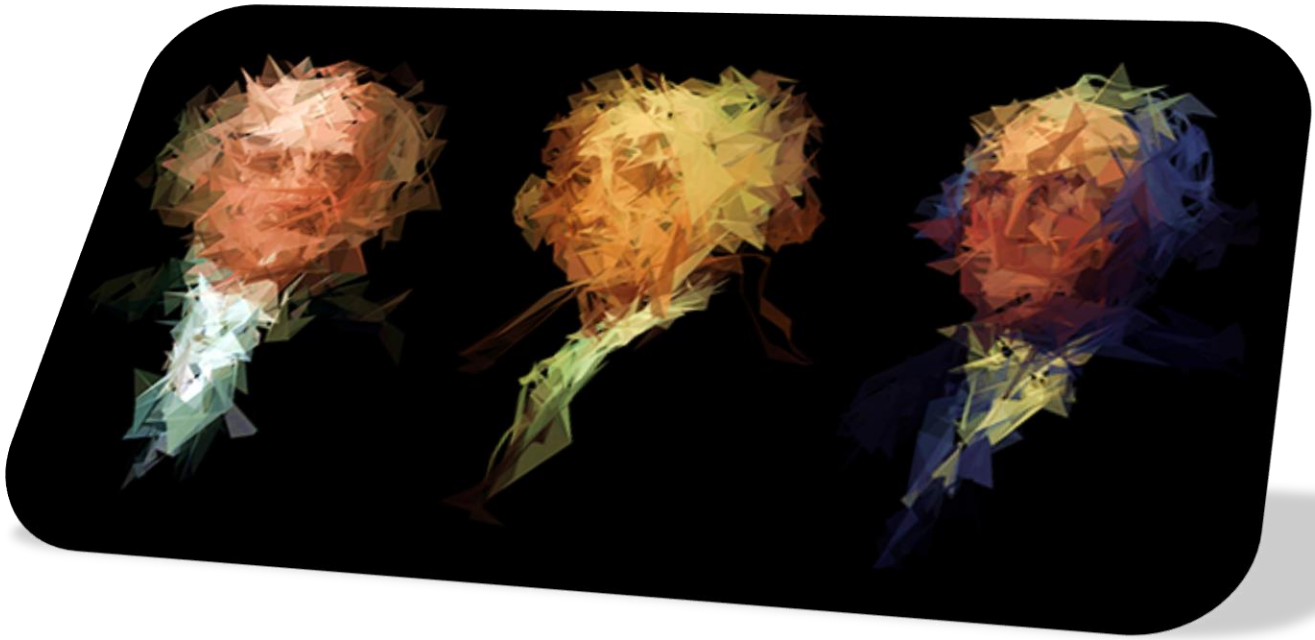
EE 569
HOMEWORK #4



SHIVA SHANMUGAPPA PATRE

spatre@usc.edu

1. Problem 1

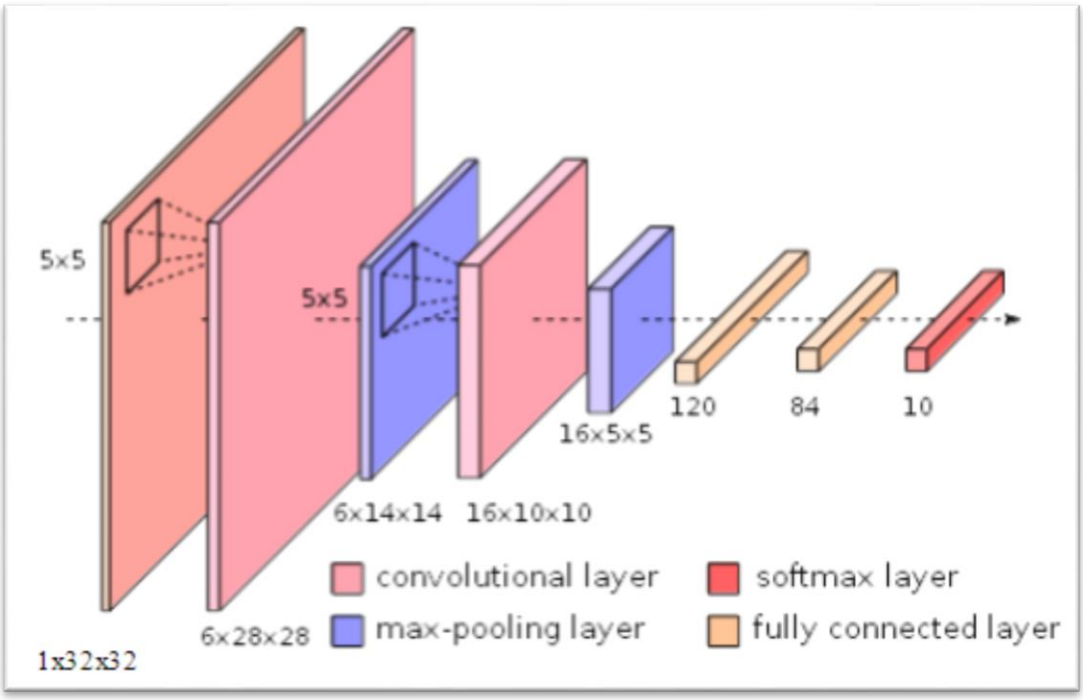
1.1 Motivation

As they say, Computer Vision is like the human eye, CNN or the Neural network is compared to the human brain. The most complex of all the human body parts. With the availability of big data and enormous computing resources, it is now possible to realize the idea of neural networks. Neural network is a way to model the data set for incorporating a form of intelligence in machines. Neural network is based on a large collection of neurons. These neurons are inspired from the working of the human brain. These neurons connect in a certain fashion which depends on the architecture of the network. Providing enormous data will help this network to learn from its intricacies and memorize the underlying pattern of the provided data.

The director of Facebook AI research LeCun wrote a paper on handwritten digit recognition that became famous as it was proven to be an accurate model. Since then the architecture of the neural networks were tweaked and applied to big datasets like ImageNet and CIFAR. Massive efforts have been put forward by researchers to optimize and generalize the networks. In this problem we shall make an attempt to solve and investigate handwritten character recognition using CNN.

1.2 Discussion

1.2.1 CNN Architecture and Training



Convolution Layer

Convolution layer from the LeNet-5 structure given to us in the question, all the components that are listed above are visible. Let us check the function of each of them in detail. Firstly, we have been given 60,000 training images that are divided into 10 different categories. One of the primary operations that are done in convolutional neural networks is the legendary act of 'convolution'. We describe each neuron as a filter or a kernel that has the same depth as the image. Each of the given images have a dimension of 32x32x1. Every filter that is used to conduct this convolution operation has a dimension of 5x5x1. Convolution that is inner dot product is taken and a value is obtained. The output of such an operation is a convoluted image that has dimensions 28x28x1. So basically, in a convolution layer, we apply some filters to get a convoluted image of a smaller size (if no padding is done). The convolution layer provides a receptive field which is basically an area equal to the filter size, for example, 5x5 in this case, and a number of such filters are used. For n number of filters of dimensions 5x5x1, we get n output 28x28x1 maps which are also called feature maps or activation maps.

It is essential to understand the function of each of these filters. Every filter that is used, corresponds to capturing certain property. For example, edges, curves, certain texture, etc. These are the simplest characteristics of the objects. The corresponding feature map is 'activated' at the locations where that feature is present. Therefore, the more the number of filters at the conv layer, the more is the depth of the activation map and therefore, more characteristics of the input image are captured.

Along with the depth, there are two more parameters that contribute to the size of the feature map. Stride is the measure by which we move the filter across the conv layer. This means that, if the stride is one, after finding the convolution sum at the first receptive field, we move the filter by one pixel to again find the convolution sum. If the stride is two, we move the filter by two pixels, and so on. A greater stride results in smaller feature maps. Padding is another parameter that decides the output size of the conv layer. When we zero pad the conv layer, the output layer would be the same size as the input layer and if we don't zero pad it, it is of a smaller size. In theory, several reasons are given why padding should be preserved. It's easier to design networks if we preserve height and width and don't have to worry too much about tensor dimensions when going from one layer to another because dimensions will just "work". It allows us to design deeper networks. Without

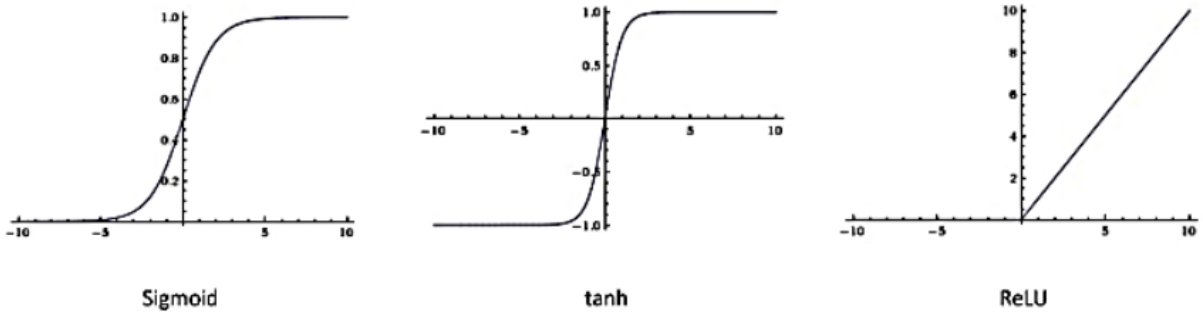
padding, reduction in volume size would reduce too quickly. Padding improves performance by keeping information at the borders.

This sums up the convolution layer. Usually we have several convolutional layers. Different architectures have different number of layers. In the LeNet-5 that we are implementing, we have two convolutional layers. But there are couple more layers that lie between these two convolutional layers.

The Activation layer

This brings us to the next important layer called the 'non-linear layer or the 'activation layer'. This layer lies immediately after the convolutional layer. The idea behind introducing this layer is to have non-linearity in a system which otherwise just had linear operations like summation and multiplication. Traditionally, we used to have tanh or sigmoid functions to introduce non-linearity. But it has been found that the ReLU layers (Rectified Linear Units) work far better as the network is able to train faster. The ReLU function is given by

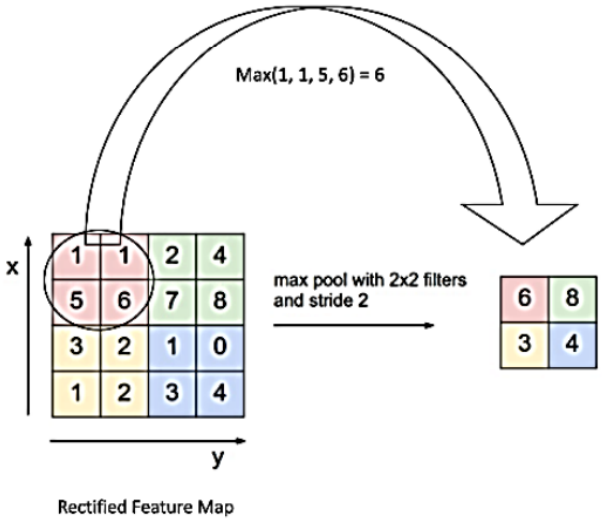
output= max(0,input)



Basically, the ReLU layer works by changing all the negative values to 0. There are several reasons why ReLU works better than other non-linear functions. It works really very fast when compared to others.

The Max Pooling layer

Next in store is the 'Pooling layer', which is also called by the name 'Max Pooling Layer'. Spatial pooling reduces the dimensionality of the activation map while preserving the most important features. This can be vaguely represented as below.



We can think of it in this way, that we only keep the features that are the 'most activated' ones. Hence, out of every four pixels, we take the maximum and this reduces the dimensionality by half in both the height and the width. If we were to think that the location of the 'important features' changes when we do this operation, it is important to understand that the relative location to the exact features is what is important. Two things are achieved by reducing the dimensionality. Firstly, the amounts of parameters or weights reduce by 75% thus, lessening the computation cost. The second is that, it controls overfitting. This term refers to when a model is so tuned to the training examples, that it is not able to generalize well, for the validation or the test sets. A symptom of overfitting is that we may get 100% or 99% for the training data, but only 50% for the testing data. So this needs to be resolved in reality to gain better results.

Dropout layer

The idea of dropout is very simplistic and it also pertains to the overfitting problem. This layer drops out random set of activations in that layer by setting them to zero in the forward pass. So, it compels the network to be redundant. This is because, the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. By doing this, we basically ensure, that the network is not too 'fitted' to the training data and therefore, it helps overcome the overfitting problem.

Fully Connected Layer

The CNN is mainly used for classification problems, which is what we are doing here. Classification is nothing but a regression problem, in which variables take in class labels. Fully connected layer is part of the hidden layer, with 120 neurons. It is connected from nearly 400 neurons from the previous layer. FC layers have the maximum receptive field. It

means every neuron has all the info about the input image which is supplied. They have the ReLU activations in their outputs to model the non-linearity. There is also another FC layer with 84 neurons. It is connected from the previous layer of 120 neurons. Even this layer is ready for ReLU activation. The FC layer is a classic multi layer perceptron that uses Softmax activation in the output layer. FC means that every neuron in the output layer is connected to every neuron in the input layer.

Softmax Function

Softmax function is usually known as the normalized exponential function. It is nothing but a generalization of the logistic function, that crushes a K dimensional vector of real random values to a K dimensional vector of real values in the range [0,1] that sums up to 1. Softmax layer trained on the hand written digits will output a separate probability for each of the ten digits and the probabilities will sum to 1. This is applied in the output layer which consists of 10 neurons which basically represent the 10 output labels. It is connected from the 84 neurons in the previous layer. This is cascaded with a log soft max layer that provides log probabilities of the output labels. The log softmax function is applied to each neuron and it is given by

$$s(x) = \log\left(\frac{e^x}{\sum_{i=1}^{N} e^i}\right)$$

The output of every neuron x is passed to this function. All the N neurons are computed with the same formula. During the training process, these values are sent to the negative log likelihood criterion to reduce entropy.

- What is the over-fitting issue in model learning?
  The basic concept of a CNN is there are training datasets and the testing datasets. The model learns from the training dataset and then uses that on the testing data to verify the results. Sometimes the training data has noise present and the model learns this noise as well. Because of this the results might not be very accurate. This is the classic case of overfitting, where the system learns the noise as well and doesn't work as desired with the testing data.
  There are a few approaches to overcome this issue of overfitting. One is to reduce the dimensionality. This is done by the pooling layer. Another method is the idea of Cross-Validation. This is done by splitting up the existing data in smaller samples and using that to train the data. We normally split the data we have into N groups and use all of them except one to train the model and the one to test it. Then each of this training set gets further divided and the same process repeats and we consider the

average. The dropout layer is also very useful with regard to overfitting. It drops out random set of activations by setting them to zero. This still doesn't affect the classification.

- Why CNNs work much better than other traditional methods in Computer Vision problems?
  There are numerous reasons as to why CNN is much better than other methods in CV. Some of them are as follows.
  - The feature extractors are not hand-designed unlike other CV algos like SVM, Random Forest etc which are used in image classification. The system learns by itself and cannot be generalized for every dataset.
  - CNNs are very closely related to the human visual system. Hence it will work similar to the human body and works the best.
  - CNN stores the results which it has learned. It consists of convolution layers, non-linearity layers, pooling layers and fully connected layers along with Backpropagation, that decides if the layers have the capability to perform with excellence, with respect to the image classification problem. This doesn't just have one single feature extractor but features of features of features etc based on the architecture. This really overcomes the semantic gap that all the other computer vision algorithms lack.
  - We observe better accuracy percentages with the CNN models when compared to other algorithms such as SVM, Random forest, Decision Trees, etc.
  - CNN is a supervised learning mechanism as we find the error to be backpropagated on the basis of the ground truth. The Backpropagation enables the network to be trained by itself and hence minimizing any effort from the supervising source to train it constantly.
  - Detection using CNN is insensitive to distortions like change in shape due to camera lens, different lightening conditions, different poses, horizontal and vertical shifts, etc. But, we see that CNNs are shift invariant. We can achieve shift invariance using fully connected layers.
  - In a CNN, since the number of parameters are drastically reduced, training time is proportionately reduced.

The loss function

The main function of Loss function lies in calculating the discrepancy between the CNN prediction and target. A typical loss function can be as follows.

$$z = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{x}^L\|^2$$

Where, t is the ground truth, and x^L is the prediction after the Lth layer. But this is more suitable for regression problem. Hence we don't use it here.For our image classification problem we use cross entropy loss function.

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

It maps all the predictions in a range of 0-1 thereby giving it a probability mass function form such that all the prediction values (for 10 different classes) sum up to 1. We convert the categorical variable t to a C dimensional vector t. When both the prediction as well as the ground truths have become probability mass functions, cross entropy finds the difference between them. Hence, we can minimize this cross entropy. Therefore, the loss function is basically modelled as the loss layer. This is a really popular choice and is also called Squared Hinge Loss.

The Backpropagation

This can be said to be the heart of the entire architecture. I had to understand numerous concepts to full grasp the concept of backpropagation.

- *Stochastic Gradient Descent*
  To give a gist of it in my own words, let us consider this scenario. We're climbing a blind folded. We need to reach the top of the mountain but we don't know the way to reach there and also when can we actually reach there as we are blind folded. So, one way to reach the top is by taking the steepest path as it is bound to lead us up, we get the most efficient way to reach the top of the mountain. To proceed with this, we look here and there and take the steepest path. This is repeated till we reach the top. The reason for this analogy is to understand how the cost function is minimized. For us, it computes the gradient using a few training examples, also known as a mini-batch.
- *The overall training*
  We initialize all the filters and weight with random values. The network takes the training image as the input, goes through all the forward propagation steps i.e. the conv layer, the ReLU layer, the pooling layer and finds the output probability for each class. Since the weights are given

random values, the output probabilities are also random. Then we find the total error at the output layer. This is found as: Total Error = ∑(target probability-output probability)^2 Then we use the backpropagation to calculate the gradients of the error with respect to all the weights that are present in the network. The above described gradient descent algorithm is what we use to update all the weight values to minimize the output error. We use the legendary chain rule technique for the back propagation to happen. So basically all the edge weights are randomly assigned values. Then for each input of the training samples, we get an output set of probabilities. This output is compared with the desired output or the ground truth that we already know. This error is transferred back to the previous layer. This error is noted and weights are adjusted accordingly. This entire process is done over and over till we get an output value that is below a pre-determined threshold.

1.2.2 Train LeNet-5 MNIST Dataset

The Pre-processing and random network initialization steps

Preprocessing: We first need to normalize the data. We can use a feature wise standard norm for this. We also need to subtract the mean across every feature of the data. Basically it is to center the data around the origin.

Weight initialization: I tried random initializer which uses minimized normal distribution with some SD and 0 mean. This works good generally, although there are better initializers like Xavier.

Activation Functions: ReLU, LeakReLU, sigmoid and tanh functions were tried. The best is obtained with ReLU. Although Leaky performs better sometimes.

Network Optimizer: This was kept at Adadelta throughout. Although it was tried with Adam as well.

Dropout was tried for 0.5 and 0.75. The best accuracy results were obtained for 0.75.

Learning Rate: This is basically how fast the system forgets old calculations and learns the new ones. Initially it was set to 0.1 and gradually decreased to get better accuracies and stuck at 0.01 as that gave the best accuracy.

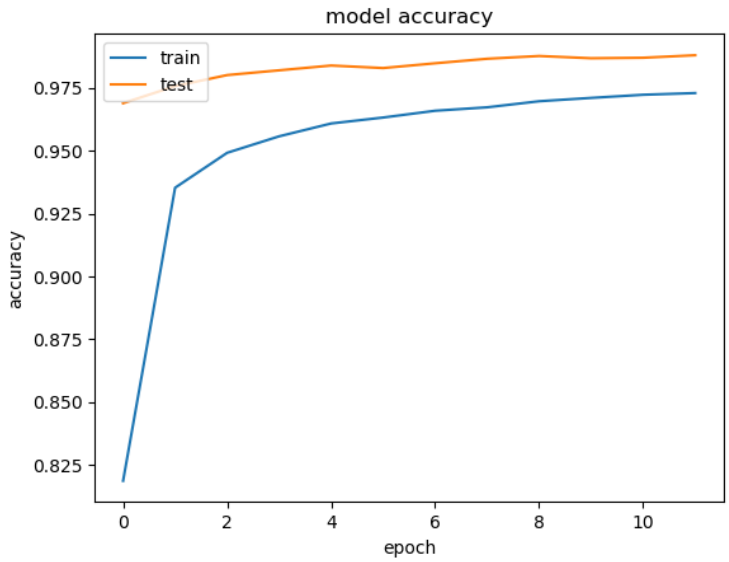
Loss Function: Categorical_Crossentropy is the Loss function that I used as it is best suited to the image classification problem

Batch Size: It was fixed at 128.

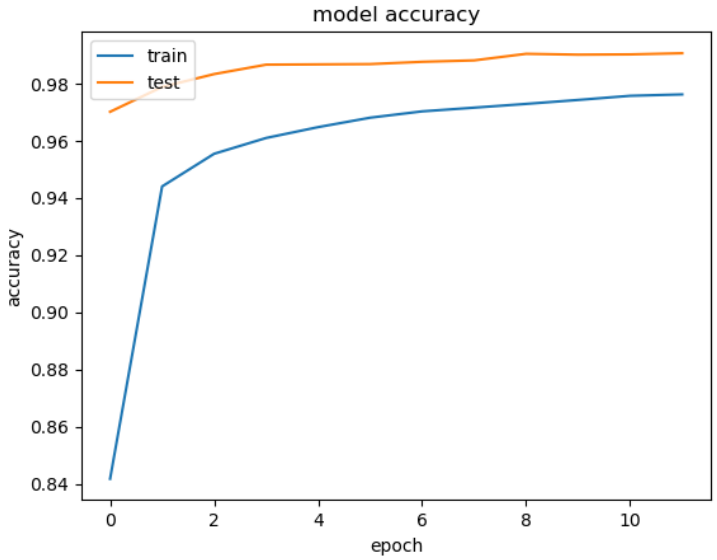
Few params like the batch_size and number of epochs affect the accuracy of the model. Smaller batch sizes affect the speed of the model and number of epochs affect the accuracy.
Data shuffling can also be done to prevent overfitting by randomly shuffling the training sets in batches. At every epoch the dataset could be shuffled.

Experimental Results

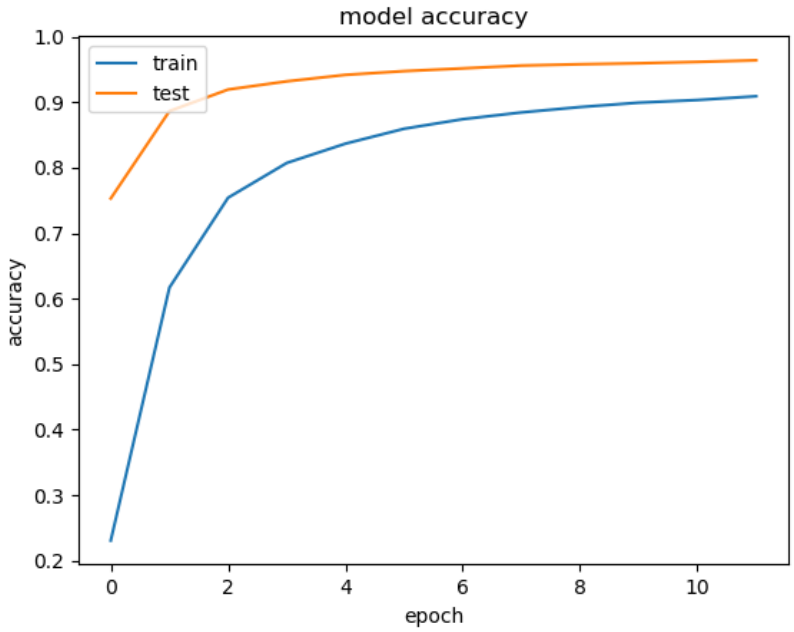
- For Batch_size = 128, Learning rate = 0.001 (Without Shuffling)
- Accuracy=97.81



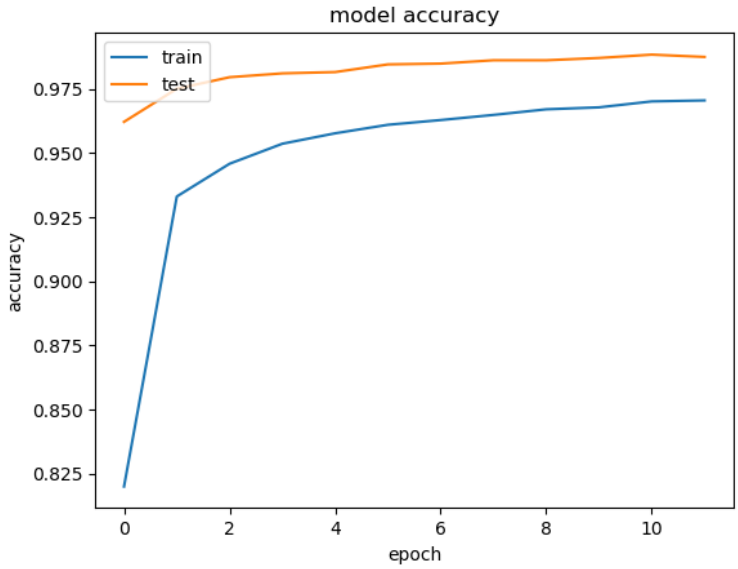
- For Batch_size = 128, Learning rate=0.001, after repeating a number of times (Without Shuffling)
- Accuracy=99.07



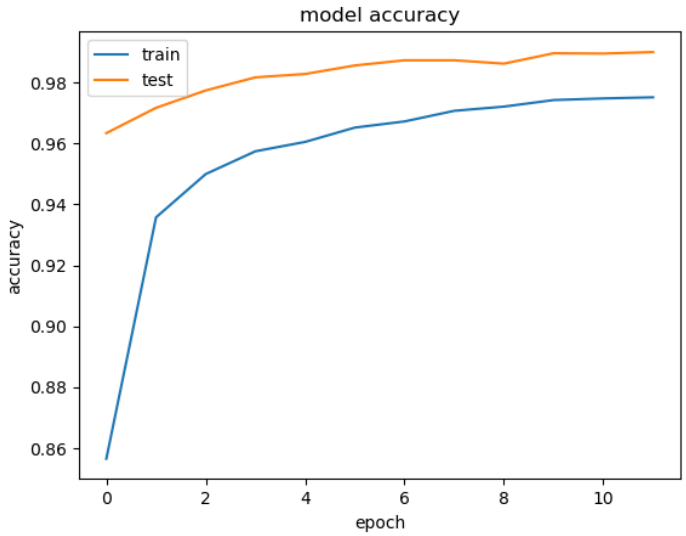
- For Batch_size = 128, Learning rate=0.06,(With Shuffling)
- Accuracy=96.41



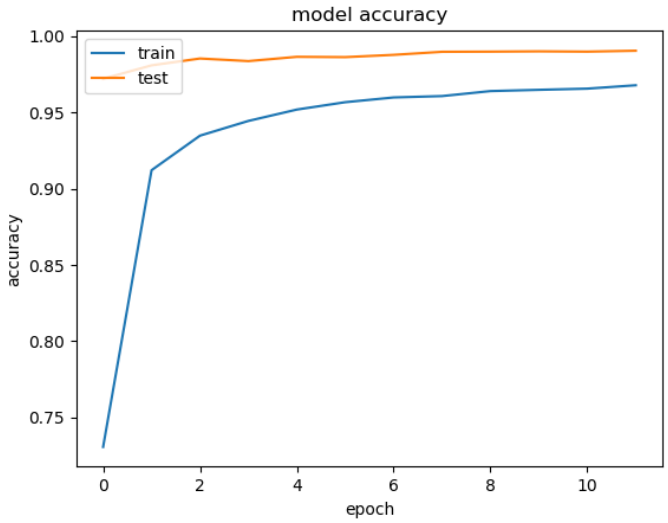
- For Batch_size = 128, Learning rate=0.001,(With Shuffling),Activation=Sigmoid
- Accuracy=98.75



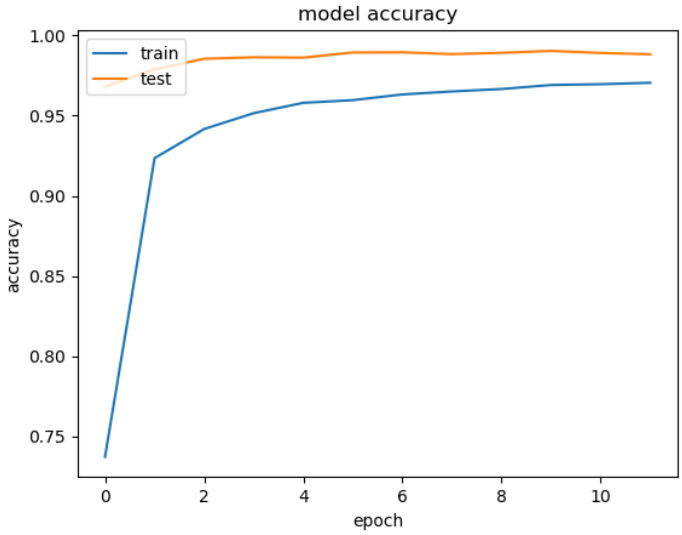
- For Batch_size = 128, Learning rate=0.001,(With Shuffling), Activation=tanh
- Accuracy=99.0



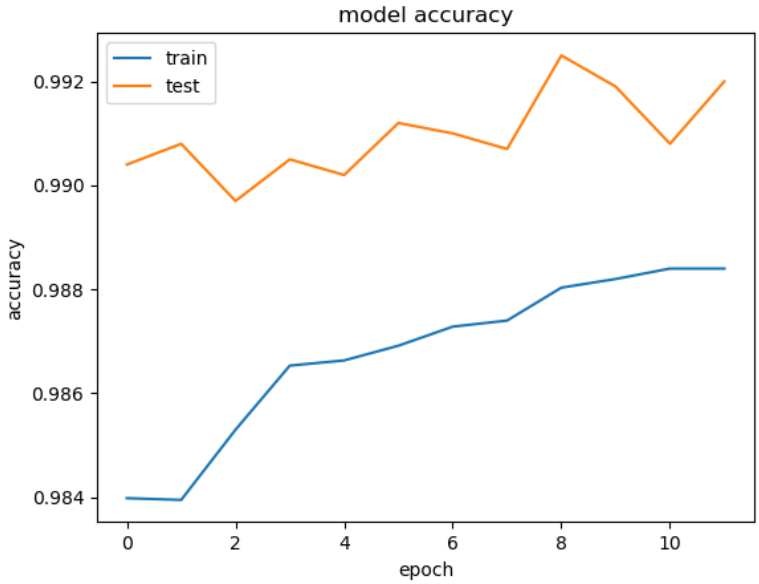
- For Batch_size = 128, Learning rate=0.001,(With Shuffling),
- Accuracy=99.22
- With extra convolution layer



- For Batch_size = 128, Learning rate=0.001,(With Shuffling),
- Accuracy=99.32
- With extra convolution layer, running for a number of times



- All values were taken as a random value and got a weird graph



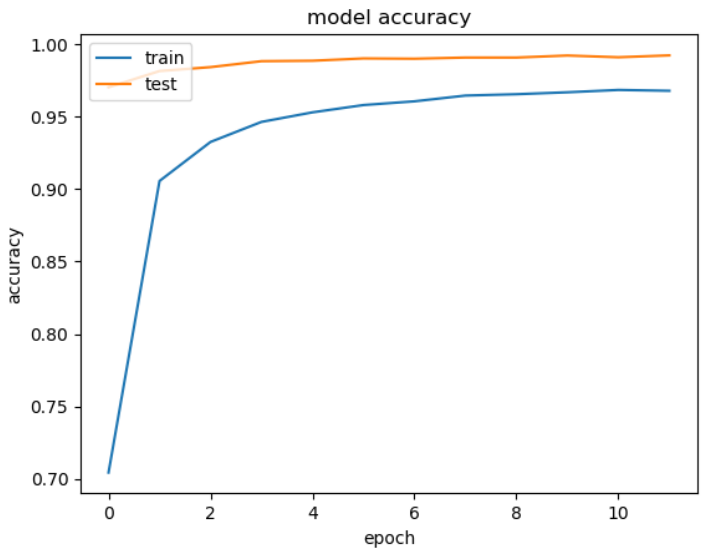
Discussion

Consider the above results, the following can be inferred from the results.
There will be a considerable impact if there is shuffling. Shuffling improves the accuracy and also with repeated runs there will be slight improvement in the accuracy as seen above. The batch size is kept constant and the learning rate is varied and the accuracy varies drastically as can be seen. The larger the learning rate, the accuracy gets worse, as there isn't much time to learn about the image. The learning rate should not be too less also. Ideal rate is found to be 0.001 and is fixed for most runs. As the batch size increases, the problem of overfitting arises and accuracy drops.

1.2.3 Improve the LeNet-5 for MNIST dataset

The default LeNet-5 architecture gives a good accuracy of around 98%, which is a brilliant result. Adding more and more fully connected layers will definitely help in improving the accuracy but we have to take care of over fitting issue too. I have used Learning rate = 0.001 and Batch size = 128 ,and added extra convolution layers. I achieved accuracy of 99.41%(shown below) which is I think is a drastic improvement . I had Shuffled the Data using Shuffle function. The deeper the network gets with more convolution layers and size of filters, there is a possibility of getting higher accuracy. As it takes a lot of time and with no GPU, I couldn't run more trials, else I would have tuned

hyper parameters to get accuracies of around 99.5% or more. The graph I obtained is as follows.



2. Problem 2

2.1 Motivation

Every great thing that has ever happened, all started with a dream. Be it the invention of the automobile, airplane, rockets etc. All experts were once beginners. Everyday, one or the other new thing surfaces and the world is taken by a storm. Just like the Apple made Nokia extinct, new things will make old things endangered. One such thing is the Saak transform, which is Prof. Jay Kuo's dream, which might one day take the CNN by a storm.

Saak stands for Subspace approximation with augmented kernels and is based on the inspiration derived from RECOS (Rectified Correlation on a Sphere). There are a few key steps in doing the Saak transform which are to build an optimal linear subspace approximation with the orthonormal basis using 2^{nd} order stats of input vectors. Then augmenting or attaching each o the transform kernel with its negative. This is followed by the application of ReLU to the output. KLT transform is used to make it happen here. To transform images of larger size, we need to cascade multiple Saak transforms. The multi stage Saak offers full spatial domain and full spectral domain representation. To form a feature vector, Saak co-efficients of higher discriminant power is chosen.

2.2 Discussion

2.2.1 Comparison between the Saak transform & the CNN architecture

As we know that Saak transforms is derived based on the RECOS, there are a few drawbacks with the RECOS transform. Beyond a certain threshold, the approximation error cannot be lowered. To make-do with the forward and inverse transform, it doesn't have any orthonormal transform kernel. The representation of signal is not that great in RECOS as the representation power is of a few main parts is lost. The above issues can be resolved by Saak as it uses KLT transform. The minimum approximation error can be reduced by changing the basis functions of the KLT Transform. Since Saak uses attaching or augmenting the negative vectors, the signal can be preserved.

Saak and CNN can be compared in the following ways. CNN is not that great when it comes to being robust, whereas Saak outperforms CNN when it comes to robustness. CNN is pretty bad at scaling, while Saak is pretty good when it comes to Scalability. CNN is extremely complex and its complexity is super high, whereas CNN is relatively less complex and isn't computationally expensive. In CNN, a lot of parameters have to be adjusted in the backpropagation whenever there is a new dataset. Saak doesn't need any such parameter adjustments. CNN follows end to end optimization, wherein the filter weights are updated using the data from the two ends of the network. Saak doesn't require labels or filter weights as it is an automatic weight selector. CNN is computationally expensive while Saak is much less expensive. CNN revolves around Backpropagation, while Saak doesn't need any Backpropagation. In CNN, if there is a change in the number of dataset or the number of classes, it needs to be retrained and this is again expensive. Saak on the other hand is less affected by the change in dataset and classes.

Now we know that Saak is made up of KLT transform in the first stage and Kernel Augmentation in the second. Consider a scenario wherein the input space dimension is in say, millions or even more, it would be tedious to perform KLT in a single stage. We may divide this input space into many smaller subspaces and using recursion, we can repeat the above process efficiently. Next Kernel augmentation is used to eliminate the rectification loss which arises when there are more transforms cascaded one after the other and a sign confusion occurs. ReLU is inserted to resolve this.

Every kernel is augmented with the negative vector and both the original and augmented kernels are used in Saak. Only one of the projected input vector on the +/- kernel pair, will go through ReLU operation and other doesn't. This facilitates inverse Saak transform and simplifies signal

representation issue which is faced by ReLU. The kernel augmentation and ReLU integration is nothing but sign-position conversion of the Saak co-efficients. All in all, I feel Saak has a great potential in the future and can be the CNN killer. Let's look forward to Prof Kuo's dream project come to life soon.

2.2.2 Application of Saak transform to MNIST dataset

We are given the MNIST dataset as the input with 60000 training data and 10000 testing data. The approach to be followed is as follows. We are given the link to GITHUB to find the Saak Transform implementation code. So we need to run the code to find Saak co-efficient generation which is given in the paper. Then we execute the command multi_stage_saak_trans() and the data in the variable will be of 1000 dimension and the number of features around 1999. Then we need to perform F-Test using f-Classif. Next we need to choose top 70-80% info from where we give the data and output using the function selectKBest. Then the dimensionality is reduced to 32,64 and 128 from 1000 using Principal component analysis. SVM and RF classifiers are used or integrated with this and the training and testing is done. We can follow the steps given in the question paper as described below and we'll get the results accordingly.

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 se lecting the important spectral components in the PCA matrix. - Augment transform kernels through the sign-to-position format conversion. - Repeat this process for each Saak transform stage. For your convenience, the number of important component in each stage (total 5 stages) are: 3, 4, 7, 6, 8.

Step 2: Saak coefficients selection and dimension reduction

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

Step 3: Utilize images of the same class label to collect features of training samples and compare the performance of the SVM and the RF classifiers with three feature dimensions as given in Step 2. (1) Report the best classification accuracy that you can achieve on the train and test sets for all 6 cases. (2) Compare the performance to the results in Problem 1 and briefly explain your observations.

RESULTS

Classification Accuracies of RF Classifier

Training Set

Dimensionality-Accuracy

32 - 96.18%

64 - 96.29%

128 - 96.23%

Testing Set

32 - 96.02%

64 - 96.15%

128 - 96.08%

Classification Accuracies of SVM Classifier

Training Set

Dimensionality-Accuracy

32 - 95.81%

64 - 96.34%

128 - 96.12%

Testing Set

32 - 95.24%

64 - 96.21%

128 - 95.81%

Observation

- As can be seen from the results above, the SVM classifier of dimension 64 has a better training and test accuracy when compared to RF classifier. For smaller datasets, SVM is a better bet and gives good results. If the data size is more and exceeds a few thousands, then RF is the king and gives the better results when compared to SVM.
- In comparison to the LeNet 5 basic architecture, this is better. But if we fine tune the parameters in LeNet-5, then we get amazing results of accuracies more than 99%. So we will require best tests to improvise the Saak transform.

2.2.3 Error Analysis

The only way to see which algorithm works better than the other is through trial and error. Based on the performances we can draw a conclusion. The accuracy can be improved by tackling data imbalance by subsampling the data and applying the algorithm to smaller and smaller groups and consolidate the results.

We can improve Saak transform by making it generalized by a combination of feature extraction and a classification model by building an end to end architecture. Using F measure, we selected the best features, but still this doesn't always fetch us the good results. We need to find more effective methods to extract important features from the features which are currently available. We can explore the area where there is a need for Saak inverse transform and compare the efficiency with RCNN and GAN.

To still improve classification accuracy in case of CNN, we can go for more complex and rigorous architectures like AlexNet and try to tune hyper parameters to improve the accuracy of the classification. In case of Saak Transformation, using RF classifier for more data and varying the number of Saak coefficients can produce better results. Both the architectures have their own pros and cons and one outweighs the other in different scenarios and we need to choose accordingly.

References

- Yan LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Muller, "Efficient BackProp".
- Glorot Xavier, "Understanding the difficulty of training deep feedforward neural networks ", 2010.
- Kaiming He, "Delving deep into rectifiers: Surpassing human level performance on ImageNet Classification", 2010.
- https://arxiv.org/pdf/1710.04176.pdf
- https://github.com/davidsonic/Saak-Transform
- http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html