2018

HOME WORK 4: IMAGE RECOGNITION WITH CNN AND SAAK

EE 569 – INTRODUCTION TO DIGITAL IMAGE PROCESSING

HARIKRISHNA PRABHU (3333077042)

PROBLEM 1: CNN Training and Its Application to MNIST Dataset:

AIM:

To learn a simple Convolution Neural Network derived from the LeNet-5 architecture.

(a) **CNN Architecture and training:**

- To explain the architecture and operational mechanism of the convolutional neural network.
- To describe the CNN components. Such as,
    - 1) The fully connected layer,
    - 2) The convolutional layer,
    - 3) The max pooling layer,
    - 4) The Activation function and
    - 5) The Softmax function.
- To explain on what is the over-fitting issue in the model learning? Explain any technique that is used to overcome this overfitting issue.
- To explain why CNN works better than many other traditional methods used in Computer Vision techniques.
- To explain the loss function and the classical back propagation optimization procedure to train such neural networks.

(b) **Train the LeNet-5 on the MNIST Dataset**

- To explain the initialization of network parameters such as filter weights, learning rate and decay etc.
- To compute the accuracy performance curves epoch wise and plot the performance curves for different parameter settings.
- To find the best parameter to achieve the highest accuracy on the test set and plot the performance curves for the Test and Training Dataset under this performance parameters.

(c) **Improve the LeNet-5 for MNIST Dataset:**

Note: Feel free to modify the baseline architecture of LeNet-5. I.E, add or modify more convolutional layers, number of filters, activation functions and so on.

- To report the best accuracy achievable and describe/explain the network architecture and parameter setting. Mention the sources for performance improvement.

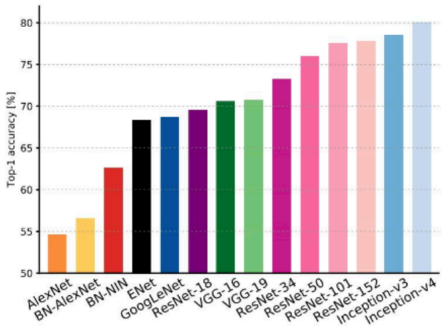
ABSTRACT AND MOTIVATION:

What is a Neural Network? It is a computer system modelled to mimic the mammalian brain. It is an alternative approach to computing based on/and to understand the functioning of the human brain. It composes of a large number of highly interconnected processing elements (Neurons: Nodes) functioning in parallel to solve a particular problem. The distinguishable feature of neural networks is that it cannot be programmed to perform a specific task but, has to learn through the examples.

Convolutional Neural Network (CNN): It is descriptively the neural network but not essentially a fully connected neural net. I.E, the neurons in one layer are connected to only few of the neurons in the previous layer. This CNN wouldn't have been possible if it were not for the avalanche advent of GPU's in the industry and for the rise of big data and data mining. CNNs have proved to be a breakthrough in the field of computer vision. When compared to other prevailing image classifiers, CNN requires very little to no preprocessing of data and is independent of prior knowledge of data. This makes CNN very favorable to use.

Our objective is to get to know the mechanism/functioning of CNN, on how it learns to and to classify the MNIST dataset. The MNIST dataset is a set of 60000+ images of handwritten digits from 0 through 9. Here we'll learn on how CNN learns through the features extracted from the images at each layer using multiple filters and how the learnt model functions to classify the unpresented data.

Let us look at some of the CNN architectures and its corresponding performance evaluations.



Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Figure 1: CNN architectures and its evaluation table.

Here in our project we will be studying about the LeNet-5 architecture.

APPROACH AND PROCEDURE:

The general structure of any Neural network including CNN is shown below.

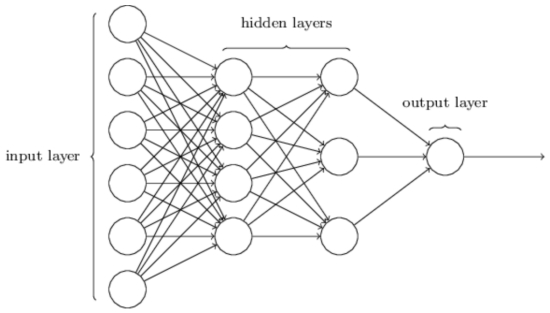


Figure 2: Neural Networks.

Boiling down to the simplest form, this network structure down samples the input features based on its prominence. Brief explanation for the structure and architecture of the convolution network is given in the discussion section. Please follow there.

Procedure:

General **Convolutional Neural Network:**

- Image from the training data set is passed as the input to the first convolution layer.
- A convoluted output is obtained as the activation map upon which a number of filters are applied to extract features. Thus, a block/volume of image features is formed. Different/Random filters are applied to this activation map to give different features that aid in prediction of class, thus they are all discriminative in nature.
- Her either zero padding or valid padding is done based our requirement to either retain the image's size the same or reduced. Valid padding is often used to reduce the complexity and time of feature extraction.
- Then this spatial-spectral volume of images is passed to the pooling layer (which can be either average pooling or max pooling) which reduces the number of parameters at play.
- This then is passed further to the next convolutional layer and pooling layers respectively and is repeated until satisfied. And it is flattened to obtain a fully connected layer (spectrum).
- This fully connected layer is densed out to equal the number of classes required by the problem statement.
- Then we see that, as we pass an image as input, it gets flattened to the output that is equal to the class size desired by the network. This predicted output is compared with the actual ones to calculate the loss (loss function).
- The calculated loss is back propagated into the network to modify the filter weights and bias so as to obtain a steady learned model.

- The model's training is complete in one forward and backward pass. Then we can pass the test images to compute its performance and classification accuracy.

LeNet 5 architecture and procedure:

The LeNet 5 architecture consists of 2 convolutional layers followed by sub sampling as described by LeCun and is followed by 2 hidden fully connected layers. Layer wise procedure is given below.

- *Input layer:* The Images from the MNIST data set is loaded into the input layer. The images are of dimension/size 32x32. Therefore, the input has 32x32 neurons arranged spatially.

Convolution Layer 1: In this layer, 6 kernels/filters of size 5x5 is applied to the input image and 6 feature maps of size 28x28 is produced. The 'Sigmoid' activation function is used to fire the neurons to the next layer with stride = 1. Thus, at the end of this stage, Convolutional Layer 1, we will have 156 [(5x5 + 1)x6 = 156] training parameters and 122,304 [$28x28 \text{ x } (5x5 + 1)x6 = 122304$] connections. (Fact: This layer has a spatial structure that is sparse enough and has weight vectors.)

- *Sub-sampling layer 1*: Here the output from the convolutional layer 1 is fed and the feature map is reduced to half of its original size (14x14) by means of average pooling. i.e 2x2 window of all 1s is used. At the end of this stage, we're left with 6 feature maps of size 14x14 with 12 [6 x 2 = 12] learnable parameters and 5880 [14x14 x (2x2 + 1)x6 = 5880] connections. (Fact: It has no shared weights/ weight vectors). As said previously, the 'Sigmoid' activation function is used to trigger the neuron into the next layer.
- *Convolution Layer 2:* Here, 16 feature maps of size 10x10 is produced from the 14x14 feature map of the previous layer. Here too 5x5 sized kernels are used to produce the feature maps. This results in having 1516 trainable parameters and 151600 connections.
- *Sub-Sampling Layer 2:* Like Sub-sampling layer 1, here too the 16 feature maps are reduced to half its size i.e. 16 5x5 feature maps that have 32 [16 x 2 = 32] trainable parameters and 2000 [5x5 x (2x2 + 1)x16 = 2000] connections. Here too the 'Sigmoid' activation function is used for triggering the neurons.

Fully connected layer: In this flattening layer, 120 feature maps of size 1x1 is produced. In our case it is called fully connected layer, if our input image is of larger dimensions then this layer would act as another convolutional layer instead. This layer produces a fully connected layer of 48120 [$120 \text{ x } (16x25 + 1) = 48120$] trainable paramters.

- *Densed-down Fully connected layer:* This layer denses down the 120 connected units to 84 fully connected units with 10164 trainable/learnable parameters.
- *Output Layer:* This final output layer has 10 fully connect units [should be equal to the number of classes required by the problem statement] representing the output class labels. Here, once the output layer is

obtained, softmax function is applied to obtain the probabilities of each output labels. (Fact: the output layer is made of Radial Basis Function (RBF))

- Again, as mentioned in the previous part, error is estimated using the loss function and is back propagated to modify the weights and fit the training model.

Please follow the discussion section for the explanation of network parameters.

EXPERIMENTAL RESULT:

(b) LeNet 5 architecture:

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 6)	0
conv2d_2 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_2 (MaxPooling2	(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

Figure 3: Table showing the dimension reduction through the layers in the CNN

(i) Parameters:

Class size = 10 : *Activation* = Sigmoid : *Padding* = SAME : *Optimizer* = SGD
Loss Function= Categorical Cross-Entropy : *Decay* = default

TABLE 1

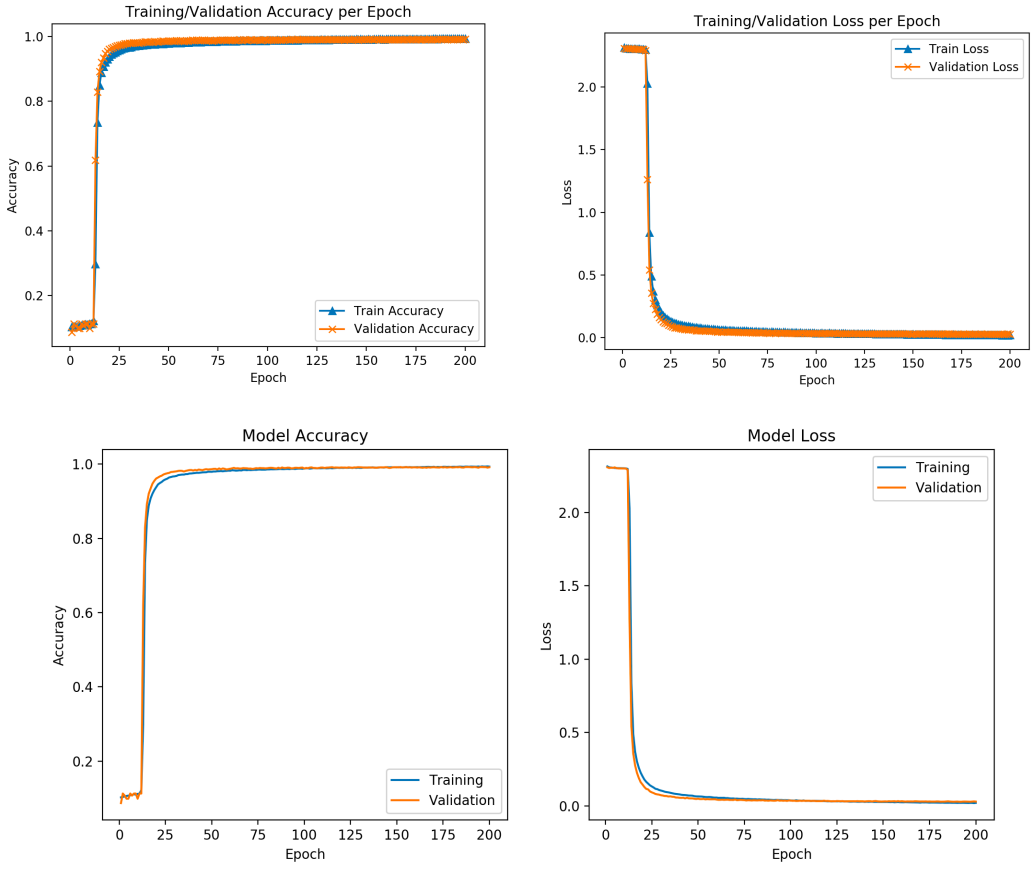
Figure Number:	**1**	**2**	**3**	**4**	**5**	**6**
Batch Size	128	196	196	196	64	128
Epochs	150	200	200	400	200	200
Kernel Initializer	Truncated Normal	Glorot Normal	Glorot Normal	Random Normal	Random Normal	Random Normal
Learning Rate	0.1	0.1	0.3	0.01	0.1	0.001
Momentum	0.5	0.5	0.7	0.5	0.25	0.5
Training Accuracy	99.91	99.97	99.93	99.98	99.3	98.96
Classification Accuracy	99.07	99.20	99.22	99.14	99.04	

The best classification accuracy obtained from the above observations is 99.22 % and its respective parameter setting is,

Batch Size: 196
Epochs: 200
Kernel/Weight Initializer: Glorot Normal
Filter size for Convolutional Layers: 6,16
Optimizer: SGD
Activation Function: Sigmoid
Pooling: Average pooling
Learning Rate: 0.3
Momentum: 0.7
Training Accuracy: 99.93
Test Accuracy: 99.22

Respective Accuracy and Loss is plotted for each parameter setting.

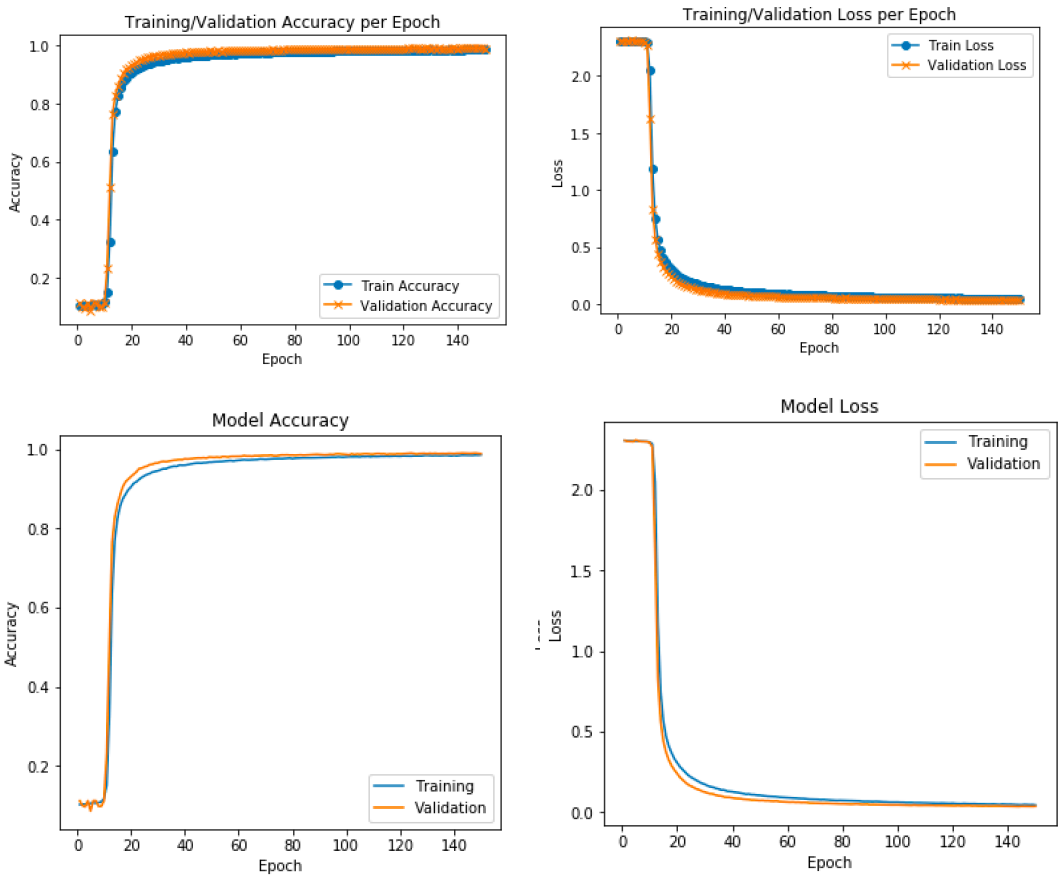
(1)




```
Model took 4945.71 seconds to train

LeNet 5 Test Error: 0.9300%
LeNet 5 Test Loss: 0.0271%
LeNet 5 Test Accuracy: 99.0700%
```

Figure 4: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

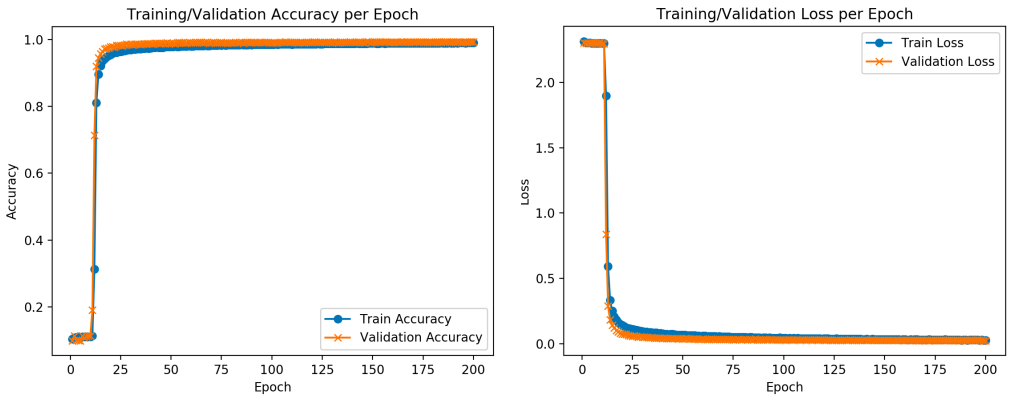
(2)

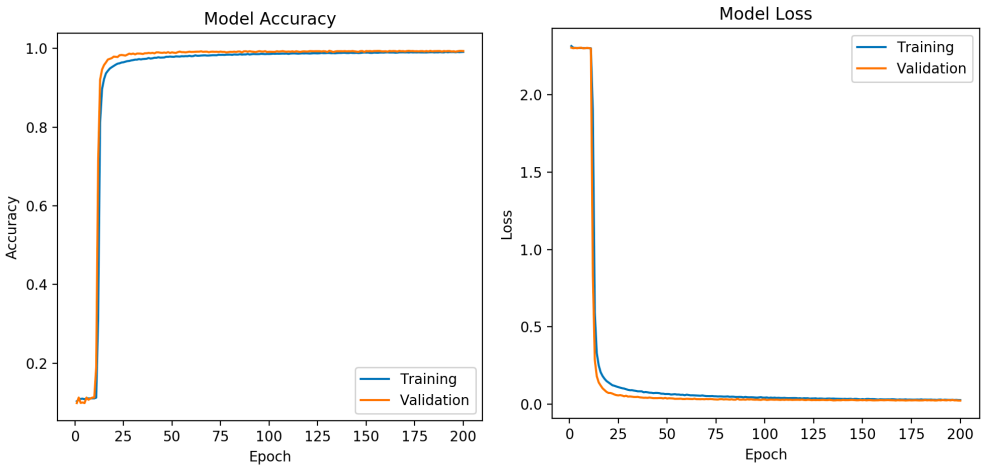



```
LeNet 5 Test Error: 0.8400%
LeNet 5 Test Loss: 0.0300%
LeNet 5 Test Accuracy: 99.2000%
```

Figure 5: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(3)

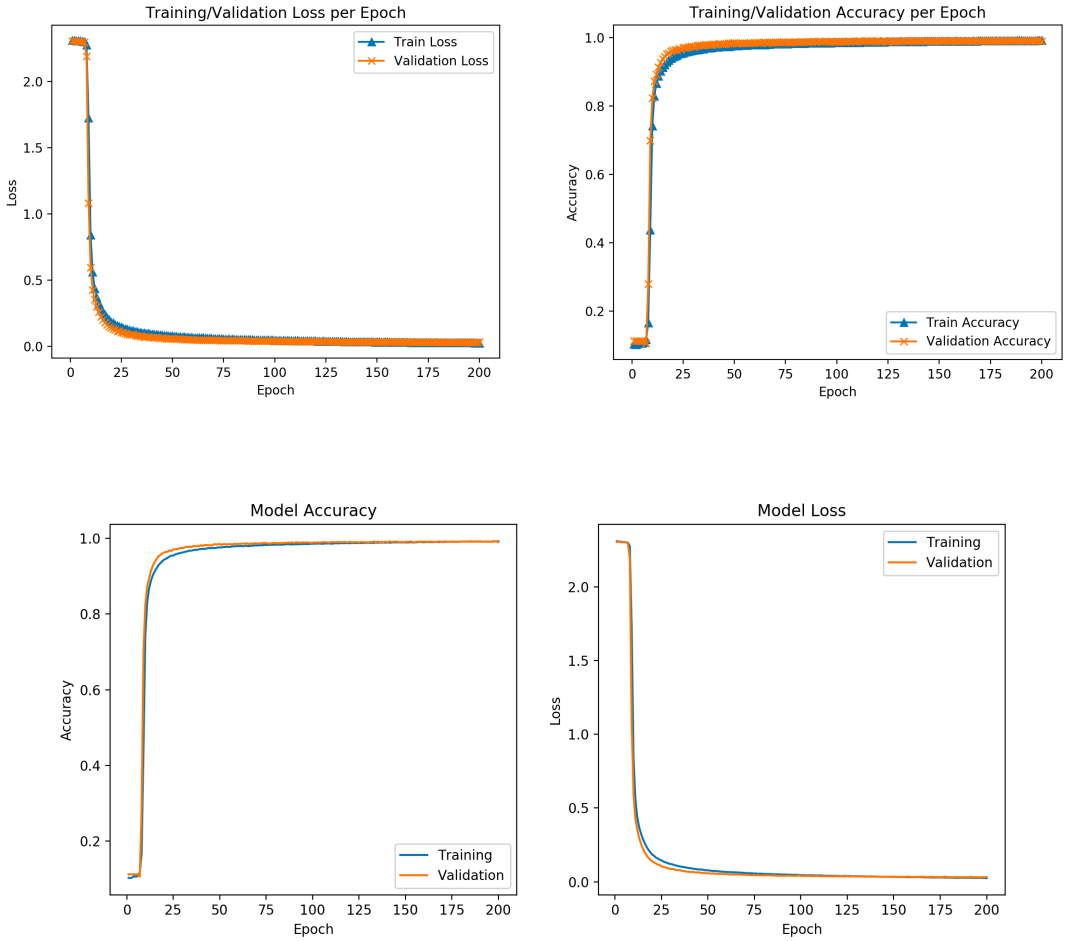





```
Model took 3779.25 seconds to train

LeNet 5 Test Error: 0.7800%
LeNet 5 Test Loss: 0.0233%
LeNet 5 Test Accuracy: 99.2200%
```

Figure 6: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(4)



```
Model took 6676.66 seconds to train

LeNet 5 Test Error: 0.8600%
LeNet 5 Test Loss: 0.0273%
LeNet 5 Test Accuracy: 99.1400%
```

Figure 7: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(5)

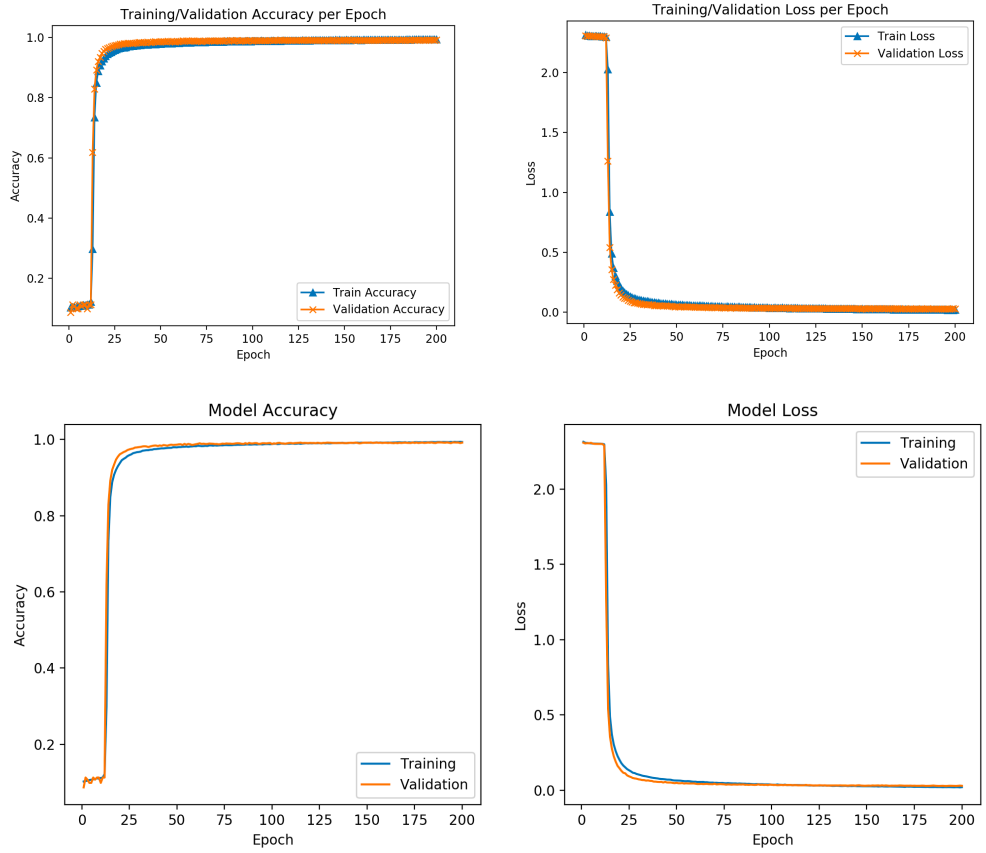
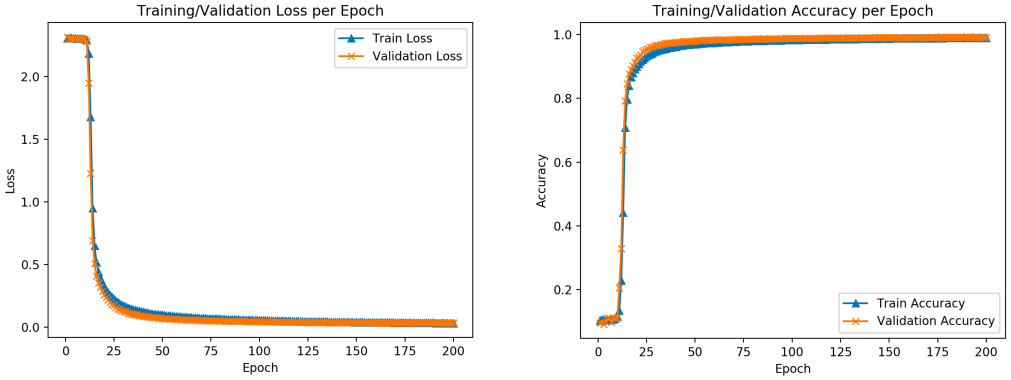
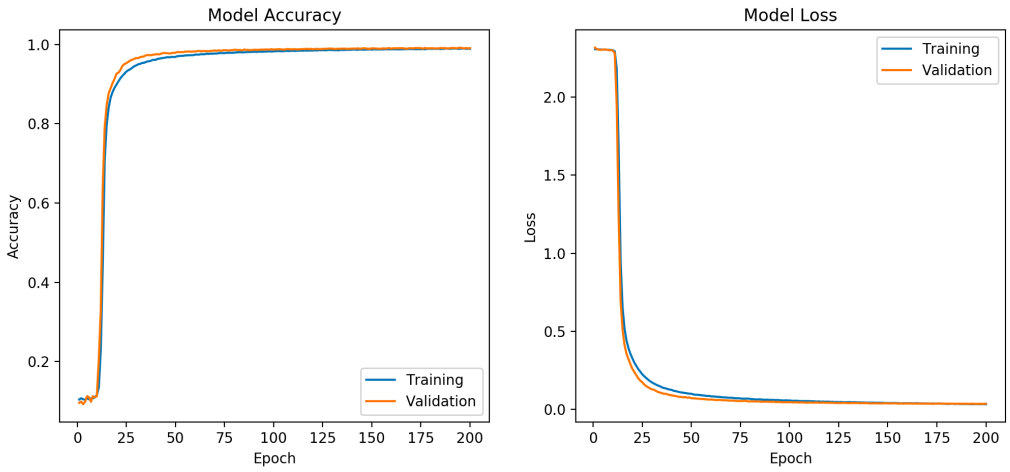


Figure 6: Accuracy plot and Loss plot (epoch wise and model).

(6)






```
Model took 3975.31 seconds to train

LeNet 5 Test Error: 1.0400%
LeNet 5 Test Loss: 0.0291%
LeNet 5 Test Accuracy: 98.9600%
```

Figure 8: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(c) Parameters:

Class size = 10 : Padding = SAME : Loss Function: Categorical Cross-Entropy

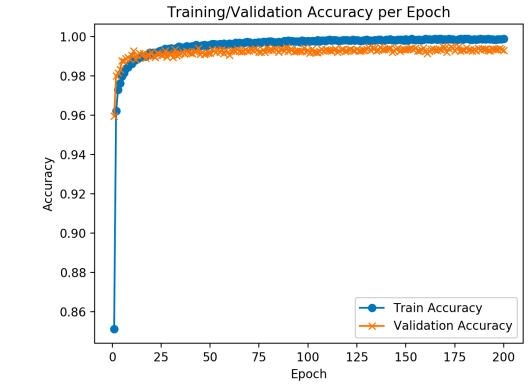
TABLE 2

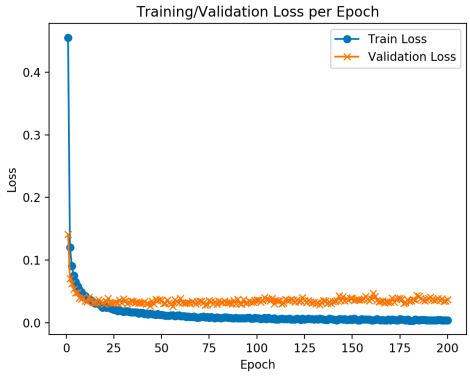
Figure Number:	**1**	**2**	**3**	**4**	**5**
Batch Size	128	128	196	128	64
Filters (Stage:)	6,16	50,100,120	6,16	32,64	32,64
Epochs	200	200	200	200	200
Optimizer	SGD	Adam (b1 = 0.9, b2 = 0.999)	SGD	Adam (b1 = 0.9, b2 = 0.999)	Adam (b1 = 0.9, b2 = 0.999)
Activation	ReLU	ReLU	Sigmoid	ReLU	ReLU
Pooling	Average	Max	Average	Max	Max
Kernel Initializer	Random Normal	Truncated Normal	Truncated Normal	Truncated Normal	Truncated Normal
Learning Rate	0.1	0.0001	0.1	0.1	0.1
Momentum	0.5	-	0.5	-	-
Drop Out	0.25	0.5	0.2	0.5	0.5
Decay	Default	0.0	Default	0.0	0.0
Training Accuracy	99.78	99.56	99.69	99.63	99.48
Classification Accuracy	99.34	99.44	98.99	99.51	99.39

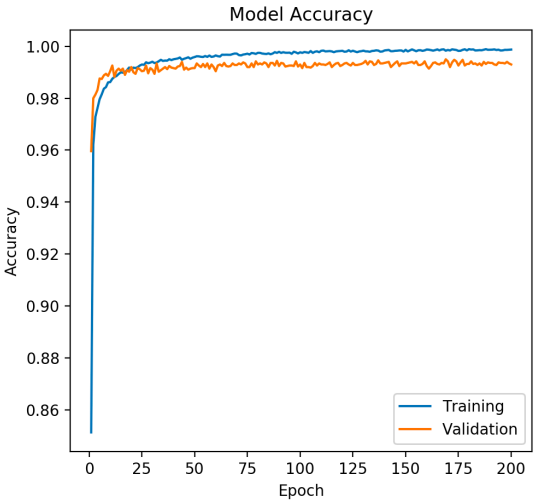
In part (c), The best accuracy is obtained for the following parameter setting,

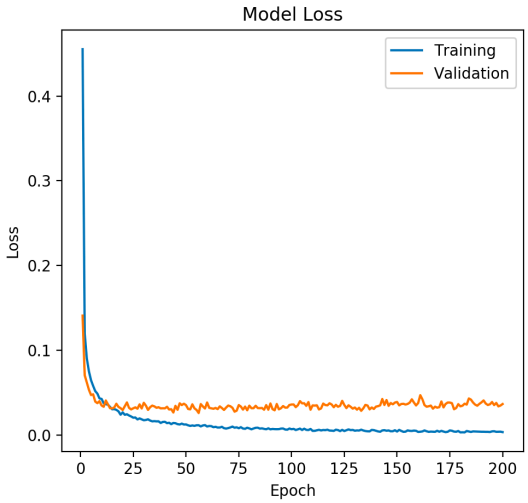
Batch Size: 128
Filter size in Convolutional Layer: 32,64
Epochs: 200
Kernel/Weight Initializer: Truncated Normal
Optimizer: Adam optimizer
Beta_1: 0.9
Beta_2: 0.999
Activation Function: ReLU
Pooling: Max pooling
Learning Rate: 0.1
Drop out: 0.5
Decay: 0
Momentum: 0.5
Training Accuracy: 99.63
Test Accuracy: 99.51

(1)





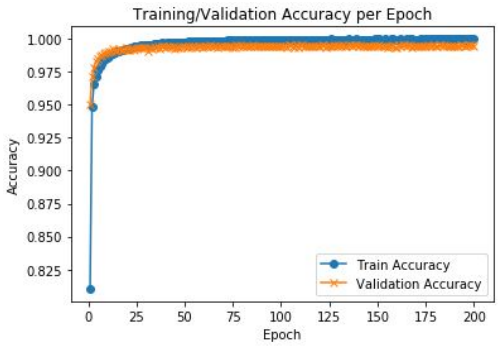


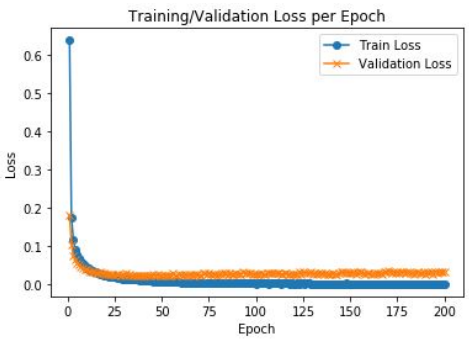


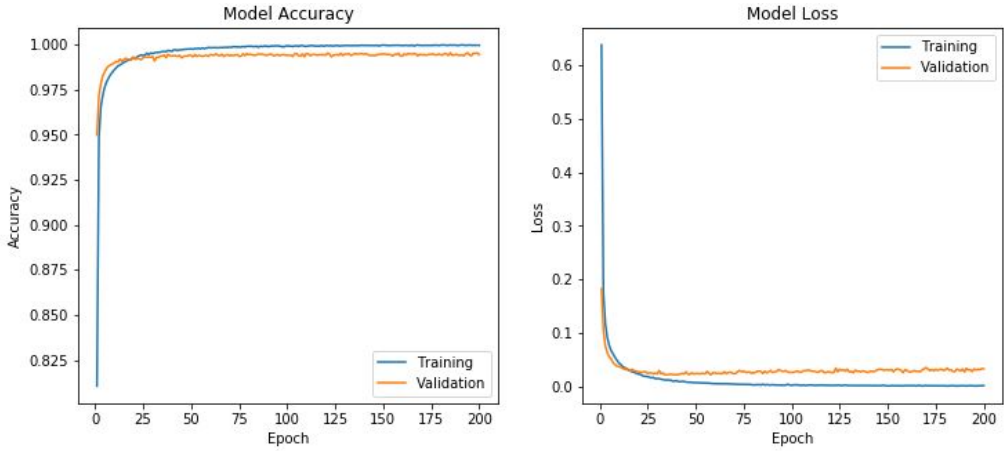

```
Model took 4485.31 seconds to train

LeNet 5 Test Error: 0.6600%
LeNet 5 Test Loss: 0.0311%
LeNet 5 Test Accuracy: 99.3400%
```

Figure 9: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(2)



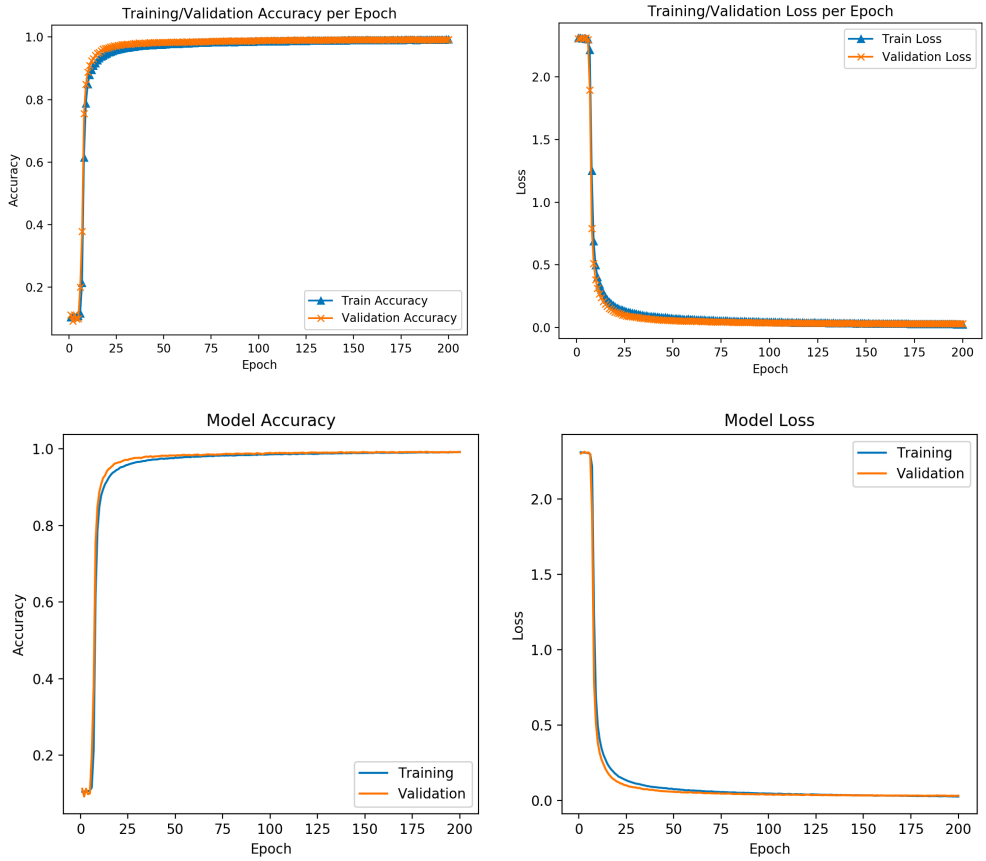





```
Model took 8435.31 seconds to train

LeNet 5 Test Error: 0.5600%
LeNet 5 Test Loss: 0.0288%
LeNet 5 Test Accuracy: 99.4400%
```

Figure 10: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

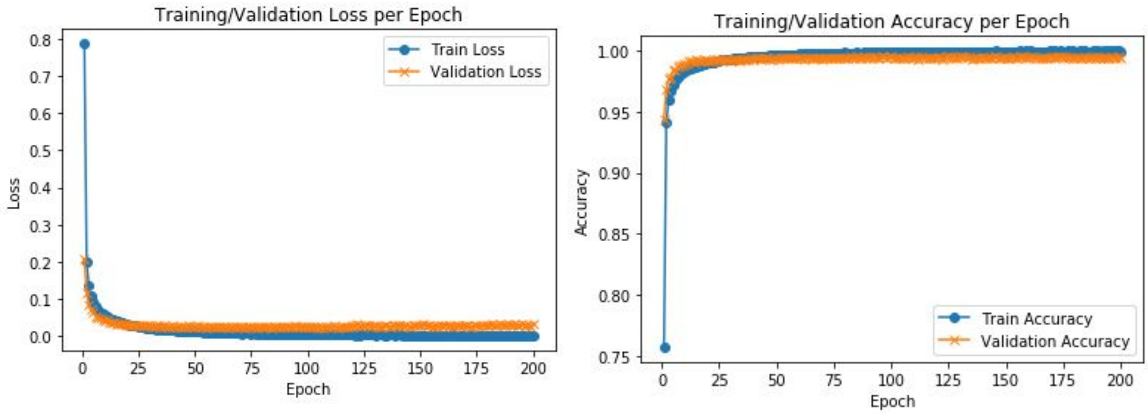
(3)

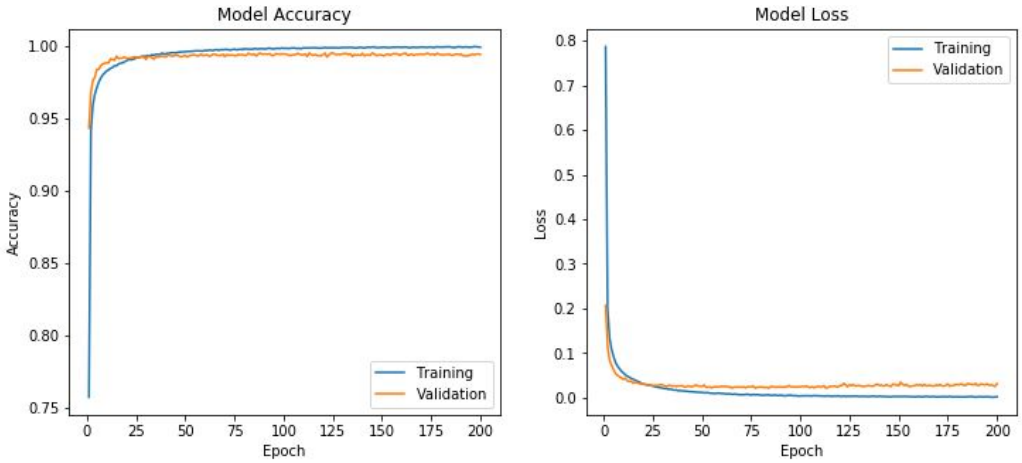


```
Model took 4204.01 seconds to train

LeNet 5 Test Error: 1.0100%
LeNet 5 Test Loss: 0.0293%
LeNet 5 Test Accuracy: 98.9900%
```

Figure 11: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

(4)






```
Model took 4611.82 seconds to train

LeNet 5 Test Error: 0.4700%
LeNet 5 Test Loss: 0.0242%
LeNet 5 Test Error: 99.5100%
```

Figure 12: Accuracy plot, Loss plot (epoch wise and model) and Terminal Output.

DISCUSSION:

a) CNN Architecture and training:

part a:

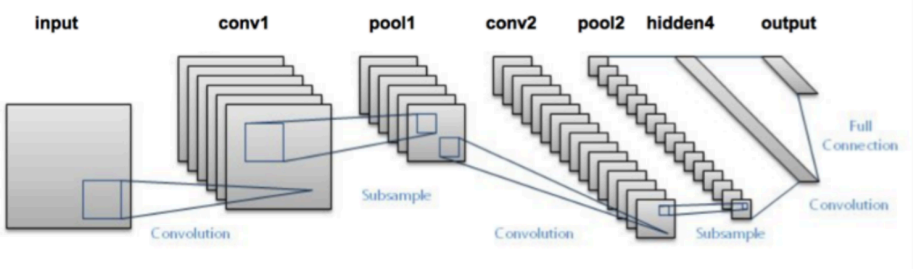


Figure 13: Convolutional Neural Network.

The above picture that represents the work flow of Convolutional Neural Network has multiple layers in it. Let us discuss the structure layer by layer in order to completely understand the working mechanism of the CNN.

[1] Input Layer: The input layer is the data that we pass to the network and here we pass a number of images as input to the system. As human beings, it is easier to comprehend the information present in it. But for a machine, it is pretty difficult to handle that level of understanding. So, what machine does is that it breaks the input image as a matrix of pixels that hold numerical value. In the MNIST dataset, the images (about 60,000 images for training and 10,000 images for testing) have a dimension of 32x32 (height x width) and they are images that hold handwritten digits from 0 through 9. (Therefor the number of classes/clusters is 10). If these images were in RGB the image size would have been 32 x 32 x 3, but here mostly the dataset comprises of black and white images, hence it is as mentioned formerly.

[2] Convolutional Layer: In the convolution layer, we try to extract the features in the image by using the spatial arrangement of the pixels in the image. For a machine to understand the image, it is extremely important for our network to understand the pixel arrangement. Hence, we make use of weighted matrices to extract the features. This weighted matrix is termed as FILTERs. In this step, it is made sure that the filter moves across each and every pixel of the image. This filter moving over the pixels can be defined as a parameter called 'Stride'. If stride = 1, then the filter moves over each pixel and if stride = 2, the filter moves over every other pixel. The weight matrix/filter can function to extract edges in the image, or color or could blur the noise present in the image. This weight matrix adds up the pixel wise multiplication between the filter and the image. As the convolutional layer gets deeper the features extracted by these weight matrices become more and more complex. Another important point to be noted is that, the convolved image/filter processed image has reduced parameters than the actual ones. Therefore, not all the pixel values are assigned to the neurons in the layer but efficient features that represent a region of the, thereby reducing computational complexity at each stage.

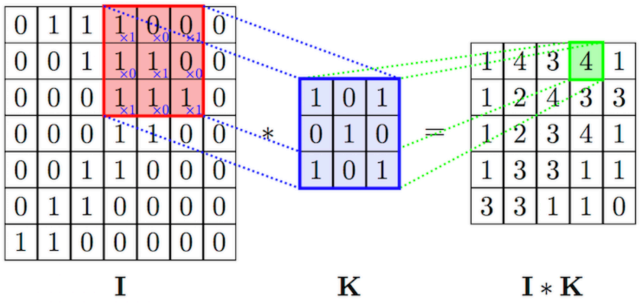


Figure 14: How weight matrix function in the convolutional layer.

In the convolution layer, we use multiple filters applied to the same image but we must make sure that the depth of the weight matrix matches the depth of the image. Given below is an example for how on a 32 x 32 x 3 images, multiple filters [5 x 5] are applied and convolved.

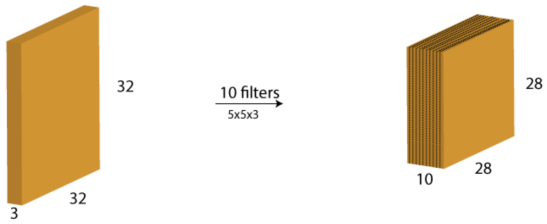


Figure 15: Multiple filters.

We can see this reduction of image size as a result of VALID padding which retains only the necessary features. If it were SAME padding, the size would have been the same and the non-necessary portion of the image will be padded with zero.

[3] Pooling layer: As we know that even after convolving, the number of trainable parameters is too high. So, in order to reduce them we use this pooling layer. Pooling is done independently on each depth dimension and its sole purpose is to reduce the spatial dimension of the image whilst retaining the features. An example is shown below.

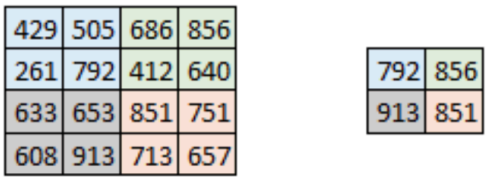


Figure 16: Pooling: Max pooling

What is shown is the Max pooling. Here the maximum of the values in the given region is pulled to represent the region. If 'mean' was used, then it would be called Average Pooling.

Once the image is pooled, it is sent to the next convolutional layer and it is further pooled. By this way the most important/prominent feature is retained and dimension is reduced. By this way we will reach the fully connected layer.

[4] Fully Connected layer: This layer takes the input from the previous layer and given an output that is of 'N' dimensions where N = number of classes. Here for our MNIST dataset, N = 10. We can say that this layer flattens the output obtained from previous convolutional and pooling layer, I.E 2D to 1D array. We see that in many cases there are number of fully-connected layers where the first reduces to about 1024, the next to about 100 features so on and finally ends up to the number of classes for the image dataset. This fully connected layer functions to correlate high level feature extracted from the images to the class. This way once the system is trained, we can follow the same procedure for test images for class assignment.

[5] SoftMax: The SoftMax function takes a 'D' dimensional vector of arbitrary real valued scores and squashes it to a vector of 'D' dimensions whose values lie between zero and one, such that they sum upto 1.

$$\sigma : \mathbb{R}^K \to (0,1)^K$$
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Figure 17: Expression for SoftMax

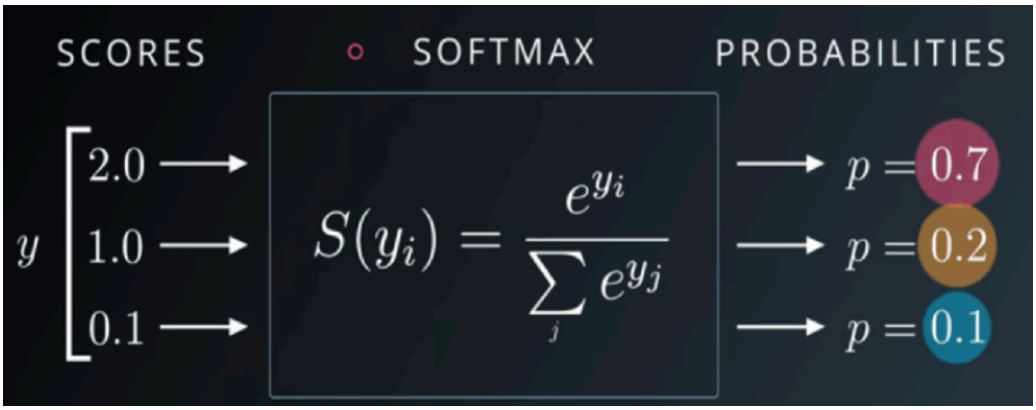


Figure 18: SoftMax

In the CNN, the SoftMax is used after the final fully connected layer. The reason for the usage of SoftMax is that there are cases where we use cross-entropy to train resulting in a non-linear, multinomial logistic regression. Hence SoftMax is used.

Mathematically, SoftMax can be defined as a generalization of the logistic function (example for logistic function, Sigmoid or 'S' type curve).

[6] Activation Function: What an artificial neuron does is it calculates the weighted average and its expression can be seen below.

$$Y = \sum (weight * input) + bias$$

And this value 'Y' can range anywhere between (-∞, ∞) and the neuron doesn't know its bounds and whether to fire or not. This, we would have noticed that among all the neurons (node) present in the single layer (of multiple layer that exists), not all neurons are fired.

Whether or not the neurons should be fired is decided by the **activation function**. The activation function of the node decides the output of that node given an input or set of inputs. This activation function can be linear or non-linear in nature, preferably non-linear because…

Let us see some of the activation functions,

(i) Step function: It is the simplest of the activation function. Based on the set threshold it decides if the neuron should fire or not.

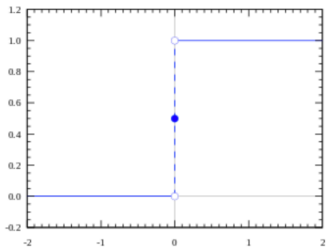


Figure 19: Step function.

Expression for step function is,

$$u_c(t) = \begin{cases} 0 & \text{if } t < c \\ 1 & \text{if } t \geq c \end{cases}$$

fires if the weighted sum is greater than the set threshold.

(ii) Linear function:

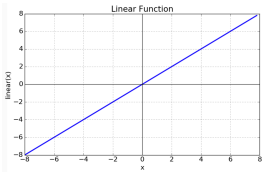


Figure 20: Linear function.

Expression for step function is f(x) = ax. This activation function gives an output proportional to input. But the problem is that, while observing error in predictions, the changes made by back propagation is a constant (as the derivative of the function is a constant) and sometimes it is always. Hence no steps can be taken to rectify the predicted error. Another reason is that while having a number of layers activated linearly, the ending/final layer is also a linear function of the first layer. Hence the point of having multiple layers becomes in vain.

(iii) Sigmoid function: Given in the figure below is the expression and plot for a sigmoid function. It is non-linear in nature and the idea of stacking multiple layers can be put in play. This function is a smooth step function and unlike the linear function, its value is bound between definite integrals. Here given in the figure is bounded between (0,1).

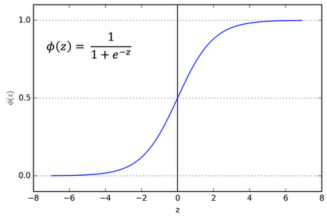


Figure 21: Sigmoid function expression and plot.

It can be seen from the plot that as the function reaches the near horizontal it becomes unresponsive to changes and also, they are not centered at zero, hence they give zig-zag gradients which delays convergence. Examples of sigmoid like activation function are tanh, arctan etc.

(iv) Rectified Linear Unit Function (ReLU) :

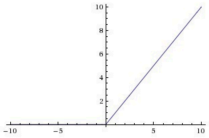


Figure 22: ReLU function.

Mathematically it can be expressed as max (0, x) i.e. it truncates the negative region and gives a proportional output for the positives (This makes the network lighter as only fewer neurons will be fired). Even though it is linear in the positive region, it as a whole is non-linear in nature. But it has its constraints too. Such as, it dies out if learning rate is improperly set and a dead ReLU cannot perform discrimination function. But amongst all the activation functions discussed, ReLUs are the most computationally light, cheap and viable. This problem of dying out of ReLUs is solved using the Leaky ReLU. A Leaky ReLU looks as shown in the figure.

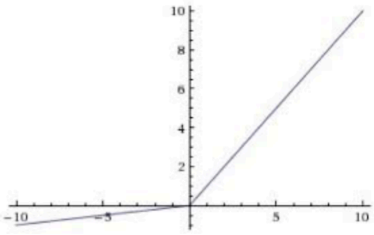


Figure 23: Leaky ReLU

It has a small gradient in the negative side unlike the ‘0’ of ReLUs. Despite not having a zero-centered function, this activation function won’t die out due to improper assignment of learning rates.

Part b:

In any/all learning models, there are chances for the developed model to perform poorly. It is because they either over-fit / under-fit. This can be figured out comparing the classification accuracies of the validation set and the test set. If the accuracy for validation is greater than the accuracy for test set, then the model is said to under-fit and over-fit if vice versa. Example for over fitting: An image from the training data set is correctly classified by the model but an image following the same general pattern is classified wrongly.

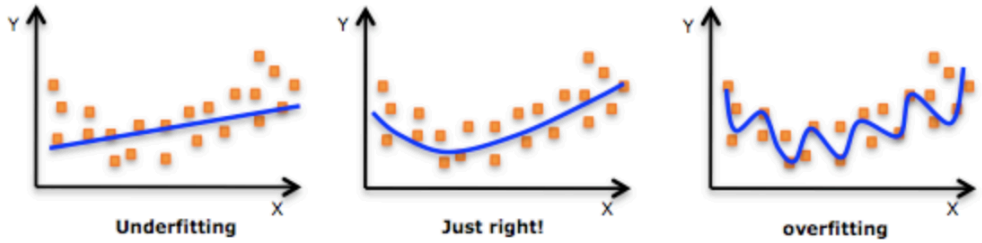


Figure 24: Comparison plot for under-fitting and over-fitting.

Methods to overcome over fitting are,

(i) Increasing the input data set: In many cases it is not possible but in those where increasing the data set is possible, it is advisable to train the model using the new increased training data set. In few cases, people resample the input data set and produce synthetic data that has similar characteristics as of the original input data. This improvised data set can also be used in training the model.

(ii) Removing features: This process can be done using any built-in functions available or manually by observing the features extracted and removing certain features to improve the generalizability of the data set.

(iii) Regularization: It is most common technique used in classical learning models. It refers to that process that forces the model to become lighter and simpler. Example, Drop-out in a neural network, penalty parameter in regression and pruning in decision trees/random forests. It can be seen as a hyper-parameter in the coding structure. Regularization helps in reducing the problem of overfitting as it takes into account only the lower/simpler weights thereby making the network less complex and provides a better powerful explanation of data.

(iv) Early stop: In many of the deep learning models, early stop is preferred. We observe the loss and classification accuracy at each iteration and see that it performs better/learning model improves up-till certain number of iterations and then starts to over-fit the data. So, early stop is to stop the learning process before it crosses this point where it starts to over-fit.

In our case we use drop-out as the regularization technique. It randomly drops some neurons with probability 'p' to zero, hence reducing the risk of complexity and likewise reduces over fitting issue as it restricts the strong features to dominate over the weaker ones.

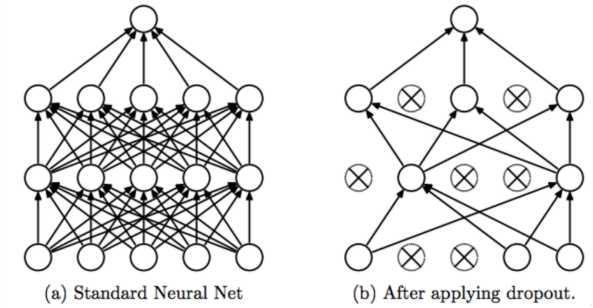


Figure 25: Drop out

Part c:

Traditional way to approach computer vision (CV) problems is to extract the features in the given image [This can be done using SIFT, SURF, Histogram of Gaussian and other techniques], qualify/quantify them based on its prominence in the image and perform classification. This job becomes very tedious as the feature size and training data size increases and due to this fact higher precision in classification is not achievable. Even though there are traditional methods that perform equally good as the CNNs, it becomes more complex as the size of the data increases as explained above and it becomes computationally expensive with real time object classification applications. When it comes to CNNs, it does all the feature handling by itself without requiring manual help irrespective of the size of the training data. Hence in today's world, CNN is the most widely used method in the field of computer vision and image classification. It is considered as the state of the art algorithm used for Big Visual data analytics. When compared to other image classifiers, CNN has some learnable parameters so it is also considered as a machine learning tool.

Let us see why CNNs are better,

(i) Easier and better training: In general, training images using a neural network (NN) is much simpler and easier than training using other traditional methods. And furthermore, training using NNs leads to requirement of more number of parameters and that in turn increases the time consumption proportionally. In CNNs number of parameters required are much lesser than the general NNs. Thereby decreases the amount of time required to train very large data sets. And also, if at all a NN was designed with similar complexity as that of CNNs in the network and parameter size, it would give poor classification accuracy over the CNN. Hence CNNs are preferred.

(ii) Memory management: In comparison to any other image classification techniques that is put to use for CV applications (they have poorer memory management in them and require more memory to store the feature vectors), CNNs require much lesser than them in the spatial-spectral domain (FC layer). Example: Say a 10 x 10 image with 10 features extracted requires 1000 memory blocks whereas in a CNN, the same image will require

much lesser as it would have automatically reduced the feature dimensions through its built-in feature selection algorithms.

(iii) Robustness: CNNs are robust/rugged to distortions present in the image as it is shift invariant. Example, difference in illumination, vertical shifts, horizontal shifts, shears etc. Shift Invariant because it uses the same weight across its configuration. This usage of same weight configuration over the entire space requires more training samples to cover possible variations.

Part d:

Loss function:

A loss function is defined as a function that maps an event to a real number representing the 'cost' or 'loss' associated with it. Our main goal is to minimize this loss function. As seen in the regularization part, there is loss due to penalizing the neurons with probability 'p' to 0 and loss in data i.e. the difference between the predicted model and ground truth. They encompass the loss function for the given CNN. In general, we can use this loss function as a guide in training our neural network. For a classification problem, we can use mean squared error (MSE or L2) and cross-entropy to compute the loss function, in a regression problem we can use L1 and in object detection focal loss is used to compute. An example expression is, loss = ∑ (target - prediction).

Since our aim is to minimize the loss incurred in the training process, we'll have to deal with the derivative of the functions.

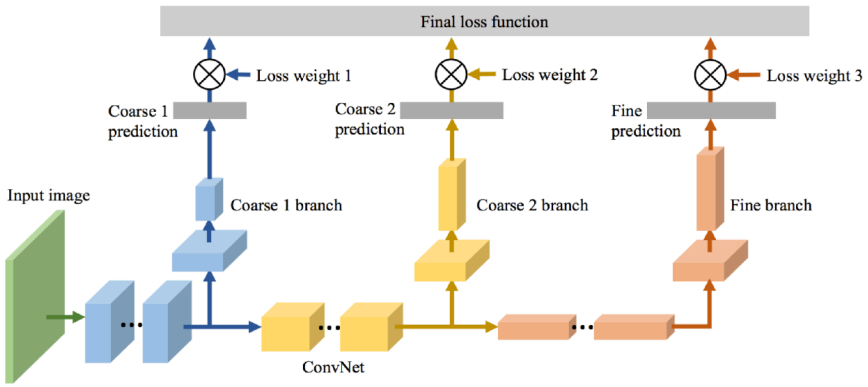


Figure 26: Loss Function

Backpropagation:

Backpropagation of errors or backpropagation is a commonly used method for training artificial neural networks which uses gradient descent as its optimization technique. Backpropagation gets its name 'back' as it calculates the error at the final stage and is distributed back through the network layers. This algorithm functions to update the weights attributed to the image in each layer. The forward propagation of a neuron is shown below, that is it gives the sum of weighted input and bias.

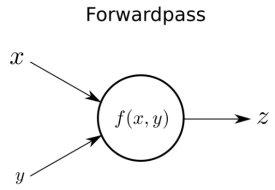


Figure 27: Forward Propagation

Say we have forwarded through the network layers and computed our output and the corresponding error in prediction using the loss function. This error gradient with respect to the weights is evaluated and is fed back through the network layers thereby updating the weights in each layer of the network. During forward propagation, itself it is taken care that all of the weight components play its role in the input elements, therefore during updating also all the weight components are updated. A simple expressive diagram for backward propagation is showed in the image.

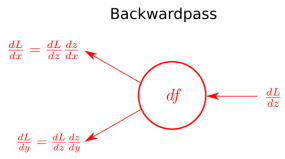


Figure 28: Backward propagation

This backpropagation step plays a very important role in training the neural network. As the training proceeds, the neurons are re-arranged accordingly such that it comprehends the input space in entirety. Since this backpropagation step requires to know beforehand the final output of each stage, it is considered as a supervised learning technique. Yet it is made use in unsupervised learning methods also.

Once the network is trained, when an unknown input image (maybe noisy or distorted too) is fed into the network, the neurons in the network's layer recognize the pattern associated with the input (compares with the trained data) and are activated accordingly. This shows the learning process of the neurons in the NNs.

On the whole, the mechanism is to compute the loss and modify the weights for the neural layers in order to learn or train the network.

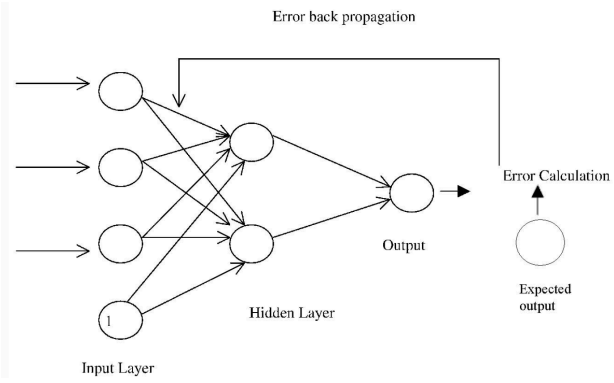


Figure 29: Backpropagation.

(b & c) Train the LeNet-5 on the MNIST Dataset and show improvisations:

1) The architecture and the mechanism of the LeNet 5 is explained briefly in the Approach and Procedure section of this question. When working with networks, there are many parameters to be taken care of in order to get a properly trained/learnt model. A few of the parameters that we have handled here are Filter weights, learning rate, drop out, batch size, epoch etc. In general, the data is normalized (0,1) / standardized (mean-0, variance-1) before fitting it as the input to the model. And also data shuffling is done prior to fitting the training data to prevent overfitting of data.

Let us see in detail the parameter initializations.

(i) *Filter weights*: By default, 'Glorot' is fixed for weight initialization. (Glorot is also called as Xavier). The ideology behind weight initialization is to ease the propagation in both forward and backward direction. The weights are deduced from the distribution of the activation function. Hence it can be of Normal/Gaussian or Uniform nature. The Variance of the chosen distribution function is calculate using the expression given below.

Var = 2 / (N in + N out)

Where N in and N out are the number of (#) neurons present in the previous and the next layer of the network. There are a lot of other initializers that can be used to initialize the weights. Namely, Zeros, Ones, Random (Normal or Uniform), Truncated (Normal or Uniform), Lecun Normal, our Glorot (Normal or Uniform) and many more.

(ii) *Learning rate*: Comprehensibly, Learning rate describes the time taken/required for the network to converge. Smaller is the learning rate, slower is the convergence for the given network, yet the network will converge to global minima. For larger learning rates, the network will converge soon and the probability that it converges to the global minima is less. This at times can lead to poorer classification performances.

(iii) *Epoch:* Epoch represents the number of times the training model will iterate over the data before it halts. Epoch and Learning rates go hand in hand. i.e. for smaller learning rates the #epoch should be high else the network wouldn't have converged to its global minima. This is shown in our observation.

(iv) *Batch size:* It represents the number of images (training data) used over single iteration in the gradient descent algorithm for finding the global minima. It impacts the learning model in both the aspects of 'time taken to converge' and 'over fitting'.

(v) Optimizer: We use Stochastic Gradient Descent (SGD) method as our optimization technique. A few other optimizers used are RMSprop, Adagrad, Adam, TFOptimizer etc.

(vi) Drop out: We would have discussed 'Drop out' in the regularization technique used to reduce the risk of over fitting. Its function is to force some neurons with probability 'p' to 0 so that they don't get activated for the next layer of the network. This way computational complexity is also reduced.

2 & 3) The classification accuracy, performance curves for both epoch wise and final model for the set parameters is shown in the discussion section of this problem. Refer to the pages from page 5 through 15. The hyper-parameters that are used are Batch Size, Epochs, Kernel Initializer, drop out, activation function, kernel window size, Optimizer used and the Learning rate.

Let us discuss the effect of each hyper parameter,

(I) LeNet 5 architecture: (Refer Table 1)

Convolutional layer 1 (C1) = 6 filters,

C2 = 16 filters,

activation function = Sigmoid,

Optimizer = Stochastic Gradient Descent.

Drop out = Not considered in LeNet 5 architecture.

The other hyper-parameters that are used are Batch Size, Epochs, Kernel Initializer, and Learning rate.

(i) Batch Size: Batch size defines the number of images to be taken into consideration for a single step of gradient descent. We observe from our different results that, as the batch size decreases, the classification accuracy also decreases accordingly. But the computation time taken for convergence decreases. i.e. the computation time taken to process one step of the gradient descent is lesser and hence the time taken for computing the entire process is lesser. And also by fact that since only fewer images are used in that training period, the model generalizes much lesser when compared to large batch size. Hence there is a drop in the classification accuracy.

(ii) Kernel Initializer or Weight Initializer: We know that choosing proper initial weights play a major role in the model's convergence but it doesn't affect the numerical calculation of the classification accuracy. The only problem that once can face is that if weights are not properly initialized, the system might not converge at all or will take a very long time to converge.

(iii) Epochs and Learning rate: These two parameters epochs and learning rate goes hand-in-hand. The learning rate represents the time taken for the model to converge and the fineness with which the model will be trained. Say we initialize sufficient number of epochs for each of the learning rate respectively, then the classification accuracy for the model with small learning rate

is higher than that of the model with larger learning rate. Yes, smaller is the learning rate, greater is the time taken for the model to learn.

Epoch: Epoch doesn't directly affect the classification accuracy for a given model, but indirectly it can harm the model from learning/training completely. Say we have a large learning rate and large number of epochs. There the model will converge quickly and for the excess number of epochs, the training accuracy will still remain that same as the model has already converged. But very large epochs can lead to overfitting of the training data [Since there is no drop out here. If there were, then epoch doesn't induce over fitting]. Say another instance where we have small learning rate and smaller number of epochs. Firstly, the convergence will take longer time and deficiency in the number of epochs will leave the model incompletely trained. Hence will result in poorer classification accuracy.

(II) ***CNN in general***: (Refer Table 2)

The effect of the hyper-parameters explained above apply here too. The hyper-parameters that were constant for the LeNet 5 are briefed here.

(i) Drop out: The concept of Drop out is to remove certain neurons from the layer. This improves the computation cost. It has to be taken care that drop out are used only in the final stages of the network, else there is high chance of neglecting important features when used in the initial layers. Greater is the drop out value, greater are the number of neurons neglected or forced from firing. Hence quicker is the training process. Say the drop out is 0.25, then all the neurons with values lesser than 0.25 is neglected from functioning and only those neurons whose values are greater than 0.25 fires. We can see from our observations that "ReLU" can have large drop out values and still function better to give good classification accuracy but when the activation function is "Sigmoid", we need to keep the drop out value low. As unlike ReLU, the sigmoid function starts to saturate for high drop-out value, which destroys the discriminative function of the network.

(ii) Activation Function: we have discussed a number of activation functions in the architecture of the CNNs. Weighing the pros and cons and observing the outputs for different parameter set, we can say that "ReLU" is more conveniently used owing to its simple computation and retentivity of the discriminative behavior of the neurons. We can also see that, using "ReLU" activation function, both the time taken for convergence and classification accuracy are improved.

(iii) Optimizer: For LeNet Stochastic Gradient Descent is the optimization technique used. But now a days most of the deep neural networks use Adam optimizer and we have also. This is because, SGD takes more time to converge when compared to Adam optimizer.

(iv) Kernel Size for filters in the convolutional layer: The size of the kernel affects the time taken for convergence and the richness in learning. Larger window sizes slow the convergence process but the learning process is rich and vice-versa for smaller windows.

The best classification accuracy obtained part (b) and its respective parameter setting is,

Batch Size: 196
Epochs: 200
Kernel/Weight Initializer: Glorot Normal
Filter size for Convolutional Layers: 6,16
Optimizer: SGD
Activation Function: Sigmoid
Pooling: Average pooling
Learning Rate: 0.3
Momentum: 0.7
Training Accuracy: 99.93
Test Accuracy: 99.22

In part (c), The best accuracy is obtained for the following parameter setting,

Batch Size: 128
Filter size in Convolutional Layer: 32,64
Epochs: 200
Kernel/Weight Initializer: Truncated Normal
Optimizer: Adam optimizer
Beta_1: 0.9
Beta_2: 0.999
Activation Function: ReLU
Pooling: Max pooling
Learning Rate: 0.1
Drop out: 0.5
Decay: 0
Momentum: 0.5
Training Accuracy: 99.63
Test Accuracy: 99.51

PROBLEM 2:

SAAK Transform and its application to the MNIST dataset:

AIM:

(a) Give a comparison study between SAAK Transform and CNN architecture.

(b) Apply SAAK Transform to MNIST dataset. Once the features are extracted, apply the F-Test score to reduce it to 1000 dimensions and further apply PCA to reduce the feature dimensions to 32,64 and 128 respectively.

(1) Now apply (i) Support Vector Machine (SVM) classifier and (ii) Random Forest (RF) classifier to the above 3 datasets and report the best classification accuracy.

(2) Compare the classification accuracy obtained using SAAK and CNN (LeNet-5 and others).

(c) Error analysis: Compare the classification error cases arising from the CNN and the SAAK transform, de-brief them and suggest ideas to overcome the errors/to improve the model.

ABSTRACT AND MOTIVATION:

Subspace Approximation with Augmented Kernels (SAAK) is an efficient and robust approach in recognizing the hand-written digits (example MNIST dataset). Like the Convolutional Neural Network (CNN), it is also network architecture with multiple layers i.e. the skeletal structure for both the CNNs and the SAAK transform is the same.

While discussing the performance evaluation of the CNNs in the last question, we witnessed many computational complexities involved in CNNs such as feature dimension handling, backpropagation, initialization of number of hyper parametrs and so on as a drawback/burden, which in SAAK, it is quite comprehended. There is so much similarities and differences between SAAK and CNNs, yet their function is the same giving us an oxymoronic feel in the understanding of their mechanisms to recognize the patterns. We still understand and implement the SAAK transform owing to its solidness in functioning and stability in the classification process. In short, with the understanding by going through the paper “On Data driven SAAK transform” we can put it in this way that CNNs function better but its scope is limited to its training data’s atmosphere only, whereas SAAK can bear with likely unprecedented data also. Hence the attribute “Robust” appears very often while describing SAAK.

Our objective here is to use the same dataset, MNIST, and apply SAAK transform to it to classify them and contrast out the features/facts between CNN and SAAK vividly. Let us see in detail about the SAAK transform below.

APPROACH AND PROCEDURE:

As explained briefly in the problem statement, the SAAK transform procedure can be divided into 3 parts, namely, (i) Generation of SAAK coefficients, (ii) SAAK coefficient selection and dimension reduction and (iii) classification. Our SAAK is a 5 stage transform. Let us explain the procedure stage-wise.

- The images are loaded as input into the network (MNIST training data). [Training set = 59000, Validation set = 1000].
- Each of the image is split into 4 quadrants and for each of it, a 2x2 non-overlapping patch is considered to compute the variance in the data and all those with low variance is removed. [i.e. KLT is applied]
- The KLT obtained data is standardized to zero mean and unit variance. PCA is applied here to select the patches with important spectral components. This step gives us the transformation kernels (4 in number, but when PCA is applied we can select the top 3 or 2 or the best).
- These transformation kernels are then augmented using the Sign to Position (S/P) conversion, so that the transform is preserved and inverse function can be performed effortlessly.
- Therefore, making use of the generated kernels we can extract the SAAK features/coefficients.

Note: This process is repeated for every stage of SAAK transform. Here, For stage 1: window size = 16x16, stage 2: window size = 8x8, stage 3: window size = 4x4, stage 4: window size = 2x2 and for stage 5: window size = 1x1. So, the number of feature maps producible at each stage is 4,16,64,256,1024 respectively. For here we will be selecting the top 3,4,7,6 and 8 feature maps based by application of PCA.

- Now since the SAAK coefficients are extracted, we will be having [(256 x ___) + (64 x ___) + (16 x __) + (4 x ___) + (1 x ___)] number of features. In our case we are having [(256 x 3) + (64 x 4) + (16 x 7) + (4 x 6) + (1 x 8) = 1168 feature dimensions. In our code, we have implemented for 4,12,10,20,16 best features from each stage. Hence, we will have [(256 x 4) + (64 x 12) + (16 x 10) + (4 x 20) + (1 x 16) = 2048 feature dimensions.
- This is then reduced to 1000 feature dimensions using the F-test score.
- This reduced data set with top 1000 features is further reduced to 128,64 and 32 dimensions using PCA.
- Once we have this feature reduced dataset, we will fit this data into the classifier to obtain the classification accuracy rate.
- The Classifiers used for predicting the accuracy scores are the Support Vector Machine (SVM) and the Random Forests (RF) classifiers.

EXPERIMENTAL RESULT:

```
Shapes: (10000, 2048) (10000,) (10000, 2048) (10000,)
Accuracy using 128 D features: 0.9564
Accuracy using 64 D features: 0.9662
Accuracy using 32 D features: 0.9685
guest-wireless-upc-1601-10-120-018-076:Saak-tensorflow-master rickerish_nah$
```

Figure 30: Training on 10000 images and test = default, 10000 images. The output values are not shown in %.

```
In [11]: runfile('/Users/rickerish_nah/D
rickerish_nah/Documents/trial/Saak-tenso
Size:
 TRAIN: 59000
TEST: 10000
Using SVM
using 128 D reduced features
Train accuracy: 0.9800847457627119
Test Accuracy : 0.976
using 64 D reduced features
Train accuracy: 0.9887457627118644
Test Accuracy: 0.9815
using 32 D reduced features
Train accuracy: 0.9922881355932204
Test Accuracy: 0.9832
Using RF
using 128 D reduced features
Train accuracy: 0.9992033898305085
Test Accuracy : 0.9057
using 64 D reduced features
Train accuracy: 0.9989830508474576
Test Accuracy: 0.928
using 32 D reduced features
Train accuracy: 0.998864406779661
Test Accuracy: 0.9325
```

```
Size:
TRAIN: 50000
TEST: 19000
Using SVM
using 128 D reduced features
Train accuracy: 0.97892
Test Accuracy : 0.9754736842105263
using 64 D reduced features
Train accuracy: 0.98816
Test Accuracy: 0.9808947368421053
using 32 D reduced features
Train accuracy: 0.9921
Test Accuracy: 0.9833157894736843
Using RF
using 128 D reduced features
Train accuracy: 0.99894
Test Accuracy : 0.903578947368421
using 64 D reduced features
Train accuracy: 0.99916
Test Accuracy: 0.9234736842105263
using 32 D reduced features
Train accuracy: 0.99906
Test Accuracy: 0.9351052631578948
```

Figure 31: Train and Test accuracies post F-Test and PCA.

```
Size:
 TRAIN: 40000
TEST: 29000
Using SVM
using 128 D reduced features
Train accuracy: 0.979075
Test Accuracy : 0.9703793103448276
using 64 D reduced features
Train accuracy: 0.987975
Test Accuracy: 0.9767586206896551
using 32 D reduced features
Train accuracy: 0.9923
Test Accuracy: 0.980448275862069
Using RF
using 128 D reduced features
Train accuracy: 0.9991
Test Accuracy : 0.8972413793103449
using 64 D reduced features
Train accuracy: 0.99865
Test Accuracy: 0.9134137931034483
using 32 D reduced features
Train accuracy: 0.99925
Test Accuracy: 0.9242758620689655
```

Figure 32: Train and Test accuracies post F-Test and PCA.

```
Size:
 TRAIN: 59000
TEST: 10000
Using SVM
Accuracy using 128 D reduced features: 0.9759
Accuracy using 64 D reduced features: 0.9818
Accuracy using 32 D reduced features: 0.9832
Using RF
Accuracy using 128 D reduced features: 0.9083
Accuracy using 64 D reduced features: 0.9197
Accuracy using 32 D reduced features: 0.9302
Size:
TRAIN: 50000
TEST: 19000
Using SVM
Accuracy using 128 D reduced features: 0.9755789473684211
Accuracy using 64 D reduced features: 0.9806842105263158
Accuracy using 32 D reduced features: 0.9831052631578947
Using RF
Accuracy using 128 D reduced features: 0.9033157894736842
Accuracy using 64 D reduced features: 0.9238947368421052
Accuracy using 32 D reduced features: 0.9343157894736842
Size:
 TRAIN: 40000
TEST: 29000
Using SVM
Accuracy using 128 D reduced features: 0.9703103448275862
Accuracy using 64 D reduced features: 0.9766896551724138
Accuracy using 32 D reduced features: 0.9805172413793104
Using RF
Accuracy using 128 D reduced features: 0.9000344827586206
Accuracy using 64 D reduced features: 0.9146206896551724
Accuracy using 32 D reduced features: 0.9243793103448276
```

Figure 33: Final Test accuracy post F-Test and PCA.

The best accuracy score is obtained for Features reduced to 32 dimensions post PCA.

Respective Training and Testing accuracies are 99.23 and 98.32 respectively

DISCUSSION:

(a) Similarities and Differences between CNN and SAAK:

Let us see their similarities,

- Both of them, CNN and SAAK makes use of a number of convolutional layers in their architecture to produce feature maps.
- For firing the neurons to the next layer, ReLU activation function is used.
- Both of them represent the extracted features in spatial-spectral domain.

The differences are,

(1) *Architecture and Building Block*: The CNN comprises of a number of convolutional layers followed by pooling layers and then the fully connected layers. The convolutional layer makes use of number of random kernels of size N x N, where N is an odd number to extract the features from the image. This then is passed through the pooling layer that can be either max pool or average pool for feature dimension reduction. Once the image passes through a number of convolutional layer and pooling response layers, we will have a spatial-spectral domain representation of the image [output from the convolutional layer] completely converted into its spectral domain. This layer is called as the fully connected layer. This at times resembles the function of Multi-Layer Perceptron (MLP). This layer is then densed down to match the number of classes present in the given data set. Once the network reaches the final output stage, it has to calculate the loss or error produced in the model and then this loss is

back propagated to modify the filter weights to obtain a stable learning model. This itself would have given us a clear intuition on CNN that it is quite complex in functioning and requires more computational time to process the data. Especially with the mandatory back propagation that has to performed. And also, it is to be noted that any change in the number of class labels (perturbation) put forth for testing would adversely impact the performance of the developed model. So, every time where there is change in the input data set's structure, the model has to be retrained and then only will have to be tested for performance. Hence CNNs are time consuming and give poorer results to any real-time modifications made in the system. This way CNNs can be loosely attributed to be an Ad-Hoc classification method.

Meanwhile in SAAK, it also contains convolutional layers, but the kernels or filters applied are obtained using the Karhunen-Loeve-Transform (KLT) and are of size N x N, where N is an even number [16,8,4,2] (odd [3,5] in CNN). Once the feature maps are obtained, Principal Component Analysis (PCA) is applied to reduce the feature dimensions (which is not done in CNN, hence leads to expensive computation owing to very large feature dimension). At the end of each stage of SAAK transform the feature map's size is reduced by half, hence number of convolutional layers required to attain the fully connected layer is lesser.

In the CNNs despite visualizing/observing filter responses using scatter networks, tensor analysis, relevance propagation, Taylor Decomposition etc., the base theory is lacking. In contrast to CNNs, the kernels in SAAK are computed using KLT. Hence, they can be fully explained mathematically (Linear algebra and statistics). In today's world, the CNN structures are growing very complex and so it has become intractable to generalize its behavior mathematically.

(2) *Determination of Filter weights and Feature extraction and handling:* The fundamental difference in determining the filter weights in the convolutional layer is the learning methodology implemented. In CNNs the weight is randomly initialized at first and based on the loss or error incurred (which makes use of both data and its label) the weights are updated through back propagation. For this update to take place properly, the objective function is optimized over a number of iterations (which is typically large in number). In SAAK, the filter weights are mathematically computed using KLT at each stage of the network. Since it is mathematically computed, SAAK doesn't require data labels and back propagation for weight update. Hence SAAK is computationally economical. Features/ Feature maps are generated using the obtain kernels/ filter weights. In SAAK, at the end of each stage PCA is performed and hence the feature (SAAK coefficient) dimension is kept under control. Since PCA is applied we tend to retain higher discriminating feature for the given class. In CNN, though pooling is done to reduce spatial dimension, feature dimensions is not handled.

(3) *Inverse Transform:* For CNN, nowhere it is mentioned on it having an inverse. i.e. CNNs cannot synthesize images. For that, two approaches have been determined namely (i) Generative Adversarial Model (GAN) and Variational AutoEncoder (VAE). Whereas for SAAK the KLT performed while computing weights [orthonormal in nature] is invertible. Hence by default due to its structure, it can produce/synthesize images directly. i.e Input image is converted to SAAK coefficient in the SAAK transform. As KLT is invertible, using the SAAK coefficient we can generate the image more likely similar.

NOTE: In CNN, after every convolutional layer the image size only reduces by a fraction depending on the filter window size and hence pooling is done for spatial reduction. But in SAAK, the KLT itself aids in spatial reduction.

(b) SAAK transform was applied and f-test scores were used to reduce the feature dimension to 1000 and then PCA was applied for further feature dimension reduction. Then the corresponding reduced feature data set is used by the classifiers SVM and RF to compute their respective classification accuracies.

The classification accuracy on Training and Test dataset using SVM and RF is tabulated below. Refer Table 3.

TABLE 3

S.No	Training Data size	Reduced Dimension using PCA	Training Accuracy	Testing Accuracy	Classifier
1	59000	128	98.00	97.59	SVM
2	59000	64	98.88	98.18	SVM
3	59000	32	99.22	98.32	SVM
4	59000	128	99.92	90.83	RF
5	59000	64	99.89	91.97	RF
6	59000	32	99.88	93.02	RF
7	50000	128	97.89	97.55	SVM
8	50000	64	98.81	98.06	SVM
9	50000	32	99.21	98.31	SVM
10	50000	128	99.89	90.33	RF
11	50000	64	99.91	92.38	RF
12	50000	32	99.90	93.43	RF
13	40000	128	97.05	97.03	SVM
14	40000	64	98.79	97.66	SVM
15	40000	32	99.23	98.05	SVM
16	40000	128	99.91	90.00	RF
17	40000	64	99.86	91.46	RF
18	40000	32	99.92	92.43	RF

PCA plays a very important role in feature dimension reduction. We can see that for different reduced feature data set we get different classification accuracies. i.e. for 32 dimensions reduced features we observe the highest classification accuracy over the 64 and 128 dimensions reduced.

The best classification accuracy obtained is 98.32% for 32 dimensions reduced by PCA and classified by SVM.

It can be seen from the above table that the *training accuracies are higher for RF* classifiers than the SVM classifiers. While the *Test classification accuracy for SVM* is greater than that of RF.

When comparing the difference between the Training and classification accuracy, the difference is greater for RF than for SVM. For SVM the difference is lesser than 1 {Diff < 1} and for the RF its greater than 1 and some cases it more or less equal to 10. Hence the overfitting issue comes into our mind. The reason for this difference could be the number of trees being used in the RF classifier. With lesser number of tress, we won't be able to tune to get the apt/desired accuracy. As the number of trees increases, we get better training and test accuracy.

But we also have to keep in mind that if we increase the number of trees in the RF, the test classification accuracy falls/ starts dropping. This could be a sign of overfitting the data also, although not sure if it is.

(c) Error Analysis:

The confusion matrices obtained for the best classification accuracies in CNN and SAAK are given below [for constant class = 10 for both test and train].

```
[[ 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]]
```

Figure: Confusion Matrix produced for Best classification accuracy in CNN

```
CONFUSION MATRIX: 32 D REDUCED AND CLASSIFIED USING SVM (TRAIN=59000)
[[ 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: Confusion Matrix produced for Best classification accuracy in SAAK

From the confusion matrixes observed above, we can say that (for same number of classes in the test dataset as in the training data) CNN has better classification accuracy than the SAAK. Looking at the diagonal elements of these confusion matrices, CNN's confusion matrix has higher count than the SAAK's. This can be used as a base for the above argument.

But based on the understanding of the mechanism of both CNN and SAAK, for a test data set with different number of classes, the CNN will have much poorer classification accuracy when compared to SAAK. Through this, we can deduce the fact that,

(i) For the testing data set having same number of class as the training data, CNN performs better. (ii) For the test data set having different number of classes as the training data, SAAK performs better. Hence SAAK is more stable and robust in operation and can be used in many unprecedented real-time operations, whereas the CNNs perform better only for the given environment.

As long as improvements go, we see in both the cases for CNN and SAAK, we use PCA or F-Test along with PCA. If there are other functions/methods that could give us better feature dimensionality reduction, we could make use of that. Then SAAK can be used for very high dimensional data as well, provided the dimension reduction takes place in a hierarchical manner. In CNN, what we can try is that, we can implement the S/P and P/S transform to make the network invert friendly.On the whole, more importance should be given to feature selection for feature dimension reduction.

LINKS and REFERENCES:

[1] https://sukhbinder.wordpress.com/2014/09/11/karhunen-loeve-transform-in-python/

[2] https://mxnet.incubator.apache.org/tutorials/python/mnist.html

[3] https://keras.io

[4] https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

[5] http://cs231n.github.io/convolutional-networks/

[6] https://arxiv.org/pdf/1710.10714.pdf

[7] https://arxiv.org/pdf/1710.04176.pdf

[8] https://github.com/morningsyj/Saak-tensorflow