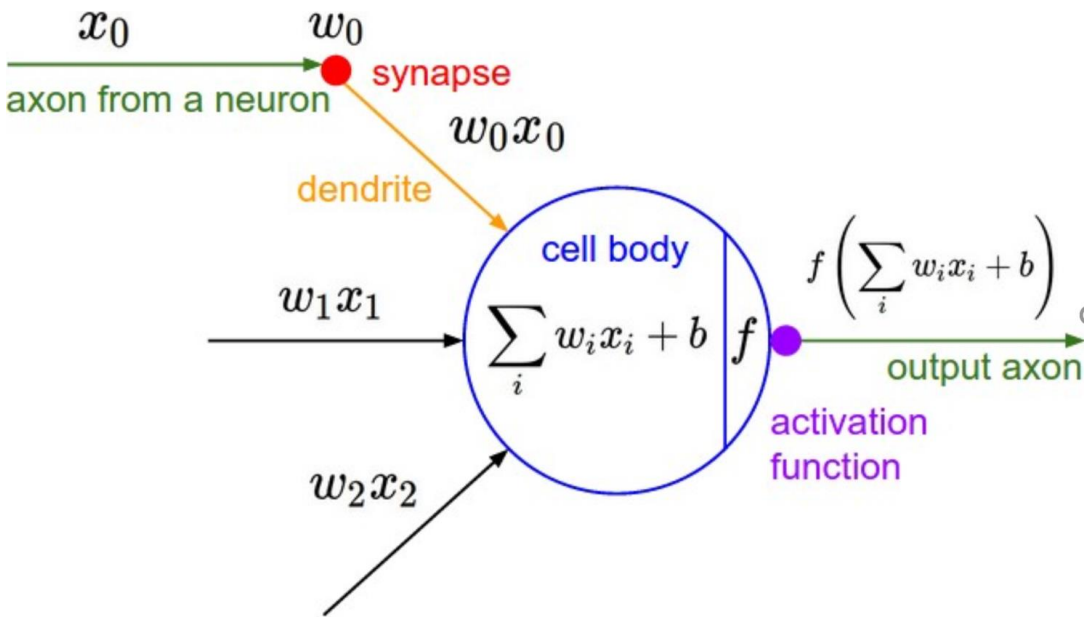
EE569: Homework #4
Report

ISHAN MOHANTY
USC ID: 4461-3447-18
Email: imohanty@usc.edu



1. Training a Convolutional Neural Network on MNIST dataset and Study its properties.

Part(a)

I. Abstract and Motivation

The Inception of Convolutional neural networks or CNNs for short began during the early years of 1990s. The CNNs are a subset of Neural networks that have been immensely successful in object recognition and classification and are recently used in powerful applications like self-driving cars. CNNs use the concept of multilayer perceptrons designed to perform minimal pre-processing. The CNNs learn and design their own filters which were earlier hard-coded by people in the traditional image processing domain.

CNNs took inspiration from biological phenomenon by emulating the connected pattern of neurons present in the innate animal visual cortex. The first of the CNNs to emerge and create a breakthrough was the Lenet-5 CNN architecture. Now advanced CNN architectures have been designed to obtain very high classification accuracies, examples of such architectures are GoogleNet, VGGNet, DenseNet etc. In this section we are motivated to understand the architecture of the CNN and the functionality of each component.

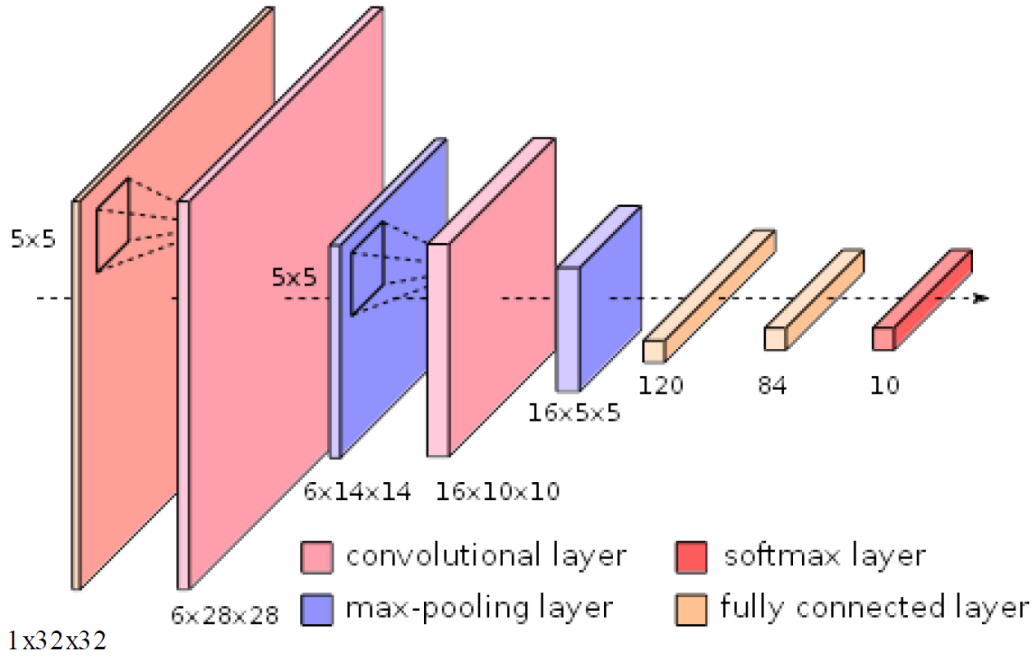


Figure 1: CNN architecture derived from LeNet-5

I. Discussion

Description of the following layers and their functionalities are given below:

1. Convolutional Layer:

CNNs get their primary name by using the "convolution" operator. The CNNs perform the convolution action on the input matrix data and submit the result to the next layer. The CNN designs most important layer is the convolutional layer. It contains filters that are deduced by learning algorithms present in the layer, these filters are called kernels. During forward pass, the automatically calculated filters are convolved with the input image matrix. The result of this convolution process produces a 2-dimensional activation map for that filter. The network during this process learns filters that activate a particular-type of feature at some spatial location of the input data.

Output of the convolutional layer is a series of activation maps stacked one below the other along the depth of the layer. The output of this layer can be interpreted as the feedback response of a neuron that observes a patch of a certain dimension of the input data and this response is passed on to the neighbouring neurons belonging to the same activation map. Three parameters decide the dimension of the activation map, they are given below:

1. **Depth:** This represents the number of kernels used for the convolution action.
2. **Stride:** It gives us information on the number of pixels by which the kernel is slided over the input data matrix. For Instance, when stride is 2, the kernel is moved two pixels at a time.
3. **Zero-padding:** It is the process of adding zeroes at the boundary of the input matrix so that the convolution process is made more simple and convenient.

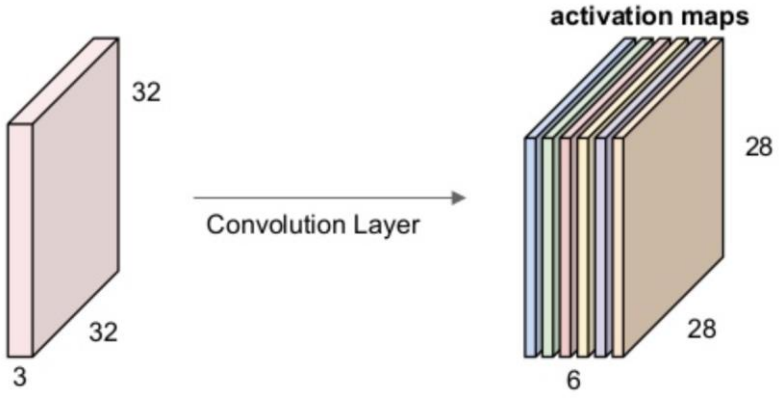


Figure 2: Function of Convolutional Layer

2. Max Spatial Pooling Layer

Pooling is also referred to as down-sampling. It's fundamental purpose is to diminish the Feature Activation Map's dimensions while retaining vital information. There are variety of pooling techniques like average, sum, max etc. More commonly, max-pooling is used while performing sub-sampling as it works the best in CNNs. It divides the input matrix into disjoint rectangles and outputs the maximum element among the elements present in each disjoint rectangle. Here, the location of the feature relative to the other features is estimated crudely and hence, the precise location of it is less significant.

The pooling layers functionality is to reduce the size of the kernels and therefore reduces the computational effort in the network. This controls overfitting and hence, It has become a rule of thumb in the CNN architecture to attach a pooling layer in between consecutive convolutional layers. The pooling operation helps achieve an almost scale invariant representation of the input image. Therefore, this property afore-mentioned is highly useful in object detection in the image without prior knowledge of its location.

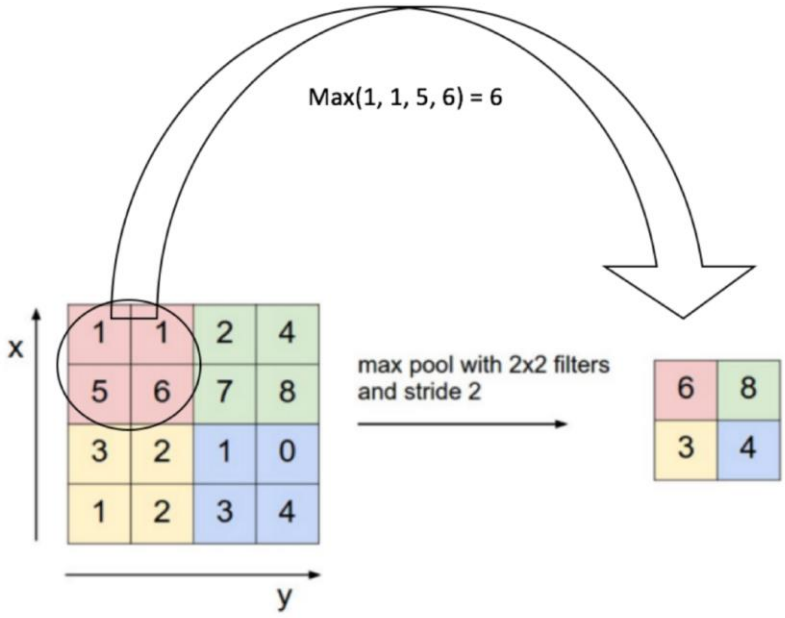


Figure 3: Illustration of the Max pooling Layer

3. Fully Connected Layer

The Fully-Connected Layer helps in logical deduction in the neural network. It has the same underlying principle as that of the traditional multi-layer perceptron. The fully-connected layers functionality is to connect neurons from previous layer to neurons residing in the subsequent layer. Their activations are calculated by performing matrix multiplication followed offsetting the bias. The yield from the numerous convolutional and pooling layers are represent the feature information of the input data. This Layer classifies the input image data into its respective classes based on the feature information.

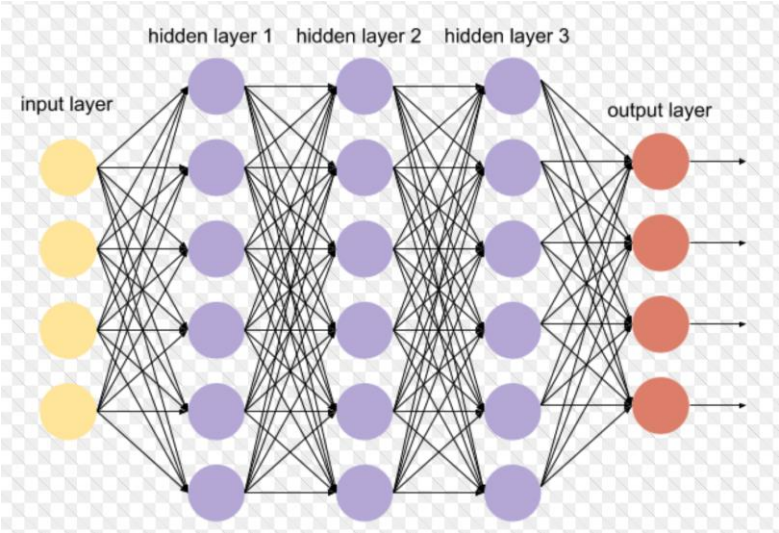


Figure 4: Connections in the Fully Connected Layer

4. Activation Function

An Activation Function is a node that you add to the output end of a neural network. It is also called as **Transfer Function**. It is utilized to determine the output of neural networks. It maps the resulting values from 0 to 1 or from -1 to 1 depending on the type of activation function. Activation functions can be of two types namely, Linear and Non-linear activation functions.

(a) Linear or Identity Activation Function

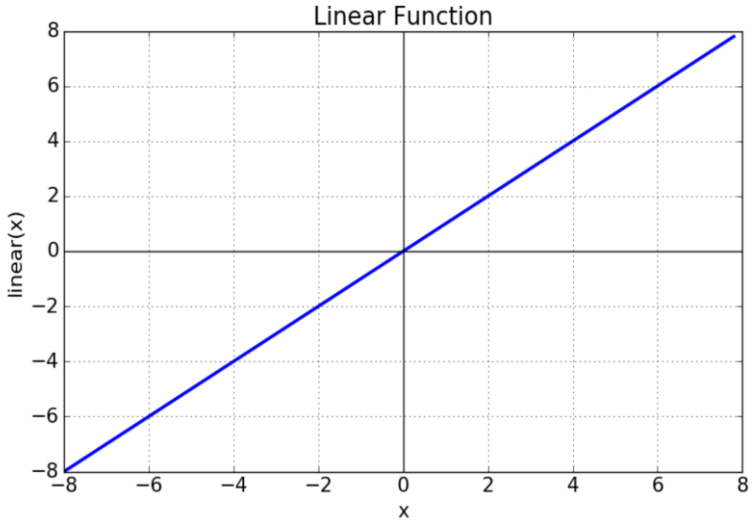


Figure 5: Linear Activation function graph

This function is denoted by f(x) = x for all x. Linear Activation function does not help with the complexity of various parameters of data that is fed to the neural networks.

(b) Non-linear Activation Function

The Nonlinear Activation Functions are the most frequently used activation functions. Non-linear functions adapt well to different data and differentiate between output.

Different Non-Linear Functions are given below:

1. Sigmoid Activation Function

The main objective behind the use of the sigmoid function is its desirable range which is between 0 and 1. It is hence used in systems that need to figure out probability as an output. The property of probability is that it's output ranges between 0 and 1 and therefore we exploit this property to make sigmoid function a desirable choice. The function is differentiable and hence, by this property we can find the slope of the sigmoid curve at any two points. The function is also monotonic**.**

2. Tanh Activation Function

The Tanh function is also like sigmoid. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). This functions primary strength, is that there is strong sign-based mapping which means positive inputs are mapped to positive outputs and negative inputs are mapped to negative outputs. The function is differentiable and monotonic. The tanh function is mainly used for classification between two classes.

3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is very commonly used in almost all the convolutional neural networks or deep learning. The ReLU is half rectified from the bottom. f(x) is zero when x is less than zero and f(x) is equal to x when x is above or equal to zero. The range of the function is from zero to infinity.The function and its derivative both are monotonic. Problems associated with this function is that negative-values end up becoming zero and this reduces information for training the model.

4. Leaky ReLU

The leaky ReLu is an attempt to solve the shortcomings of the ReLu Activation Functions. The leak in the ReLu function helps to increase the range of the ReLU function. Therefore, the range of the Leaky ReLU is from -infinity to infinity. Usually, the value of constant multiplied with x is 0.1 or so.

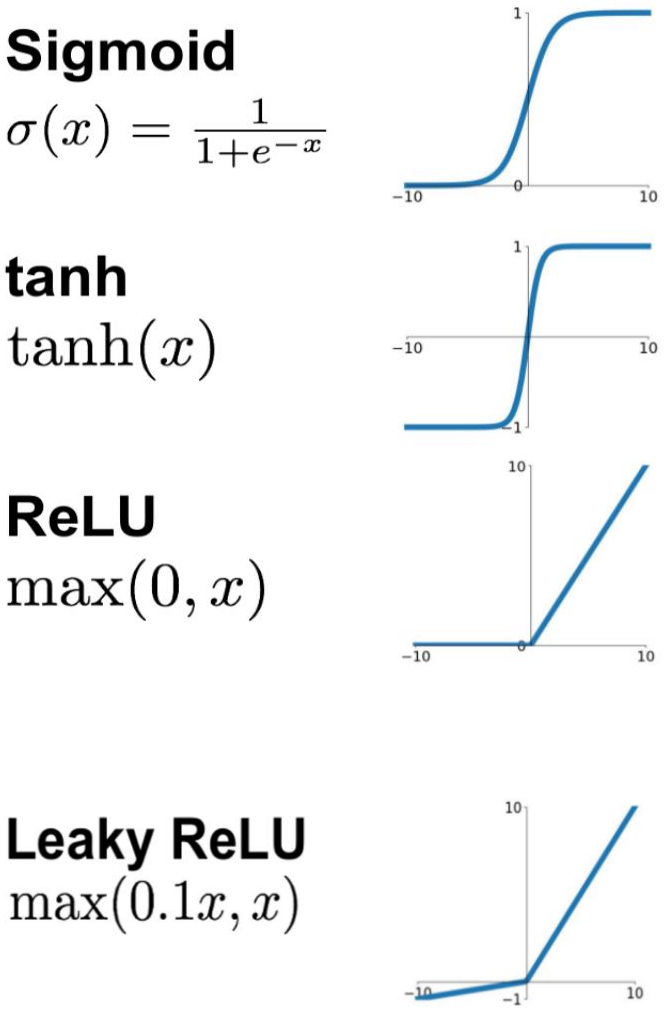


Figure 6 : Different Types of Non-linear Activation Functions

5. SoftMax function

The SoftMax function restrains the outputs of each unit in the range of 0 and 1, similar to the sigmoid function. It is a non-linear activation function. It normalizes each output unit so that the total sum of the outputs is equal to 1. We can observe this in the figure below,

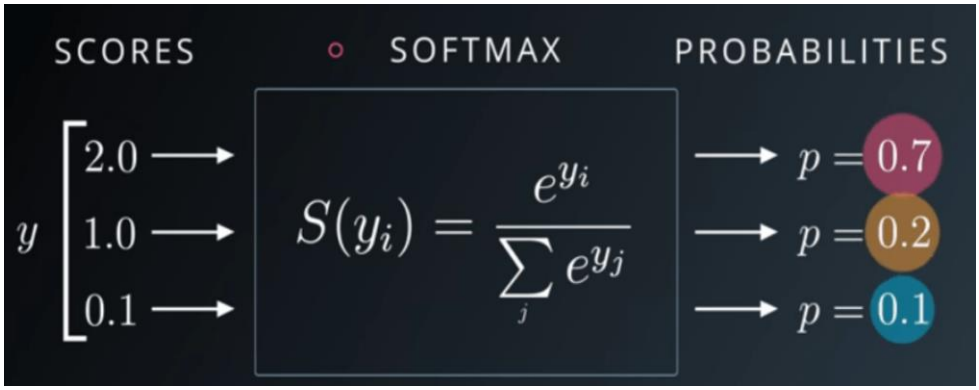


Figure 7: SoftMax Functional Property

Mathematical formula for the SoftMax function is given below, here z is an input vector to the output layer.

$$\sigma : \mathbb{R}^K \to (0,1)^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

The softmax activation is usually applied to the layer in a neural network, instead of using ReLU, sigmoid, tanh, or another activation function. The reason why softmax is extremely useful is because it converts the output of the last layer in the neural network into a probability distribution.

Over-fitting complications in CNN and steps taken to prevent it during training of the network.

Overfitting is nothing but forcing the training set to learn by-heart to such a huge-extent that its generalisation property becomes very limited. Hence, our model after training may have learnt it well, however it eventually neglected to catch the basic procedure that created it and hence, cannot generalize well.

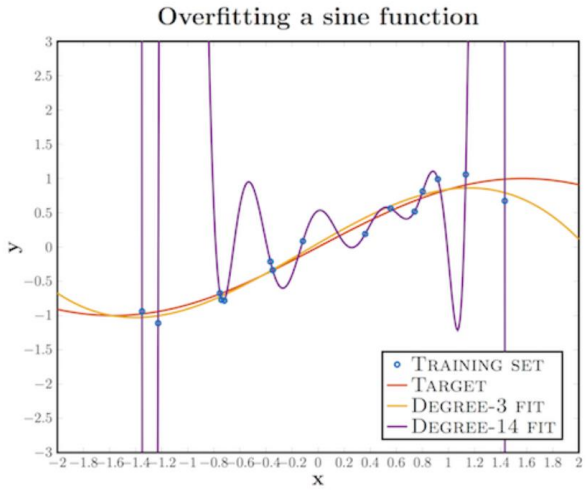


Figure 8: Sine-Function Overfitting Scenario

Convolutional neural systems have vast number of parameters, particularly in the Fully connected layers. Overfitting may happen in the accompanying cases: in the event that we don't have adequate training instances, a little gathering of neurons may wind up in charge of doing the majority of processing work while other neurons end up in a redundant or erroneous condition. This results in performance reduction.

For our models to generalise better in the cases of over-fitting, we introduce techniques such as regularisation. Rather than decreasing the quantity of parameters, we force limitations on the model parameters to prevent them from learning noisy aspects during training. We introduce the dropout technique which helps to eliminate the loopholes mentioned above. Dropout with a certain predefined elimination probability will recursively remove neurons from the network through that particular epoch. This activity has the impact of convincing the neural system to adapt to faults, and not to altogether depend on presence of a specific neuron and rather to depend on the cumulative decision of the neurons inside the layer. This technique works well within the layer removing the possibility of over-fitting and reduces the number of regularizers.

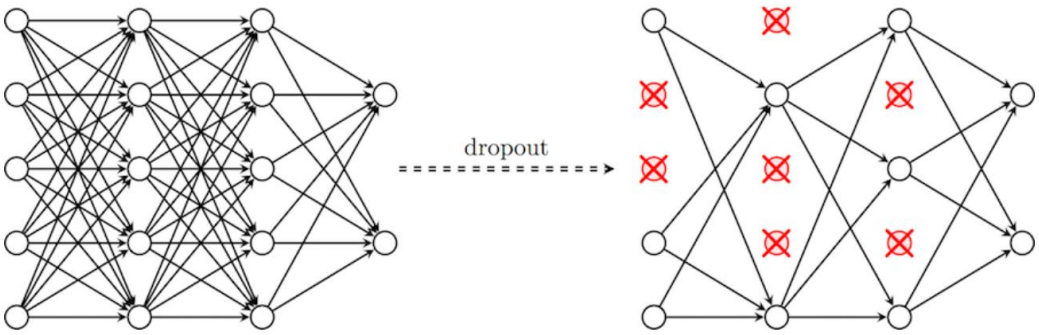


Figure 9: Regularization Technique using Drop-out

CNNs Superior performance in computer vision problems when compared to traditional methods.

The problems associated with computer vision include image classification and object detection. These problems are tackled by using manually selected features from algorithms like SIFT and SURF. The bag of words algorithm worked very efficiently to create a vocabulary dictionary used to classify images based on visual histograms. After the advent of these algorithms, Support Vector Machines or SVMs were introduced to perform classification of images and were very efficient in doing so. All these algorithms which are now traditional, relied heavily on these hand-crafted features.

CNNs However, have a natural preferred standpoint over traditional strategies that is it can consequently learn various feature highlights i.e what features are valuable and how to figure them. From the Traditional standpoint, image classification is done by extracting relevant and important features manually and using it to decide the final result whereas in the case of CNNs, it will automatically learn all the important features by training over many epochs and perform image classification.

Some other advantages of CNN apart from the ones mentioned above are invariance to shifts and distortion in the image, Fewer memory requirements and Easier and better training due to drastic reduction of the number of parameters.

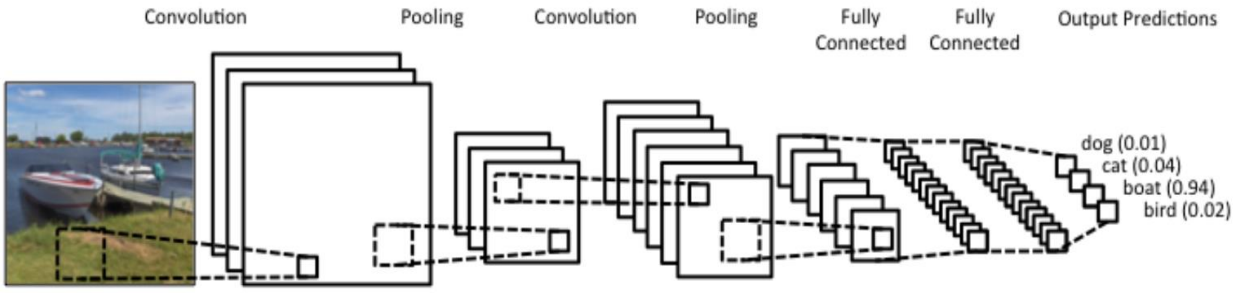


Figure10 : Image Classification using CNNs

Loss Function

Neurons are described by a weight vector (w) represented through kernels. The main objective during training is to learn and assign weights by minimizing an error or loss function, E(w). An example of a learning algorithm commonly used in neural networks is the gradient descent algorithm. The gradient descent algorithm iteratively optimizes the loss function and changes or updates the weight vector of the neurons in the direction of steepest descent. It has a learning parameter symbolically represented by Ŋ which influences rate of neural network convergence. The gradient descent mathematical equation is given below:

$$\vec{w} \leftarrow \vec{w} - \eta \frac{\partial E(\vec{w})}{\partial \vec{w}}$$

The loss function consequently represents as to how flawed the neuron is with estimations utilizing its present parameter value and henceforth, helps direct the training procedure of a neural system. Mean squared error and cross entropy loss are used more commonly.

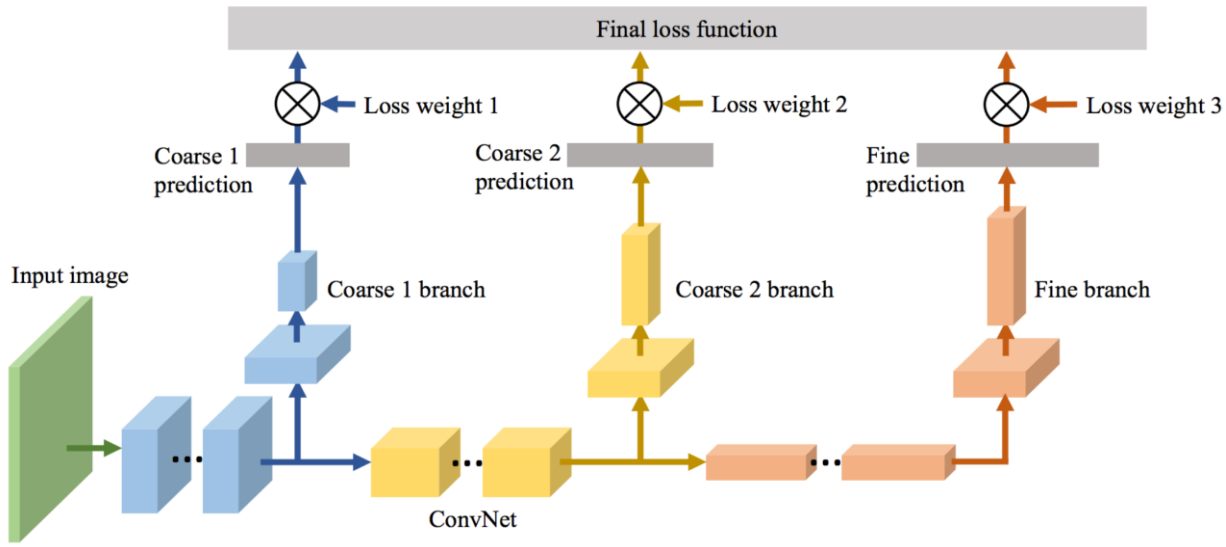


Figure 11: Loss Function Illustration

Classical backpropagation (BP)

The weights of the output neurons can be calculated and updated based on the optimization of the loss function previously mentioned using gradient descent. In the case of other neurons, the losses computed are propagated backwards. This is the main concept of the classical backpropagation.

Backpropagation calculates gradient of the loss function and uses this to change the kernel or filter weights which influence convergence in the neural network. It is also commonly referred to as the backward propagation of errors as the error is calculated through gradient descent and pushed back through the various network layers.

It is considered to be a supervised learning procedure as it requires a known, desired output for each input value. The motivation for backpropagation is to train a neural network in such a way that it can understand and deduce optimal internal weights and expand its knowledge to learn any unknown parameters which are mapped from input to output.

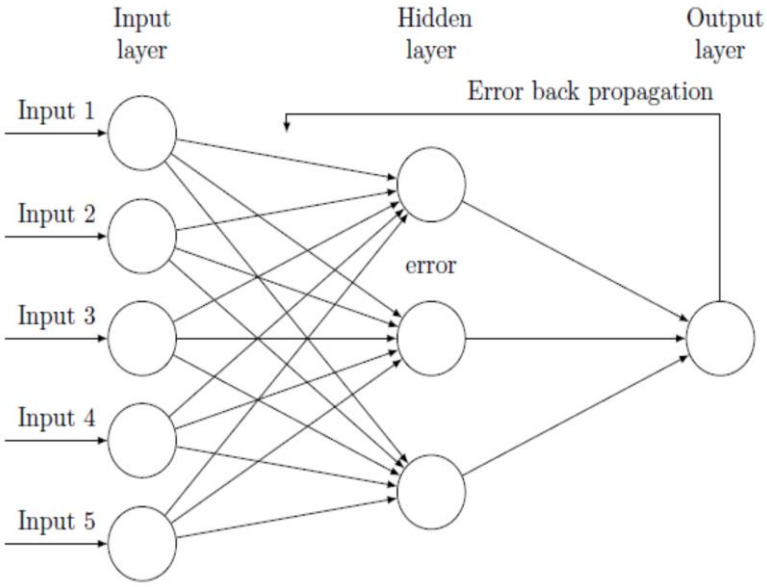


Figure 12: BackPropagation Mechanism

Part (b)
Part (c)

I Abstract and Motivation

LeNet-5 was one of the earliest CNNs to be designed. Its inventor is ***LeCun et al***. It found its application in Multi-national banks where in its primary goal was to recognize hand-written numbers on digitized cheques with 32x32 images. At this juncture we are highly motivated to understand how to train the LeNet-5 on the MNIST dataset and also try to improve its performance by tuning hyper-parameters and altering the LeNet-5 architecture to obtain the best test accuracy.

Figure 13: MNIST Dataset

II. Approach and Procedures

Part(a)

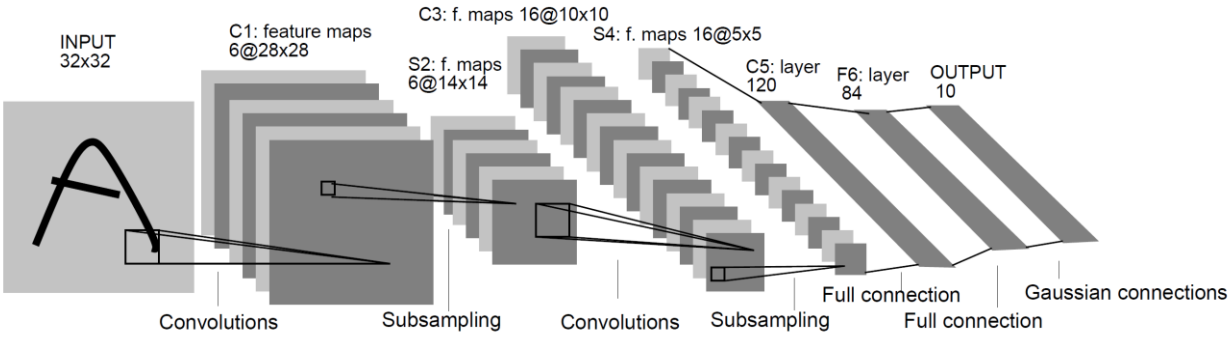


Figure 14: LeNet-5 Architecture

The Lenet-5 architecture comprises of two spatial-convolution and two sub-sampling layers as described in the paper by LeCun with two fully connected hidden layers. The following modules are placed in LeNet – 5 in the respective order:

1. **Input Layer**: The measurements of the information picture is 32x32 for the MNIST dataset. The input layer accordingly comprises of 32x32 pixels spatially organized..

2. **Convolutional Layer C1:** This layer takes in the 32×32 picture from the input layer. The layer has 6 pieces of size 5×5 which produce 6 feature maps of size 28×28 subsequent to going through a sigmoid function. The stride of 1 is utilized amid convolution. C1 contains 156 training parameters and 122,304 associations. This layer has a topological structure, sparse associations and has shared weights.

3. **Sub-Sampling layer S2**: This layer decreases the span of the element maps to half by utilizing a portion of size 2×2 with each of the ones in it. In this manner, we get 6 include maps of size 14x14. Layer S2 has 12 trainable parameters and 5,880 associations. There are no weights present in this layer. At this stage, the sigmoidal function is connected here and this presents non-linearity in the system for better training execution. The definition for sigmoidal function is given above.

4. **Convolutional Layer C3:** This layer has 16 include maps every one of size 10x10. Every unit in C3 is associated with a few 5x5 responsive fields in S2. This layer has 1516 trainable parameters and 151600 associations.

5. **Sub-Sampling layer S4**: like the past S2 sub-sampling layer, S4 layer lessens the measure of the element maps considerably, consequently, now there are 16 feature maps of size 5x5. Every unit S4 is associated with the corresponding 2x2 responsive field at C3. There are 32 trainable parameters and 2000 associations. We apply the sigmoidal function here also just like the sub-sampling layer S2.

6. **layer C5:** This layer has 120 feature maps of size 1x1. Every unit in this layer is associated with the 5x5 responsive fields of S4. This is the fully connected layer however can likewise be called as the convolution layer if the input data has a greater dimension. On the off chance that the image has a bigger dimension then the CNN dynamically alters the layers and C5 turns into a convolutional layer. There are 48120 trainable parameters and associations.

7. **Fully Connecter Layer F6**: This layer has 84 fully connected units. It is connected by 120 units from the previous layer C5. There are 10,164 trainable parameters and connections.

8. **Output Layer**: it comprises of 10 fully connected units which depict the classes. It is associated from 84 units in the past layer F6. Here, we utilize the SoftMax function to decide probabilities for each class. This layer is made up of of Radial Basis Function(**RBF**) units. There are 840 associations in total as there are 84 units in each class. These RBFs have output that correspond to the negative-log likelihood of a gaussian distribution. After this stage, the weights are learned from back-propagation The outputs of each RBF unit yi is calculated as given below:

$$y_i = \sum_j (x_j - w_{ij})^2.$$

x is the input vector and w is the weight vector.

Initialization of Network Parameters:

For training and testing the MNIST data, apart from the steps above lot of other parameters were also changed and tinkered with to produce the desired output. They are discussed below:

1. **Pre-processing**: The input image was normalized in the range of 0 to 1. The data was zero-centered as well by subtracting the mean of each image from each pixel in itself to prevent any high frequencies to dominate the image data.

2. **Weight Initializer**: The network uses the Xavier or Glorot initialization. The main idea behind this initialization is to make it easier for a signal to pass through the layer during forward as well as backward propagation, for the sigmoidal activation function. It draws weights from a probability distribution (uniform or normal) with variance equal to:

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

where n_{in} and n_{out} are the numbers of neurons in the previous and next layer respectively.

3. **Epochs**: represents the number of times the training algorithm will iterate over the entire training set before terminating. In our setup, we have tried different epoch sizes like 20, 200 & 400.

4. **Batch size**: This represents the number of training examples being used simultaneously during a single iteration of the gradient descent algorithm. The batch size is used when fitting the model, and it determines how many predictions must be made at a time. The batch size impacts the CNN training both in terms of time to converge and amount of overfitting.

5. **Learning Rate**: The time required for the network to converge is deduced by the learning rate. Small-scale learning rates converge slowly whereas larger learning rates may not converge faster than the optimum. We report below the behaviour of the network with respect to different learning rates as shown in the experimental results.

6. **Optimizer:** The Stochastic Gradient Decent or SGD was used in the CNN design.

7. **Data Shuffling**: Shuffling of the dataset was executed and trained in batches to prevent the ill-effects of over-fitting.

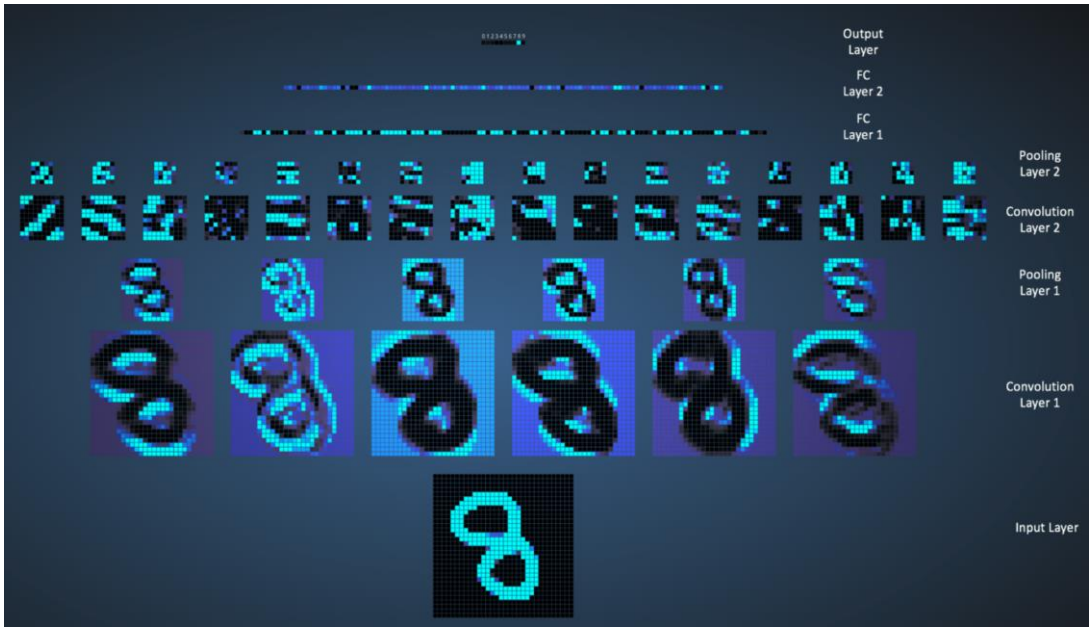


Figure 15: Visual Depiction of the whole CNN process

III. Experimental Results:

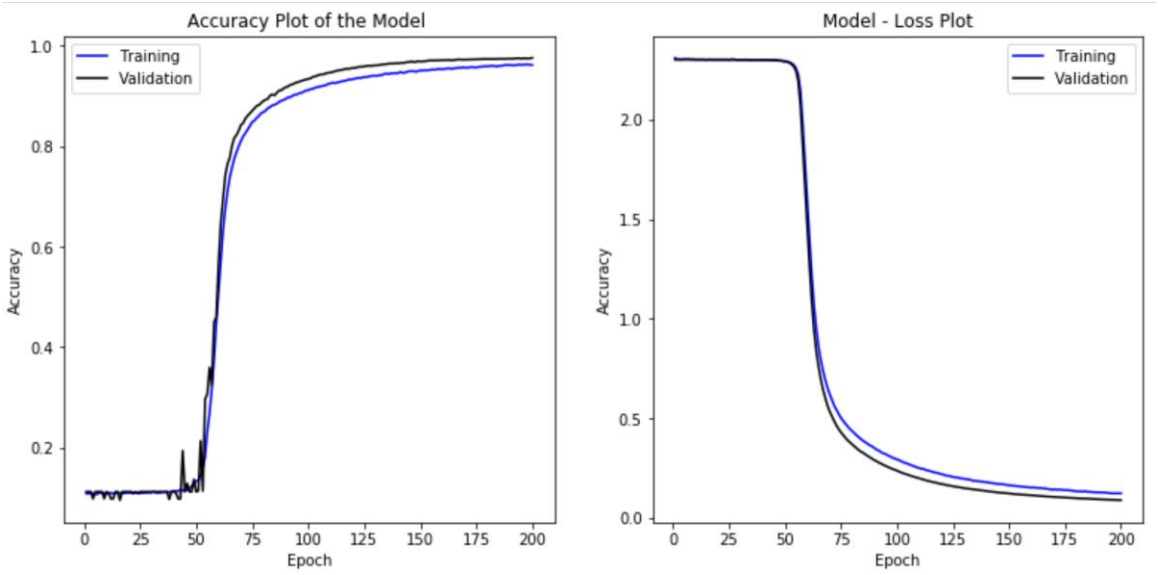
Kernel initializer	Epochs	Learning rate	Batch Size	Momentum	Train Accuracy	Test Accuracy
Random Normal	200	0.05	512	0.5	99.38	97.52
Random Normal	200	0.1	128	0.5	99.45	99.006
Random Normal	200	0.1	20	0.8	99.89	99.15
Random Normal	400	0.2	128	0.5	99.92	99.19
Default (glorot_uniform)	200	0.1	128	0.5	99.68	99.05
Random uniform	200	0.1	128	0.5	99.77	99.08
Variance Scaling	200	0.1	128	0.5	99.73	99.07

Table 16: Classification Results for Different Hyper-Parameter Settings

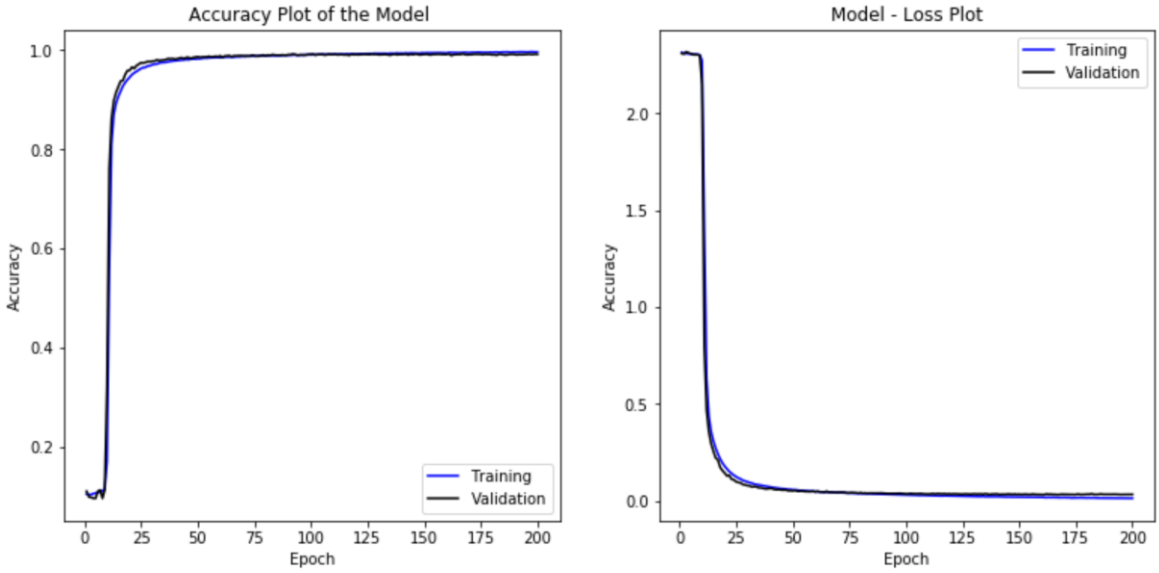
Plots for the above results for accuracy and loss

Parameters:

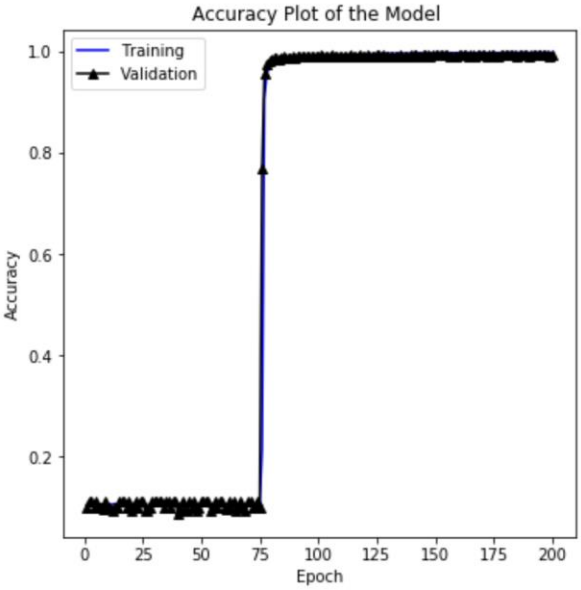
1. Kernel Initializer: Random Normal, Learning rate: 0.05, Momentum: 0.5, Batch size: 512, Epochs: 200, Train Accuracy: 99.38, Test Accuracy: 97.52

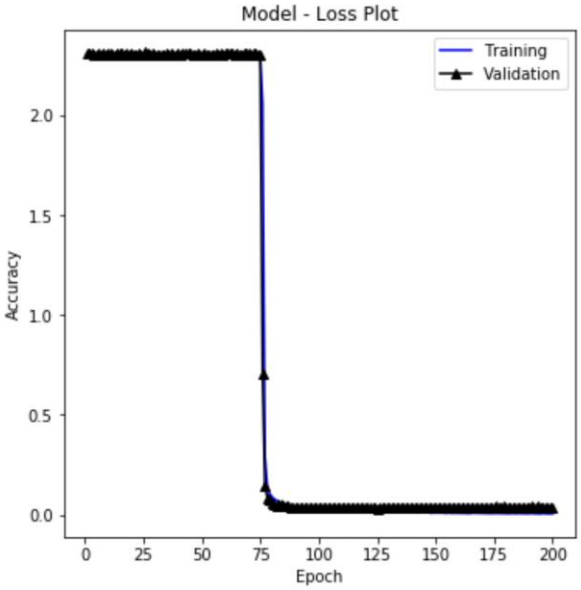


2. Kernel Initializer: Random Normal, Learning rate: 0.1, Momentum: 0.5, Batch size: 128, Epochs: 200, Train Accuracy: 99.45, Test Accuracy: 99.006

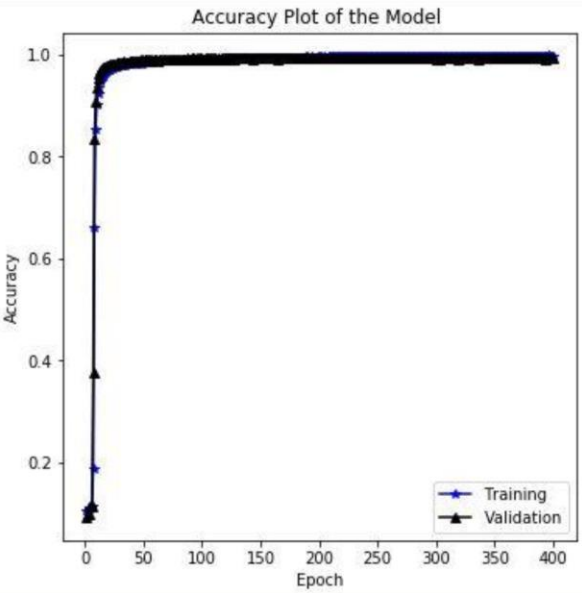


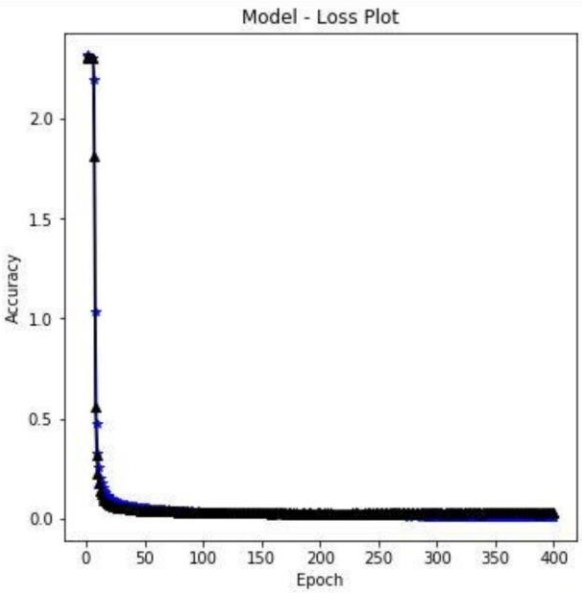
3. Kernel Initializer: Random Normal, Learning rate: 0.1, Momentum: 0.8, Batch size: 20, Epochs: 200, Train Accuracy: 99.89, Test Accuracy: 99.15



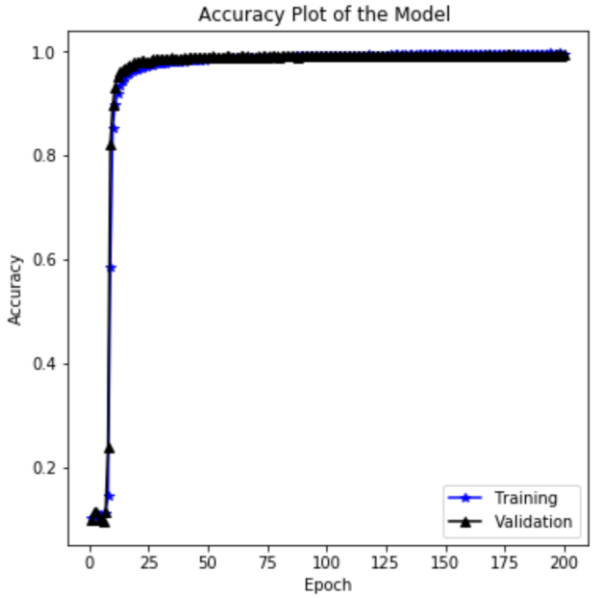


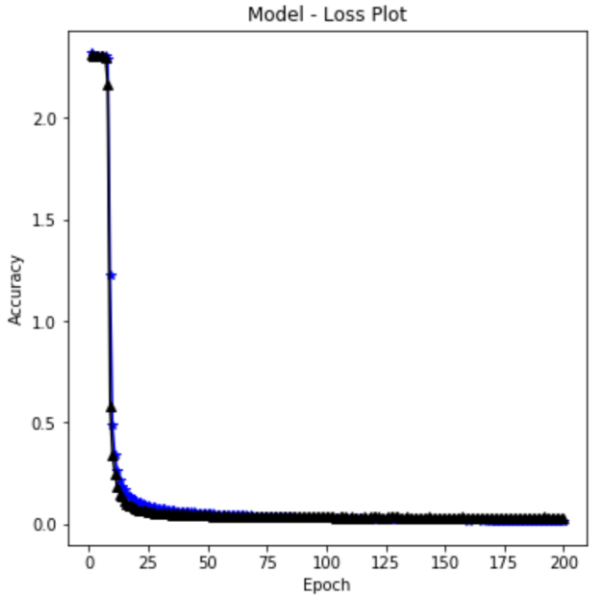
4. Kernel Initializer: Random Normal, Learning rate: 0.2, Momentum: 0.5, Batch size: 128, Epochs: 400, Train Accuracy: 99.92, Test Accuracy: 99.19



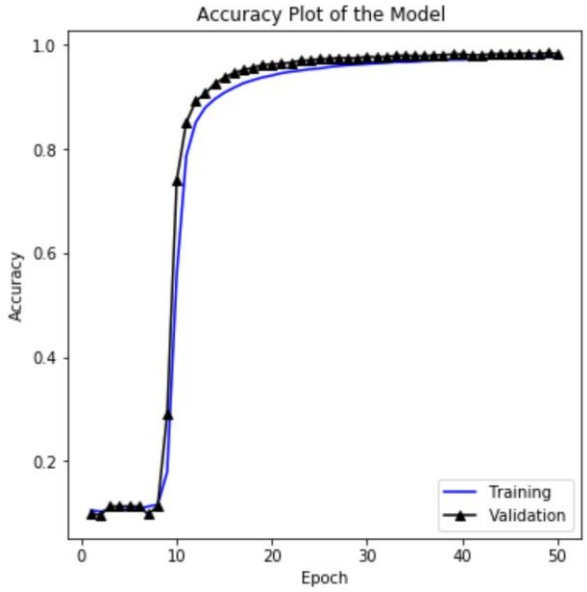


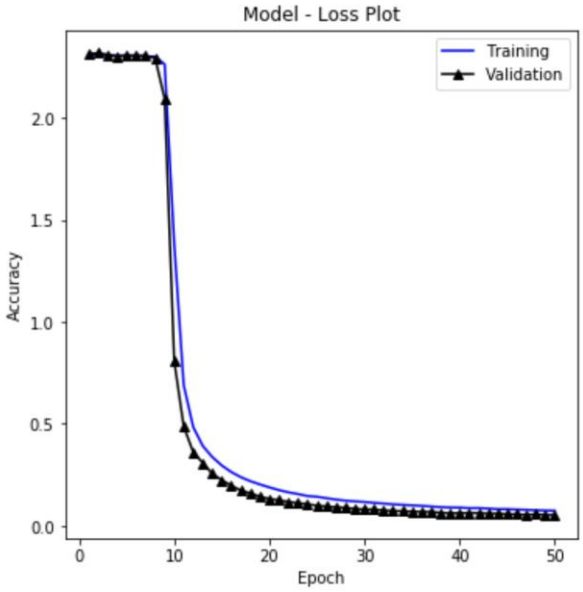
5. Kernel Initializer: Default, Learning rate: 0.1, Momentum: 0.5, Batch size: 128, Epochs: 200, Train Accuracy: 99.68, Test Accuracy: 99.05





6. Kernel Initializer: Random Uniform, Learning rate: 0.1, Momentum: 0.5, Batch size: 128, Epochs: 200, Train Accuracy: 99.77, Test Accuracy: 99.08





7. Kernel Initializer: Variance Scaling, Learning rate: 0.1, Momentum: 0.5, Batch size: 128, Epochs: 200, Train Accuracy: 99.73, Test Accuracy: 99.07

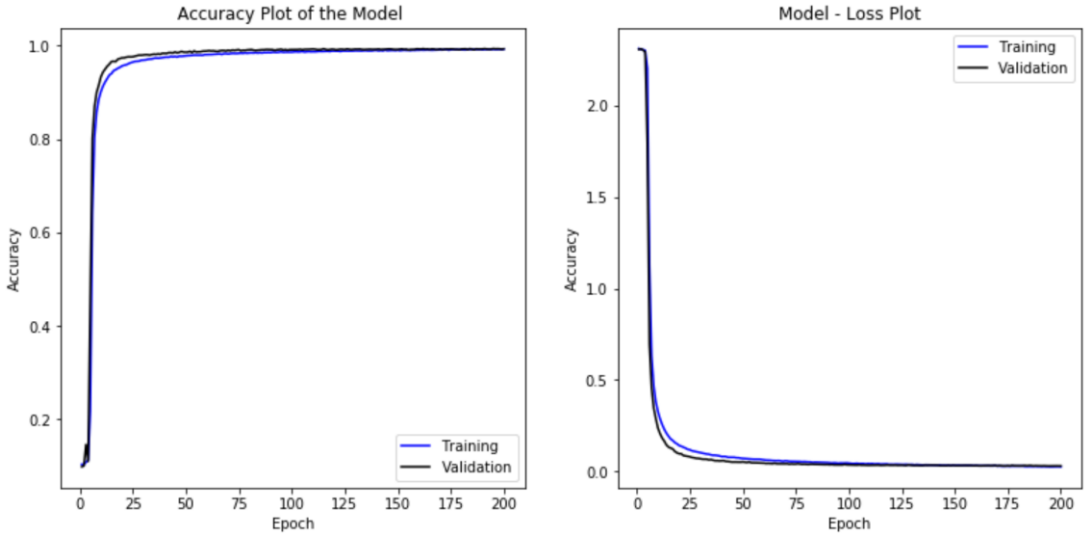


Figure 17: All Accuracy and Loss plots for different Models due to different hyper-parameter setting

Best Case Result for the improvement in performance

Kernel Initializer: Random Normal, Learning rate: 0.1, Momentum: 0.5, Batch size: 128, Epochs: 200, activation function: ReLu, Number of filters in Convolutional Layer C1: 32, Number of filters in Convolutional Layer C3: 64, Spatial pooling: Max-Pooling, Dropout: 0.5, 0.25 after each Max-Pooling stage, Optimizer: Adam.

Train Accuracy: 99.95, Test Accuracy: 99.31

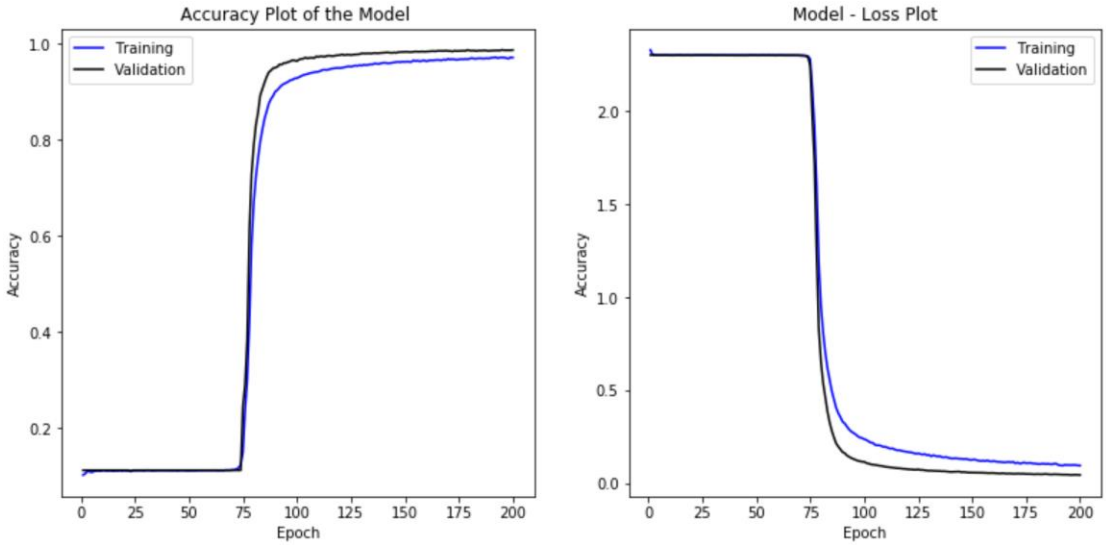


Figure 18: Accuracy and Loss plot for the best Model due to good hyper-parameter setting

IV. Discussion:

1. Filter weight initialization (kernel or weight initializer):

Proper and careful Initialization of filter weights for each layer could determine the difference between achieving great performance and achieving no convergence. Therefore, Initialising weights in a clever and suitable way can be significant and hence, influences how easily the network learns from the training set. If it is done well then, the loss function can be optimized efficiently. Here, the default Xavier or Glorot initialization, Random Normal, Random Uniform and Variance Scaling kernel initializer is used and results are gathered.

2. Batch-Size

Batch size is the total number of training examples present in a single step of gradient descent. If the batch size is too small then the training time until convergence goes up and if the batch size is very large then the training time until convergence reduces but is still not the lowest. Hence, batch size must be optimally chosen so that it is neither too high nor too low to achieve minimum training time until convergence. Smaller batch sizes also give better resultant quality of the model as they generalize better with respect to the model. Hence, here we have selected an ideal batch size of 128 which gives efficient computation and low training time for convergence. Other batch sizes like 20 and 400 were tried but were influenced by other hyper-parameters like learning rate, momentum etc. If all the other hyper-parameters were kept constant and batch size would have varied. We would see that training time until convergence would be the best for an ideal optimal batch size, neither too small nor too large. The variance of the parameter changes is updated by the optimizer- Stochastic Gradient descent is dramatically reduced using batches but by using smaller batch size we get less accurate estimate of the gradient.

For improvement in Performance:

Here we changed some hyper-parameters and introduced new ones apart from the given CNN architecture from part (b) mentioned in the approach and procedure section. These hyper-parameters are given below:

1. **Dropout**: This hyper-parameter is responsible for the prevention of over-fitting. Using this we remove a certain percentage of neurons present in the network after the sub-sampling is done. We introduce drop out after each sub-sampling layer. The effect of this term, forces the network to learn valuable and strong features with respect to the different neurons present in the network layer. We introduce dropouts after S2 and S4 layer in the Lenet-5 CNN architecture with values 0.7 and 0.3.

2. **Kernel Size**: We change kernel size in the C1 layer from 6 to 32 and in C3 from 16 to 64, so that we obtain more feature activation maps containing higher number of features.
3. **Pooling**: In the previous architecture, we used average pooling. We replace this with max-pooling as it is the most efficient pooling technique to achieve sub-sampling.

4. **Activation function**: Instead of the Sigmoid function we use the ReLu activation function. ReLu does not allow the gradients to disappear especially during back-

propagation hence, it does not have any loss of information due to which the classification accuracy increases. This is the major advantage of using ReLu instead of Sigmoid function. Another advantage is that ReLu represents the responses from the neuron in sparse form as all the negative components are clipped off. Hence, by this property we can achieve low convergence time during training. We can clearly see from the experimental results that the classification accuracy (Test Accuracy) reached 99.31% and in less time.

5. **Optimizer**: We change the optimizer from Stochastic Gradient Descent Optimizer to the Adam optimizer. The main reason for this is that SGD takes more time to achieve convergence than Adam optimizer. This makes Adam optimizer the favourable choice in training neural networks.

The final Hyper-parameter values utilized for getting the best-case result are given below with values:

Kernel Initializer: Random Normal

Learning rate: 0.1

Momentum: 0.5

Batch size: 128,

Epochs: 200

Activation function: ReLu

 # filters in C1 layer: 32,

 # filters in C3 layer: 64

Spatial pooling: Max-Pooling

Dropout: 0.7, 0.3 after each Max-Pooling stage

Optimizer: Adam

Train Accuracy: 99.95, Test Accuracy: 99.31

2. Saak Transform applied to the MNIST dataset

I. Abstract and Motivation

The Saak transform is inspired by the CNN concept but adds its own flavour on top of it using traditional methods. The new concept on top of the traditional CNNs is subspace approximation and kernel augmentation. It is an efficient, scalable and robust methodology for the recognition of Hand-written digits in the MNIST dataset. It uses spatial and spectral properties to represent the input images. It abides by the traditional methods in terms of having a strong mathematical foundation and selects features automatically rather than by "hand-engineering". It is inspired by the the multilayer RECOS (REctified-COrrelations on a Sphere) transform concept and is used for forward and inverse transforms for image analysis and generation. It is a powerful idea as it presents an alternative to CNNs in certain application scenarios and hence, we are motivated to study its architecture, functionality, differences between itself and CNNs, application on the MNIST dataset and also to perform error analysis on it.

II. Approach and Procedures

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.

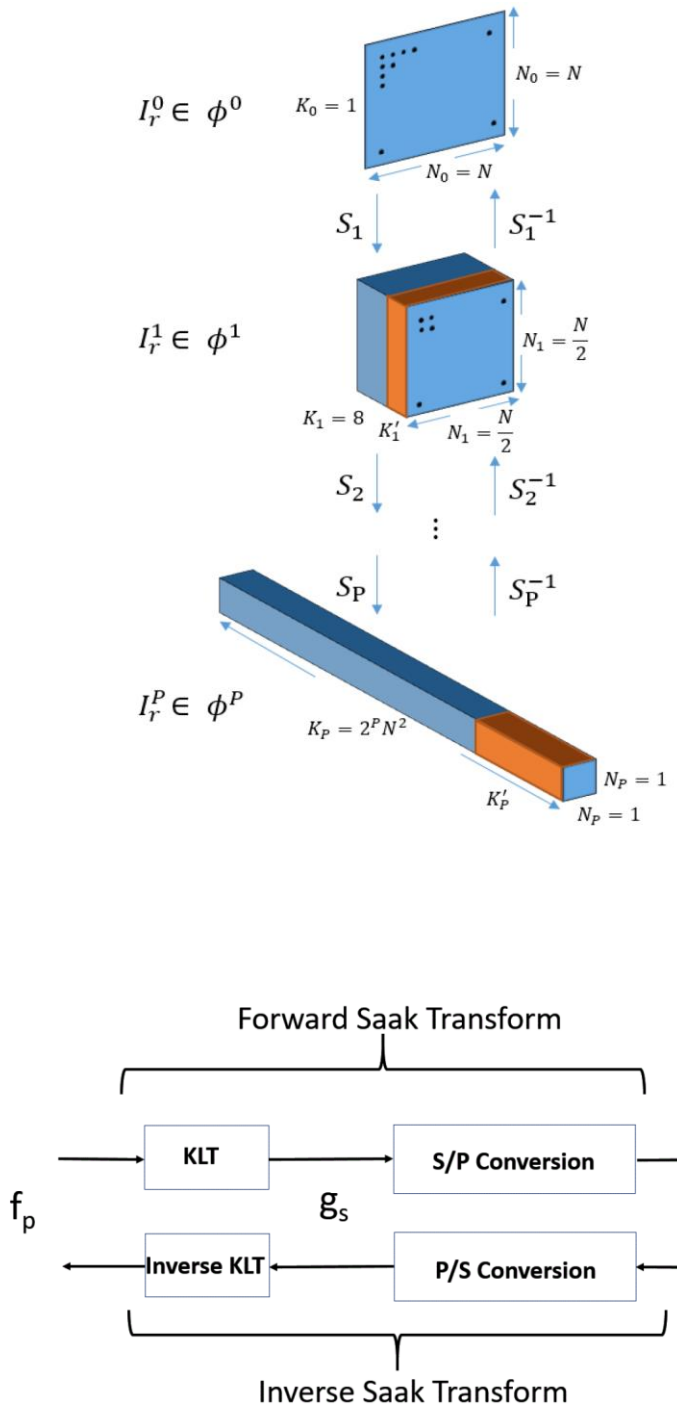


Figure 19: Saak Transform forward and Inverse Transform, top: pictorial depiction, bottom: block diagram depiction

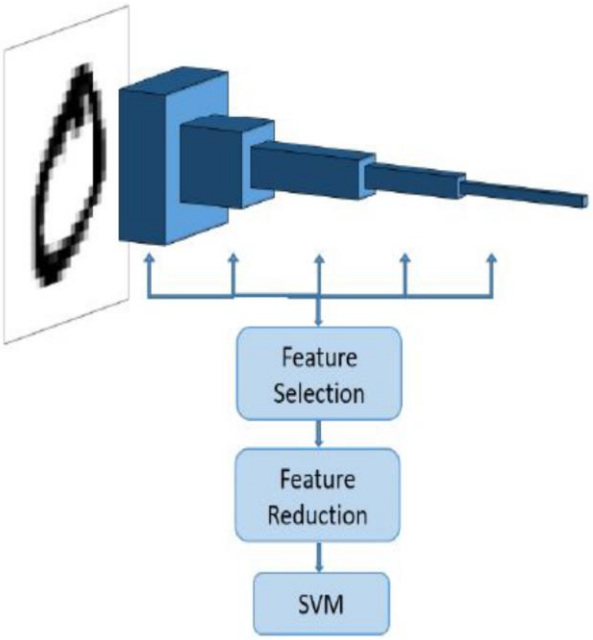


Figure 20: Saak Transform Stage by Stage Process

III. Experimental Results

```
wdir='C:/Users/ishan/Desktop/MNIST/Saak
Train size =
 59000
Test size =
 10000

Classification using SVM
128 D reduced features
Train accuracy: 0.9810072
Test Accuracy : 0.9791
64 D reduced features
Train accuracy: 0.986366
Test Accuracy: 0.9821
32 D reduced features
Train accuracy: 0.993456
Test Accuracy: 0.9831
Classification using RF
128 D reduced features
Train accuracy: 0.999108
Test Accuracy : 0.9023
64 D reduced features
Train accuracy: 0.997869
Test Accuracy: 0.931
32 D reduced features
Train accuracy: 0.998794
Test Accuracy: 0.937
```

```
wdir='C:/Users/ishan/Desktop/MNIST/Saak
Train size =
 50000
Test size =
 19000

Classification using SVM
128 D reduced features
Train accuracy: 0.973097
Test Accuracy : 0.971
64 D reduced features
Train accuracy: 0.985349
Test Accuracy: 0.9824
32 D reduced features
Train accuracy: 0.991261
Test Accuracy: 0.9802
Classification using RF
128 D reduced features
Train accuracy: 0.999651
Test Accuracy : 0.9039
64 D reduced features
Train accuracy: 0.999403
Test Accuracy: 0.9189
32 D reduced features
Train accuracy: 0.999875
Test Accuracy: 0.9342
```

```
wdir='C:/Users/ishan/Desktop/MNIST/Saak-ter
Train size =
 40000
Test size =
 29000

Classification using SVM
128 D reduced features
Train accuracy: 0.976781
Test Accuracy : 0.9734
64 D reduced features
Train accuracy: 0.986395
Test Accuracy: 0.9789
32 D reduced features
Train accuracy: 0.992142
Test Accuracy: 0.9806
Classification using RF
128 D reduced features
Train accuracy: 0.999253
Test Accuracy : 0.8889
64 D reduced features
Train accuracy: 0.996789
Test Accuracy: 0.9084
32 D reduced features
Train accuracy: 0.99934
Test Accuracy: 0.9213
```

Figure 21: Accuracies after performing F-test and PCA

Classifier Type	Training size	Dimension reduction after PCA	Train Accuracy	Test Accuracy
SVM	59000	128	98.100	97.910
SVM	59000	64	98.636	98.210
SVM	59000	32	99.345	98.310
RF	59000	128	99.910	90.230
RF	59000	64	99.786	93.10
RF	59000	32	99.879	93.7
SVM	50000	128	97.30	97.10
SVM	50000	64	98.53	98.24
SVM	50000	32	99.126	98.02
RF	50000	128	99.651	90.39
RF	50000	64	99.940	91.89
RF	50000	32	99.987	93.42
SVM	40000	128	97.678	97.34

SVM	40000	64	98.639	97.89
SVM	40000	32	99.214	98.06
RF	40000	128	99.925	88.890
RF	40000	64	99.678	90.84
RF	40000	32	99.934	92.13

Table 22: Performance table for Saak Transform

IV. Discussion

Part (a)

Similarities between Saak transform and CNN solution:

1. Architecture and operations used

There are similarities in the design architectures of Saak transform and CNN system. In both systems the convolution operations are performed to get feature maps which are then fed to a ReLu Layer before passing it on to the next layer. Therefore, both Saak and the CNN have similar architecture with two additions in the Saak transforms on top of the conventional CNN. These two new operations in the Saak transform are namely: the subspace approximation and the kernel augmentation. In Saak Transform we have Saak coefficients analogous to the filter weights in the CNN system.

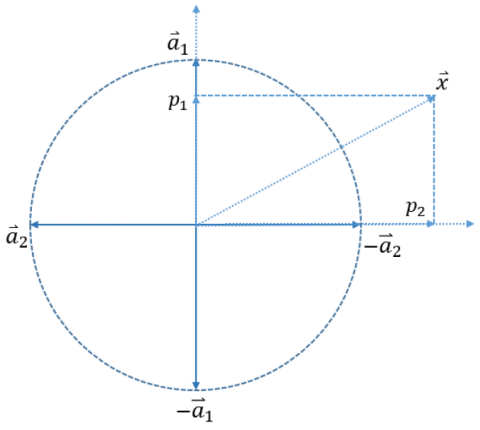


Figure 22: Diagram for Kernel Augmentation idea

Differences between Saak transform and CNN solution:

1. **Architecture Design and Flexibility**

   The CNN has an end-to-end design where in the filter weights are optimized using back-propagation throughout the epochs whereas the Saak transform follows the traditional pattern recognition process of first extracting features and then does classification.

   The problem with CNNs is that if the number of object classes are changed where in we increase or decrease the number of object classes, then all filter weights in the entire network which contain information about the extracted features and the decisions, have to be re-trained. This is very disadvantageous as it will be very time consuming for re-doing the whole-process. In contrast to this, due to the modular design of the Saak Transform the network is less sensitive to the change in the number of object classes as the kernels in the previous stages in the Saak Transform do not change much due to less change in their covariance matrices. Therefore, the Saak Transform is immune to this disadvantage and hence, the same network can be used for feature extraction for the new set of object features.

   CNNs are also not robust against small perturbations in the network whereas again due to the modularity of Saak transform, the last stage Saak coefficients do not change due to the application of PCA which reduces the number of kernels in the stages. The Saak Transform method is especially immune to salt and pepper noise (these come under perturbations) which makes it a very desirable method to use in Images with high noise present in them.

2. **Image Synthesis**

   Saak has an inverse transform which can be used for image generation or synthesis. This is a big discovery as CNNs cannot directly perform such an operation. In CNNs they would have to use two indirect methods in conjunction. These methods are the use of Generative Adversarial Network (GAN) and the Variational Auto Encoder (VAE). Using the afore-mentioned techniques for CNNs is painful to use. The Inverse Saak transform has orthonormal kernels and hence can use KLT to easily get back an image after breaking it down for classification.

3. **Mathematical foundation**

   The CNNs have excellent performance owing to the recent concepts of scatter networks, tensor analysis, generative modelling, relevance propagation, Taylor decomposition etc. However, As the complexity of the CNNs get more and more, the primary intuition behind the CNN conceptual theory becomes mathematically unreasonable. The CNN working cannot be explained clearly as the network behaves like a black-box. We cannot get a proper explanation on how the parameters are calculated properly in the network and hence, there is a total lack of a strong mathematical model. Everything seems like magic happening inside the network and a lot of hyperparameter change and calculations cannot be reasoned, which makes debugging and understanding the system very difficult.

The Saak Transform, in contrast to the CNNs has strong Mathematical and can be demonstrated clearly.

4. **Parameter determination, feature selection and miscellaneous**

In CNNs, the filter weights are updated using the class and input data using backpropagation over many epochs by learning best parameters by optimizing the cost function. In case of Saak Transform the kernel coefficients or filter weights are selected automatically through the stages using KLT.

Features in CNN are learned through backpropagation and are automatically selected. In the case of Saak, the features are extracted based on the discriminative power of the multi-stage Saak coefficients and form a vector to form features.

CNNs perform convolution on over-lapping cuboids and then push the responses through the successive layers whereas in case of Saak, Non-overlapping action is employed on the cuboids and hence, there is no additional pooling layer required as PCA is carried out for dimension reduction.

Dimensionality of CNN kernels are odd numbers whereas dimensionality of the kernels produced by the Saak Transform can be even or odd through the stages.

CNN is like a black-box, parameters inside it cannot be theoretically deduced whereas the parameters and their changes or updates can be explained in Saak as it has a strong mathematical base.

Part (b)

From the table given above, we observe that we obtain the best test accuracy for the 32-dimensional reduced case through PCA. The Classifier type for this is the Support vector method or SVM. The Highest Test accuracy reported is 98.31%.

We can make a keen observation from the tabular results, we see that the train accuracies of SVM are lower than that of the Random forest or RF classifiers. But we also observe that the RF classifiers have lower test accuracies than the SVM method. We observe that the magnitude of difference between the corresponding train and test accuracies for RF is really high when compared to that of SVM. Therefore, we can arrive at the conclusion that RF classifiers have a tendency to over-fit the train data which hence, hampers its test accuracy.

Part (c)

Error Analysis

When data is misclassified by an algorithm, it is said to commit an error. The lower the error rate better is the algorithm. Hence, we need to perform error analysis to justify the performance of any classifier. The errors result due to data imbalance, Missing data, bad algorithms used for classification, high noise in the data etc.

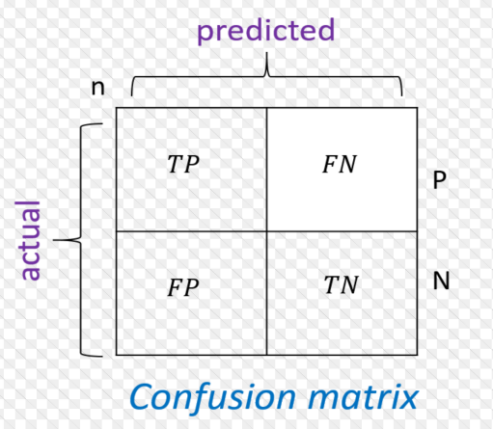


Figure 23: Confusion Matrix

The confusion matrix can be a vital tool for error analysis. The figure for it is shown above and is self-explanatory.

```
[[ 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]]
```

Figure 24: CNN Confusion Matrix

```
[[ 973    0    4    0    0    2    3    0    2    3]
 [   0 1130    0    0    1    0    2    4    0    5]
 [   1    2 1015    1    2    0    0    7    1    0]
 [   0    0    1  995    0   10    1    1    3    7]
 [   0    0    1    0  962    1    2    1    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 1004    2    4]
 [   2    1    2    3    1    2    2    1  958    2]
 [   0    1    1    2   12    1    0   10    2  976]]
```

Figure 25: Saak Confusion Matrix

Observing the above Confusion matrices for both CNN and Saak we see that the matrix has actual values on the one hand and predicted values on the other. We clearly observe that the principal diagonal elements have extremely large values in both cases. Hence, both Saak and CNN can be found to perform well. For smaller dataset Saak performs better and for larger datasets CNNs perform better based on our experimental analysis.

Improvement of performance for CNN and SAAK

For both Saak and CNN

Data Method:

We can improve the performance for both Saak and CNN by providing larger amount of data to it. So, we would need large quantity as well as meaningful data for feeding to the two systems. We can get more data by generating them through replication or changing orientation of images etc. Rescaling data (Normalization or Standardization) to the range of the specific activation function also reduces lot of loss of information clipped off by the activation function. Resampling of data can be done to prevent the ill-effects of data imbalance.

CNN Only

Algorithms and parameter tuning:

We can use better algorithms for specific problems. First, we can try them and see performance and then rank them in order to see which one performs the best. After that we can tune the hyper-parameters of the CNN in accordance to get a better solution.

Ensemble Models:

We can combine different models and extract predictions from each model and decide or threshold based on the majority decision. This will definitely boost performance in the CNN.

Saak Only

Ensemble Models:

We can combine different classification models and extract predictions from each model and decide or threshold based on the majority decision. We can use the outputs from KNN, SVM, Random forests etc to come up with a good decision. This will definitely boost performance in the Saak.

Feature Selection:

We can devise better ways of feature selection of Saak Multi-stage coefficients rather than just selecting Saak-coefficients to form features based on F-score. Rather than using PCA which is a dimensionality reduction technique, we could use Random forest decision tree model and select a threshold to see which features are more important and select those Saak features accordingly.

Inverse Saak Transform

The Inverse Saak Transform can be deployed to applications and performance can be observed after which comparisons with GANs and VAE can be done.

References:

All Figures and Materials were used and refered to from the below sources:

1. https://en.wikipedia.org/wiki/Convolutional_neural_network
2. https://www.slideshare.net/Tricode/deep-learning-stm-6
3. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
4. https://sefiks.com/2017/11/08/softmax-as-a-neural-networks-activation-function/
5. "**A Taxonomy of Deep Convolutional Neural Nets for Computer Vision**"- Suraj Srinivas, Ravi Kiran Sarvadevabhatla, Konda Reddy Mopuri,Nikita Prabhu, Srinivas S S Kruthiventi and R. Venkatesh Babu Video Analytics Lab, Indian Institute of Science, Bangalore http://val.serc.iisc.ernet.in/
6. https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html
7. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
8. https://en.wikipedia.org/wiki/Rectifier_(neural_networks). https://en.wikipedia.org/wiki/Softmax_function
9. Saak Transform https://arxiv.org/abs/1710.04176
10. http://yann.lecun.com/exdb/mnist/h