


Training accuracy: 99.97, Test Accuracy: 99.37%

Case (iii):

Conv1 Filters: 32, Conv2 Filters: 64, Dense Layers: 120, 84 Activation Function: ReLu, Optimizer: Adam (lr:0.0001)

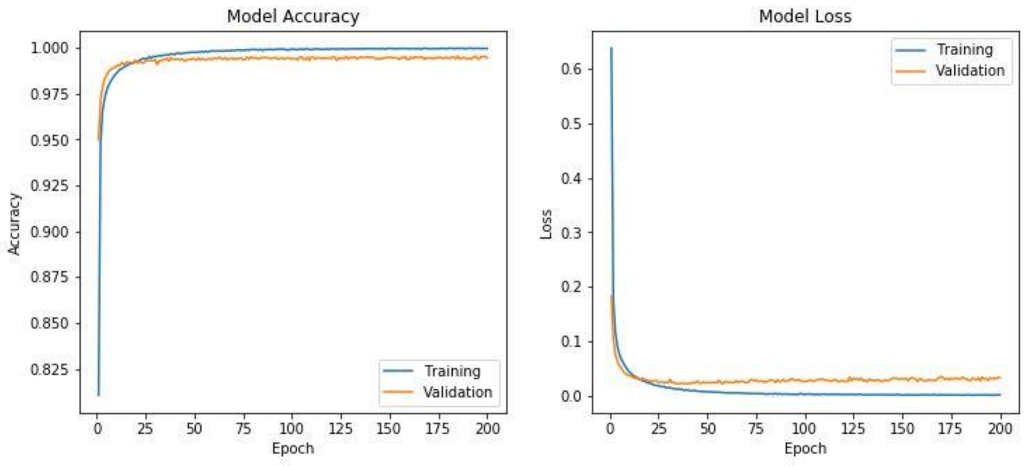




Training accuracy: 99.975%, Test Accuracy: 99.39%

Case (iv):

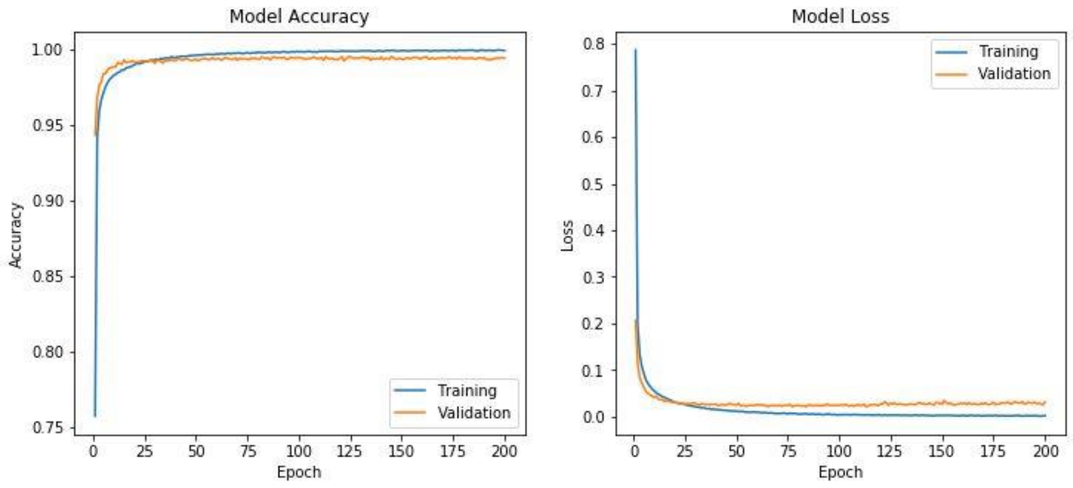
Conv1 Filters: 50, Conv2 Filters: 100, Dense Layers: 120, 84 Activation Function: ReLu, Optimizer: Adam (lr:0.0001)



Training accuracy: 99.985%, Test Accuracy: 99.44%

Case (v):

Conv1 Filters: 32, Conv2 Filters: 64, Dense Layers: 120, 84 Activation Function: ReLu, Optimizer: Adam (lr:0.0001)



Training accuracy: 99.99%, Test Accuracy: 99.54%

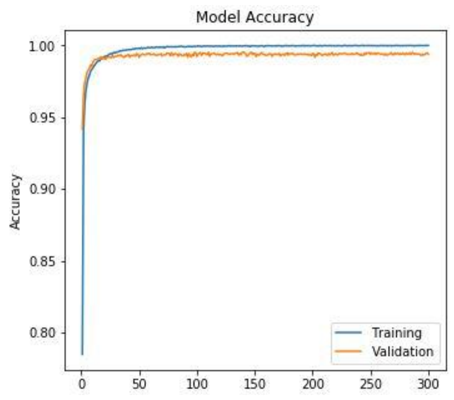
For the following cases, the number of convolution layers were also increased and the optimizer was changed from Adam to Adadelta

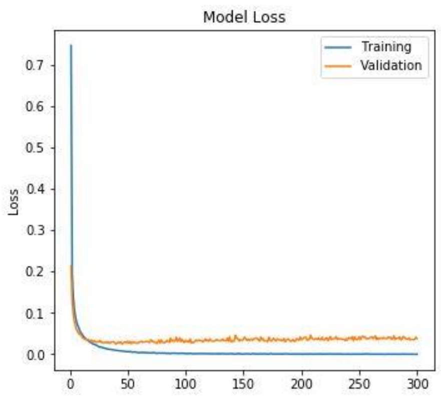
Case (i)

Convolution Layers: 3

Convolution filter in each layer: 32, 64, 128

Optimizer: Adadelta (lr: 0.001)





Training accuracy: 99.99%, Test Accuracy: 99.44%

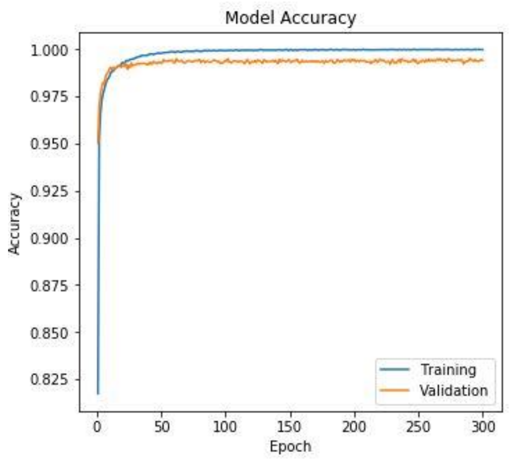
Case (ii)

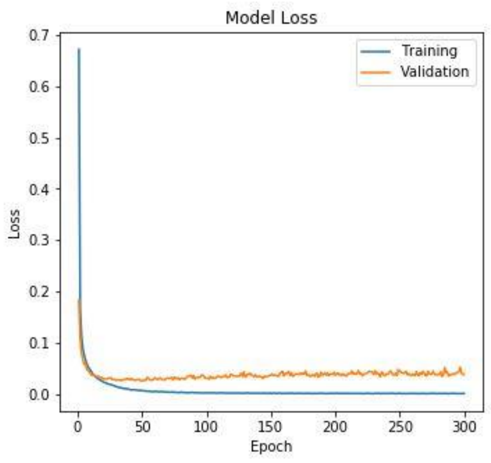
Convolution Layers: 3

Convolution filter in each layer: 32, 64, 128

Optimizer: Adadelta (lr: 0.001)

Kernel Initializer: Random Uniform





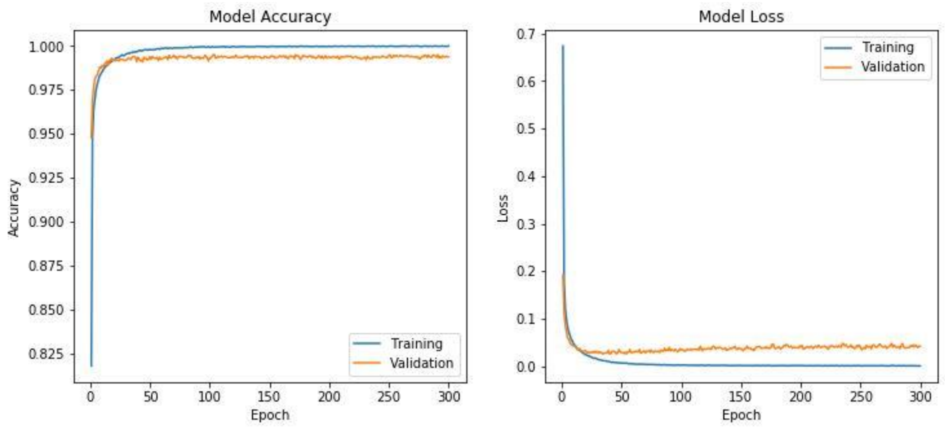
Training accuracy: 99.98%, Test Accuracy: 99.47%

For the following case, the kernel size was changed from 5x5 to 3x3 and other parameters are given below:

Convolution Layers: 3

Convolution filter in each layer: 32, 64, 128

Optimizer: Adam (lr: 0.001)



Training accuracy: 99.94%, Test Accuracy: 99.42%

Different parameters have different accuracy curves and model loss. We can discuss the effects of the different parameters used in the LeNet-5 architecture.

First let's talk about the effects of initializing different network parameters. The parameters of the network that have to be initialized are filter weights, learning rate for the optimizer, batches, epochs, and dropout parameter.

<u>Filter weights:</u>

Filter weights for any model are initialized during the very first step. In Keras, there are different weight initializers such as Random Normal, Truncated Normal, Random uniform, Glorot Normal etc. Glorot Normal is used as the default weight initializer. The kernel initializer plays a major role in deciding the convergence to the filter weights for the model. For example, the kernel initializers may start with 0 as well. But the loss function may pertain for a while before the final region for the loss function is reached. This is because, when the weight initializer is 0, all the filters will have the same output for the very first iteration and hence it will become a problem for the optimizer to work on. Hence, deciding the kernel initializer plays a vital role in the convergence of the filter weights to their final values. Glorot normal is proven to be the best kernel initializer for many cases. Even from the graphs given above, we can see that Glorot normal has got slightly higher slope in the initial iterations when compared to the other kernel initializer. Hence, Glorot normal is the default kernel initializer in Keras.

Learning rate and Momentum:

Learning rate is the most important parameter for the optimization function. This parameter decides how much change there will be in the weights for every batch in case of SGD. In stochastic gradient descent optimizer, the weights will be updated for every batch in an epoch. The learning rate decides how fast the changes in the weights will take place. Ideally, the learning rate must be very low, for getting a very good model. If the learning rate is low, the number of epochs must be really high. If the learning rate is very high, there will be faster training but the accuracy of the model may vary. Generally, the learning rate is supposed to be very low. For SGD, for faster convergence higher learning rate can be used. If Adam optimizer is used, the convergence will be faster even for a lower learning rate as it is a better optimizer when compared to the SGD. In most of the current day deep networks, Adam optimizer is most used.

Momentum decides the influence of the previous weight changes in the current batch optimization. It adds a factor of the previous batch's weight update to this current batch, to help for faster convergence. If the weight change in the previous batch is very less, then the weight change for the current batch will also be low. Momentum can have values between 0 and 1. If momentum is 1, then the influence of the weight update in the previous stage will have more influence on the current stage and vice versa.

Epoch and Batch Size:

Epoch and batch size decide how fast the model will converge. If the number of epochs are very high, then the model will get trained slowly and at the end of training, we may have the best model. But the problem with increasing the epochs is that, the training time for the model will become greater. Batch size decides the optimization algorithm's performance. Most of the optimizers such as SGD and Adam, take images in batches and then perform the optimization procedure. For example, if the batch size is 128, then 128 images are taken at once and the gradient is calculated and the weights are updated. During the next set of 128 images, the previously updated weights are used. And this process is repeated until all the images in the training set are used to optimize the loss function. Batch size and epochs decide the convergence rate of the model. Also, increasing the number of epochs while training the model increases the validation loss after a certain point of epochs. Thus, increasing the number of epochs, leads to overfitting. To reduce this effect, dropout has to be added, when the number of epochs is high.

Dropout:

This value kills the neurons which have an output lesser than the dropout value. If the dropout is 0.5, all the neurons having an output lesser than 0.5 will be killed. This dropout parameter makes the loss function give less importance to the neurons which have very low output. And another important thing is that, dropout has to be used only before the fully connected layers. If dropout is used in the initial layers, vital information may be lost. Having dropouts of 0.25 or 0.5 for ReLu will be useful, but if dropout is used for sigmoid functions, then the results will not be as expected as the function itself exists in the range of 0 to 1 and saturates everything.

From the results of the first table, we can see that the convergence happens faster for the optimizers having learning rate greater than 0.1. The accuracies are mostly in the range of 99.2% for the original LeNet 5 architecture for most of the parameters. This is mainly because of the activation function and the average pooling used. For getting accuracies higher than that of the original LeNet-5, the activation functions were changed, convolution layers were added, fully connected layers were added and dropout was also added. The effect of each is explained below:

1) When the convolution layers were added, the model became more computationally intensive as the number of training parameters increased. But the accuracies were considerably high when the convolution layers were added. When the number of filters were increased, the accuracy increased but the training time increased and the learning rate has to be reduced.
2) When the Fully connected layers were added, the accuracies shot up at the same time training parameters increased, but not like number of parameters when the convolution layers were added. The model took lesser time to train. With the presence of dropout and the activation function to be ReLu the effect of increasing the number of neurons in the fully connected layers produced increased accuracy results as the neurons having lesser influence for all the classes were killed before being connected to the fully connected layer.
3) Changing the activation function from sigmoid to ReLu improved the results very much since Sigmoid function provided less variations to the neurons, resulting in less

discriminant power for the classifier. ReLu gives better results in terms of the model accuracies and faster convergence, mainly due to the reason that, the variance in the outputs of the neurons increases as there is no saturation in ReLu when compared to that of Sigmoid. Changing the pooling type also provided better results. When max pooling was used, the accuracy was at least 0.1% higher when compared to the models that used average pooling. Thus, the pooling also plays a major role in the model accuracy.

4) Changing the optimizer, gave changes in the convergence rates. SGD was very slow when compared to that of Adam and Adadelta. This is because, SGD is very traditional and doesn't utilize the present day computational power. Adam optimizer works very well when compared to that of SGD. For example, when SGD is used with a learning rate of 0.001, it takes a minimum of at least 300 epochs for convergence. In the case of Adam optimizer, even if the learning rate is low, the convergence is faster. Even with the learning rate of 0.0001, Adam optimizer can provide convergence in less than 200 epochs and it is more stable when compared to that of SGD, because it discards high variance weight updates. This is another important change made when compared to the original LeNet-5 architecture.
5) Changing the kernel size alters the number of parameters that need to be trained. For example, if the kernel size is 3x3 then the number of parameters for each filter will be 9, when compared to 25 for a 5x5 kernel dimension. The accuracy of the model may vary when the kernel size is decreased, but we cannot be always sure of increasing accuracy when the kernel dimension is increased. Other parameters also greatly decide the model performance when the kernel dimension is changed.

Of the models trained above using various parameters the best accuracy for the original LeNet-5 architecture was given by the following parameters:

Filters in first convolution layer: 6

Filters in first convolution layer: 16

Neurons in first fully connected layer: 120

Neurons in first fully connected layer: 84

Kernel Initializer: Random Normal

Learning Rate: 0.1

Momentum: 0.5

Batch size: 32

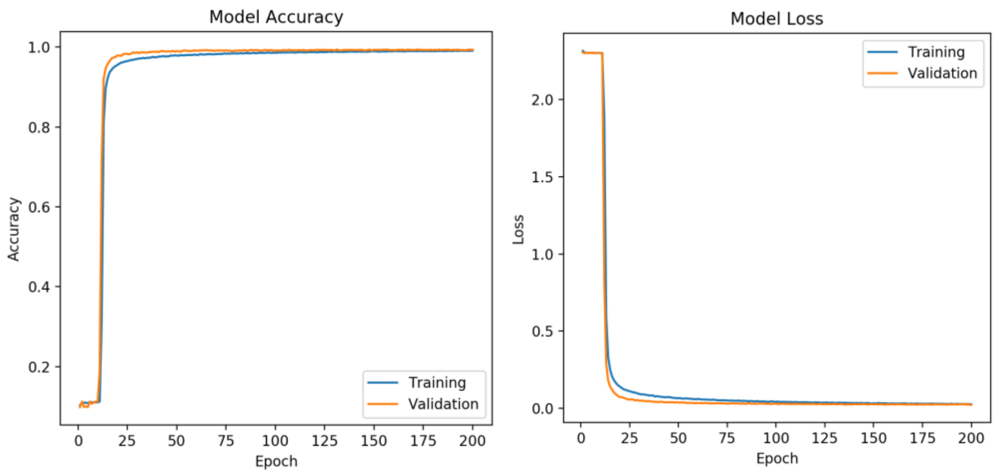
Optimizer: SGD

Batch size: 32

Pooling: Average

Training Accuracy: 99.74%

Test Accuracy: 99.26%



Best Result for the LeNet-5 architecture

By developing the original LeNet-5 model, many changes were made to the model. The following result gives the best improved model:

Kernel Initializer: Glorot Normal

Filters in first convolution layer: 32

Filters in first convolution layer: 64

Neurons in first fully connected layer: 120

Neurons in first fully connected layer: 84

Activation Function: ReLu

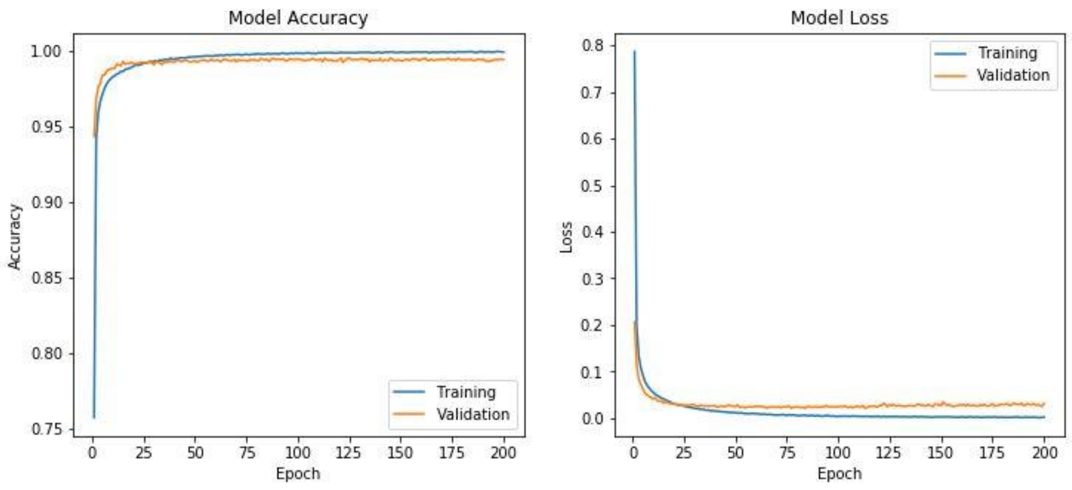
Optimizer: Adam

Learning rate: 0.0001

Batch size: 128

Training accuracy: 99.99%

Test Accuracy: 99.54%



Best result for the improved LeNet-5 architecture

Thus, the highest test accuracy obtained with the original LeNet-5 architecture is 99.26% and the highest test accuracy obtained with the improved LeNet-5 architecture is 99.54%.

## Problem 2: Saak Transform and Its Application to the MNIST Dataset (50%)

In this problem, please read and understand the paper "On Data-Driven Saak Transform" [5] introduced by professor Kuo.

**(a) Comparison between the Saak Transform and the CNN architecture (10%)**

Use your own language to explain the similarities and differences between the Saak transform and the CNN solution. The length should be at least of 1 page. Do not copy any sentences from [5] or other papers directly, which is plagiarism. The scores will depend on your degree of understanding.

**(b) Application of Saak transform to MNIST dataset (30%)**

The overall process of this task can be divided into three parts: (1) Saak coefficients generation, (2) Saak coefficients selection and dimension reduction, (3) classification. In the first part, it is a purely unsupervised process based on the statistics from the data. The details are given below.

*Step 1: Saak coefficients generation*

There are 5 stages in the Saak transform pipeline. Here are the detailed steps to find the Saak coefficients in each stage:

- For each input image (or data cube), select the non-overlapping patch region with spatial size 2x2. Then calculate the variance of each patch and remove the small-variance patches.
- Perform Principle Component Analysis (PCA) to the zero-mean patch data and get the PCA transform matrix. Transform all the input data patches by selecting the important spectral components in the PCA matrix.
- Augment transform kernels through the sign-to-position format conversion.
- Repeat this process for each Saak transform stage. For your convenience, the number of important component in each stage (total 5 stages) are: 3, 4, 7, 6, 8.

*Step 2: Saak coefficients selection and dimension reduction*

After getting responses from all Saak stages, you will get a feature vector of 1500 dimension. Perform the F-test on it and select features with larger F-test scores (around 1000 dimension). Then, perform another round of PCA to reduce the feature dimension to 32, 64 and 128.

*Step 3: Utilize images of the same class label to collect features of training samples and compare the performance of the SVM and the RF classifiers with three feature dimensions as given in Step 2.*

(1) Report the best classification accuracy that you can achieve on the train and test sets for all 6 cases.

(2) Compare the performance to the results in Problem 1 and briefly explain your observations.

**(c) Error Analysis (10%)**

Please compare classification error cases arising from the CNN solution and the Saak transform. What percentages of errors are the same? What percentages are different? Please give explanations to your observations. Also, please propose ideas to improve the CNN solution, the Saak transform solution or both and justify your proposal. There is no need to implement your proposed ideas.

## Problem 2: SAAK Transformation and its Application to the MNIST

Abstract and Motivation:

Convolutional Neural Networks take a lot of time for training, and the computational power required is very high. The model trained with CNNs are not robust and cannot be used without being retrained for a specific application. They are not very portable. The model has to be retrained for another application. This is the biggest problem with CNNs. Although their accuracy is very high, they cannot be used without being trained for all the specific applications. Also, choosing the number of filters and kernels dimension, deciding the number of convolution layers, number of hidden layers is a black art and there is not much mathematical background behind the working of the Convolutional Neural Networks. To overcome some of these issues, various methods have been tried to either understand the working of CNNs or to identify other methods using traditional signal processing techniques and methods having a strong mathematical basis.

One such method is SAAK transform. SAAK stands for Subspace Approximation with Augmented Kernels. This method is also based on finding the transformation kernels for each of the image, similar to CNNs. The most important difference between SAAK transform and CNN is that, in SAAK finding the transformation kernel is a one step process, whereas in CNN the transformation kernels are found in an iterative method. The main advantage with SAAK is that, it is not very specific to a particular application unlike CNN. SAAK trained model can be easily built, in a matter of hours whereas good CNNs take weeks to be trained. Another main advantage of the SAAK transform is that, the images can be reconstructed using the inverse SAAK transform which is not possible with CNNs as there is no inverse transform existing. The main objective of this question is to understand the working of the SAAK transform, and train different classifiers to get accuracies on training and testing sets.

Approach and Procedure:

SAAK transform uses the concept of Eigen vectors and Eigen values, which give a transformation for projecting the given vectors to the orthonormal space using the orthonormal transformation matrix. This is otherwise called as the Karhunen-Loeve Transform (KLT). SAAK transform can be implemented in 3 main steps.

1) Projection of subspaces with their orthonormal basis
2) Augmenting the vector with their negative counterparts
3) Applying ReLu activation function to the augmented vector

So, the first step is finding the transformation kernels, otherwise called as ‘anchor vectors’. These can be found using a quad-tree structure. If we consider the 2x2 blocks in the images, then we can easily frame the kernels as it is very easy to project the 2x2 kernels onto their orthonormal subspaces. Thus, the objective is to split the SAAK into multiple steps, giving rise to Multi-stage SAAK transform. For MNIST application, if we consider the size of images to be 32x32, we can first divide the image into four 16x16 sections. Since it is easy to project 2x2 subspaces, we can divide each 16x16 into 2x2 blocks and find the projections and choose the important component. This is something very similar to the pooling concept in CNN. So, we will have 8x8 block for each 16x16 block of the main image, giving rise to 4 transformation kernels. This is the first stage of the SAAK transform for MNIST classification. After the coefficients are extracted, the next step is to augment each kernel with their negative form and apply ReLu to the augmented kernel with the Sign to Position conversion, since ReLu will kill the negative components of the kernel and hence prohibiting the inverse transformation which can occur otherwise. After the kernels are obtained with the above procedure, the image patches can be extracted and then passed onto the next stage for getting the transformation kernels.

In the next step, we have four 16x16 images from the first stage, which can be split into four 8x8 images, for which we can find the important component for each 2x2 block giving rise to 4 transformation kernels for each block, and since there are 4 blocks in the main image, this gives rise to 16 transformation kernels. Here, we can decide how many features we can pass onto the next stage. When this process is repeated, we will get 64 transformation kernels for this stage and for the next stage we will have 256 transformation kernels. For the final stage, we will have 1024 transformation kernels each of a very small dimension. If we consider the total number of kernels, it is very high. When the convolution between the kernels and the image is performed, and when each image is converted to a 1D array, the dimension becomes very high and in most cases, it is difficult to train a classifier with this much number of dimensions, and ultimately training the model needs computation power very similar to the CNNs. Hence, we can decide the number of components we want to pass on for the next stage. This can be performed arbitrarily choosing the number of components (filter kernels) to be preserved for the next stage.

For example, the first stage convolution output will have 16x16 blocks, the second stage will have 8x8 blocks, third stage will have 4x4 blocks, fourth stage will have 2x2 blocks and the last stage will just be 1x1 blocks. Arbitrarily we can decide the number of components as follow:

$$3 * 256 + 4 * 64 + 7 * 16 + 6 * 4 + 8 * 1 = 1168$$

Even after choosing only the best components from each stage, the output dimension is very high. The last step is to apply PCA or other means of dimension reduction techniques to get a reduced dimension dataset. Once the reduced dimension data points are extracted, the next step is to train a classifier with the dataset. Once the trained model is obtained, it can be used for classifying the test images. This is the working principle of the SAAK transformation for classifying MNIST dataset.

<u>Experimental Results:</u>

Training and testing accuracies obtained used the SVM and RF classifiers with different number PCA components namely 32, 64, 128. The number of components were chosen in such a way that the total dimension for one combination was around 1160 and for another combination, it was around 2048. The results obtained after choosing the reduced dimension dataset taking different number of components from each stage, gives the same accuracy for all the three PCA dimensions. Hence the results obtained are common for both 1160 dimension and 2048 dimensions.

The following table was obtained using 60,000 images for the training data

| Classifier | Components after F-Test | Components for PCA | Training Accuracy | Test Accuracy |
|---|---|---|---|---|
| SVM Classifier | 1000 | 32 | 99.22% | 98.33% |
| | 1000 | 64 | 98.85% | 98.16% |
| | 1000 | 128 | 98% | 97.57% |
| Random Forest Classifier (15 trees) | 1000 | 32 | 99.96% | 93.97% |
| | 1000 | 64 | 99.96% | 93.63% |
| | 1000 | 128 | 99.97% | 92.4% |
| Random Forest Classifier (10 trees) | 1000 | 32 | 99.85% | 93.16% |
| | 1000 | 64 | 99.89% | 92.22% |
| | 1000 | 128 | 99.91% | 90.14% |

The following table was obtained using different number of images for training data

| Classifier | Number of images used | Components for PCA | Training Accuracy | Test Accuracy |
|---|---|---|---|---|
| SVM Classifier | 50,000 | 32 | 99.212% | 98.20% |
| | 40,000 | 32 | 99.23% | 98.13% |
| | 30,000 | 32 | 99.20% | 97.86% |
| | 20,000 | 32 | 99.10% | 97.48% |
| | 10,000 | 32 | 99.07% | 96.79% |
| Random Forest Classifier (10 trees) | 50,000 | 32 | 99.21% | 92.81% |
| | 40,000 | 32 | 99.23% | 92.67% |
| | 30,000 | 32 | 99.2% | 92.23% |
| | 20,000 | 32 | 99.1% | 90.77% |
| | 10,000 | 32 | 99.07% | 88.96% |

Discussion:

**Comparison between the CNN architecture and SAAK transform**

There are many differences between the CNN architecture and the SAAK transform architecture. In CNN, the features are extracted based on the kernel weights tuned by an iterative procedure. In SAAK transform, there is a prescribed way to extract the transformation kernels. This is based on the KLT transformation. Also, the complexity for CNN is very high when compared to SAAK.

Comparing CNN and SAAK transformation step by step, the first stage in CNN is to find the filter kernels to extract the features from the image. In CNN, the filter weights have to be tuned in order to extract the specific features from the image. In SAAK transform, the feature space is obtained by performing the convolution between the image and the different kernels obtained with the orthonormal transformation of every 2x2 block in the image. SAAK transform is multi stage, which is very similar to framing a number of convolution layers in CNN. In CNN, the convolution is performed stage by stage, whereas in SAAK, all the kernels are first obtained using the KL transformation and then applied to the image, leading to a 1D vector of dimension specified by the number of components extracted from each stage of the SAAK transformation.

In SAAK transform, a concept similar to the pooling in CNN takes place. For finding the kernel in SAAK, we consider the 2x2 block in each image. We apply an orthogonal transformation to each of the 2x2 block and take the most important component in the 2x2 block and consider it for the next stage. In CNN, we use the concept of Max pooling or average pooling, which is used to down-sample the image. For example, if we pass a 2x2 block to the max pooling layer, it chooses the largest element in the block and conveys it to the next stage. Thus, choosing the important component in a 2x2 block is very similar to the pooling concept in CNN, except for the fact that the projection is considered for the SAAK transform unlike averaging/maximum pooling in CNN.

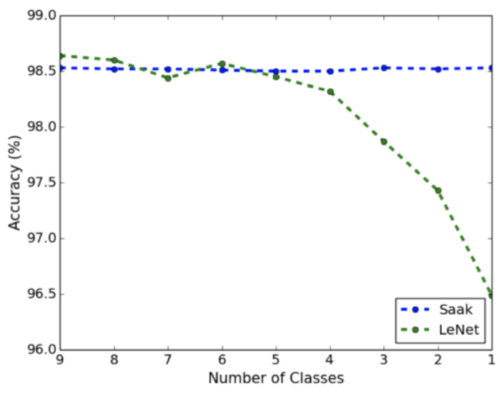
The next step in CNN, after the convolution and the pooling layers is that, CNN requires the conversion from convolution layer to a stretched 1D vector. The number of convolution layers and the pooling layer parameters decide the number of components that will be present in the fully connected layers. In the case of SAAK, we arbitrarily decide the number of components of each data point that we will be using for our classification purposes. This is done in 2 steps. The first one is based on the F-test, in which the features are reduced based on the variance between the features. The second step is to further reduce the input feature dimension with the help of PCA. This is how the feature dimension is reduced in SAAK transform.

The most important difference between SAAK transform and CNN is that, inverse SAAK transform can be used to produce the original images from the final feature vector. In CNN, ReLu activation used kills the components in the negative side, and sigmoid function saturates all the values greater than 1, to 1 and hence the diversity in the input is lost. In case of SAAK, before the end of each stage, the feature vector is augmented with its negative component and passed to the activation function only after that. This is the main reason why inverse SAAK transform exists. The sign to position conversion is used during the forward SAAK transform and the position to sign conversion is used during the inverse transform. Thus, SAAK transform can be used to reproduce the images from the features, enabling this transformation called as feature-to-image. This idea of feature-to-image is used in many concepts such as Image in-painting, etc.

Feature selection in CNNs depend only on the filter kernels used for the convolution layers whereas in SAAK transform it can depend both on the feature reduction step as well. The features in SAAK transform are selected using the F-test which uses two relations present in the dataset namely, Between group variability and Within group variability. Using these 2 relations, the dimension of the feature space is reduced. To further reduce it, PCA is used. These are the two steps used in SAAK transform for feature selection, whereas in CNN, no such things are used. Only the stride in the max pooling layer and the kernel size along with the stride value during the convolution layer can decide the dimension of the feature space produced by the CNNs.

Apart from the architectures used, there are many differences between CNN and SAAK based on their performances. When the efficiency of the SAAK transform is compared with that of CNN, the accuracy for CNN is very high. Even though the accuracy is high, the training time is very high for CNN. Also, the complexity of the system is very high for CNN and low for SAAK. The number of dimensions can be easily manipulated in SAAK transform unlike CNN. The only computationally heavy part in SAAK is to extract the SAAK anchor vectors. Once that is done, only the feature selection and feature reduction stages and training are left.

The next performance parameter to be compared is the scalability. Class labels are very important for CNN when compared to that of SAAK. Without the labels, CNNs will not work. Since the loss function is calculated for every iteration to tune the filter parameters, CNNs cannot exist without the true labels. Whereas in SAAK, the coefficients can be extracted without the need for the true labels. The labels are need only for training of the classifier model. Also, other unsupervised classifiers such as EM or k-means can be implemented with SAAK coefficients, but this idea is totally not possible with CNNs. The number of classes also plays a major role in CNNs. The optimizers work pretty well when there are large number of classes. This is not the case with the SAAK transform, the number of classes do not matter. Even if we implement a classification algorithm for just a subset of digits between 0-9, SAAK transform will give us the same results unlike CNN, which needs all the ten classes for a batter accuracy. SAAK transform will not affected by any change in orientation of the images, unlike CNNs since the coefficients are extracted based on the largest eigenvalue criterion, and the Eigen values will not change for any 2x2 block being present in a different location when compared to the original location. All these facts convey that, SAAK transform method of MNIST classification is a more scalable model when compared to that CNNs. They can be used to any number of classes, and the orientation of the images does not matter.



The figure given above compares the accuracy between SAAK transform and CNN based on the number of classes present in the dataset. We can clearly see that, CNNs require high number of classes to give their best performance.

Another important difference between CNN and SAAK is the robustness. When noisy images are used to train the CNN model, the kernels get affected. But in the case of SAAK transform since we choose the important components based on their Eigen values, the noise factors get discarded. This is another important difference between SAAK and CNN. Though Alexnet outperforms SAAK in the noisy aspects, the computational power required to train Alexnet is very high when compared to that of SAAK. These are the important differences between SAAK and CNN (LeNet-5).

Discussion on classification accuracies:

The accuracies obtained with different PCA components and different number of images in the training stage using both SVM classifier and Random Forest classifier are given in the table above. We can see that, the number of components used in PCA play a very important role in deciding the accuracy of the model. The accuracies for SAAK transform are generally lower when compared to that of CNN. One main advantage with SAAK is that, the training time is very less when compared to the time taken to train a CNN. The accuracy for SAAK with SVM classifier and 32 components for PCA has given a test accuracy of 98.33% which is high when compared to that of the accuracies obtained for PCA components 64 and 128.

We can see from the tables that, Random forest classifier gives high accuracy for the train model, but gives very low accuracy for the test model. When SVM is used, the training and the testing accuracies are in the comparable range unlike the accuracies obtained with Random Forest classifier. When the trees are increased, the test accuracy increases by a small value. Also, when the number of trees become very high, then the training accuracy will go high, and the testing accuracy will decrease very much. Usually, Random Forest classifiers have better accuracy when compared to that of PCA. This is true only when the number of classes are really high. When the number of classes are low, SVM tends to have a better accuracy when compared to that of Random Forest Classifier.

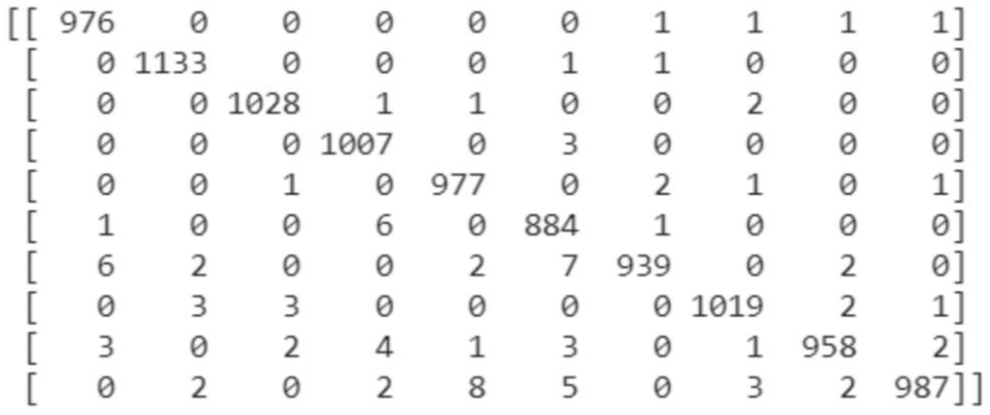
The highest accuracy for both the classifiers is obtained using number of components in PCA to be 32. As the number of components are increased, the accuracy reduces and the lowest accuracy is for the dimension unreduced dataset. Astonishingly, the accuracy reduces for any lower number of dimension and is specifically highest only for 32 features. When the number of features are reduced further, the model starts to over-fit the data. Before applying feature-reduction using PCA, feature selection has to be performed. This can be done using the F-test measure. This selects the features having the highest discriminant power. The number of dimensions being 1168 in this case, choosing 1000 best features, or 800 best features does not give any difference in the accuracies. Hence, applying PCA directly without feature selection, also gives very similar results.

Another important difference with SAAK and CNN is that, the number of training images do not affect the accuracy to a very great extent. If we see the testing accuracy for SAAK with 30,000 images the accuracy is not very less when compared to that of the testing accuracy obtained after training the model with 60,000 images. This is the main difference between SAAK and CNN. When the number of training images are lesser for CNN, the accuracy will be very low because a CNN cannot be trained with lesser number of images.

Error Analysis:

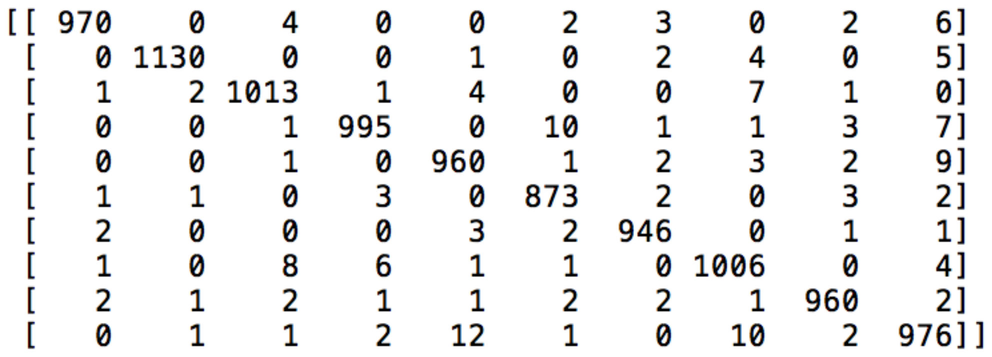
Confusion Matrices obtained using CNN and SVM with PCA components 32 are given below:

Confusion matrix for best performing CNN



```
[[ 976    0    0    0    0    0    1    1    1    1]
 [   0 1133    0    0    0    1    1    0    0    0]
 [   0    0 1028    1    1    0    0    2    0    0]
 [   0    0    0 1007    0    3    0    0    0    0]
 [   0    0    1    0  977    0    2    1    0    1]
 [   1    0    0    6    0  884    1    0    0    0]
 [   6    2    0    0    2    7  939    0    2    0]
 [   0    3    3    0    0    0    0 1019    2    1]
 [   3    0    2    4    1    3    0    1  958    2]
 [   0    2    0    2    8    5    0    3    2  987]]
```

Confusion matrix for SAAK transform (PCA: 32 components)



```
[[ 970    0    4    0    0    2    3    0    2    6]
 [   0 1130    0    0    1    0    2    4    0    5]
 [   1    2 1013    1    4    0    0    7    1    0]
 [   0    0    1  995    0   10    1    1    3    7]
 [   0    0    1    0  960    1    2    3    2    9]
 [   1    1    0    3    0  873    2    0    3    2]
 [   2    0    0    0    3    2  946    0    1    1]
 [   1    0    8    6    1    1    0 1006    0    4]
 [   2    1    2    1    1    2    2    1  960    2]
 [   0    1    1    2   12    1    0   10    2  976]]
```

We can see from the confusion matrices that there are more misclassifications for SAAK when compared to CNN. Comparing the accuracy scores of SAAK and CNN, we can see that the best performance for CNN is 99.54% and the best performance for SAAK is 98.83%. We can clearly see that CNN has got a better classification accuracy when compared to that of SAAK.

As we had mentioned above, number of training images need for SAAK transform classification do not make that much of a difference in the classification accuracy. For CNN, the number of images need to be 60,000 otherwise the accuracy reduces by a wide margin. For SAAK, that is not the case. Even if the number of training images are less, the classification accuracy does not reduce by much. This is one of the main differences between SAAK and CNN.

CNN models require more time to train unlike the SAAK model. Also, SAAK models are somewhat generic when compared with CNN. SAAK requires the same procedure to be performed all the time, for any application. For CNN, it is very application specific and the CNN model needs to be trained specifically for every application. The number of classes also need to be higher for CNN as it is the basis on which the model is trained. In SAAK, the number of classes do not make that much of a difference. The accuracy is always the same for SAAK transform, no matter what the number of classes is. This is another main advantage of SAAK. If we use SAAK, a image classification problem behaves just like a data classification problem. Thus, SAAK is more robust.

As we had discussed above, SAAK has 3 main advantages when compared to CNN. The efficiency is higher, the model is scalable and the model is more robust. The efficiency is high even though the accuracy is low because, the time needed to train the model is low for SAAK. The model is scalable as the number of output classes in deciding the performance of SAAK and the SAAK model is more robust because it is not very sensitive to noise. These are the main differences between SAAK and CNN.

The SAAK transform can provide ground breaking results after making changes to algorithm. Since the SAAK transform is just discovered, time has to be spent on it to improve the accuracy. Another main advantage with SAAK transform is that, the inverse SAAK transform exists. So, from the last layer, we are able to rebuild the original image. We can use the same concept for CNNs as well. By using this concept, we can generate the images from the last layer. The CNNs can help getting inverse transformations as well. This idea can be used in generative deep nets to generate images. The generated images can be used for many other applications such as in-painting, image restoration, image enhancing etc.

The SAAK transform is in the inception stages and there are lot of changes that can be implemented to improve their performances. We can find a solid foundation for choosing the number of components from each stage in SAAK transform. Now we are arbitrarily finding the number of components in each stage. If we find the correct method to properly choose the number of components, then the accuracy of SAAK transform may increase.

We can choose different methods for dimension reduction instead of F-test and PCA to choose the different reduced components. Different methods of feature reduction may improve the classification accuracy. SAAK transformation does not depend on the dimension of the feature vector. It can work for any application like communication etc., until the feature vectors can be decomposed into lower dimensional vectors. Since SAAK transform can restore the images, we can implement them for generative deep networks to restore images, perform in-paining etc., Thus, this area of SAAK transform has not been ventured completely, and there is a lot of scope for improvement in SAAK transform. If properly studied, SAAK transform can perform better than CNNs.

REFERENCES:

- https://sukhbinder.wordpress.com/2014/09/11/karhunen-loeve-transform-in-python/
- https://mxnet.incubator.apache.org/tutorials/python/mnist.html
- https://keras.io
- https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/
- http://cs231n.github.io/convolutional-networks/
- https://arxiv.org/pdf/1710.10714.pdf
- https://arxiv.org/pdf/1710.04176.pdf
- https://github.com/morningsyj/Saak-tensorflow