EE 569: Digital Image Processing
Homework #6
Feedforward CNN Design and Its Application to the MNIST Dataset

By
Brinal Chimanlal Bheda
USC ID - 8470655102
bbheda@usc.edu
Spring 2019

Issue date: 04/08/2019
Submission Date: 04/28/2019

Problem 1: Understanding of feedforward-designed convolutional neural networks (FF-CNNs)

I. Abstract and Motivation

(Source: Chen, Y., Yang, Y., Wang, W., & Kuo, C. C. J. (2019). Ensembles of feedforward-designed convolutional neural networks. arXiv preprint arXiv:1901.02154)

(Source: Kuo, C. C. J., Zhang, M., Li, S., Duan, J., & Chen, Y. (2019). Interpretable convolutional neural networks via feedforward design. Journal of Visual Communication and Image Representation)

(Source: Discussion lecture dated 04/09/2019)

In this homework, I'm going to talk about the feed forward design CNN. It is also called as FF-CNN. In the previous homework, I implemented the LeNet-5 architecture using the back propagation to learn the networks and do some recognition tasks on the MNIST datasets and became familiar with the convolutional neural networks. However, in this homework, we do not need to use the back propagation, optimization to learn all the parameters in the CNN, use the feed forward design to learn all the parameters in the CNN.

(Source: Lectures by Prof Kuo dated March 2019)

In this part, we use a new way to do CNN, the one pass feedforward (FF) design interpretable CNN. The feedforward and back propagation are two different designs, different properties, behavior and choose weights different but same architecture. Using the feature extraction and bag of words ideas to do the classification. So get some features, using bag of words and then some classifier. The CNN isn't good enough due to following reasons.

- **State-of-the-art performance**
  - Yet, …
- **Non-convex optimization for over-parameterized systems**
  - Thousands/millions of model parameters determined by backpropagation (BP)
  - Mathematically intractable
- **Numerous tricks**
  - Empirical architecture choice
  - Random dropout
  - Data augmentation
- **Sensitivity to small perturbations**
  - Adversarial attacks

2

II. Approach and Procedure

(Source: [MNIST] http://yann.lecun.com/exdb/mnist/)

(Source: https://github.com/davidsonic/Interpretable_CNN)

(Source: Interpretable convolutional neural networks via feedforward design ppt)

Path to Interpretable CNN:

Use the same CNN architecture, but implement how to select weights and use the feed forward. Feed forward means go to layer 1, choose the parameter in layer 1 and create response, the output of layer 1, which becomes the input of layer 2, based on the statistics of layer 2 then decide the layer 2 parameters and use that to generate layer 2 response output, which becomes layer 3 input and so on. So with the input, study the statistics of the input and use it to determine the parameter with the current layer and give the output. This mathematics should be good to get a good performance. Another idea is to use k means clustering on the input, use the centroid to be the filter weight to find the match filter. Depending on the number of filters, choose the number of clusters. Do this k means from beginning to end 25%. The design is very different from the back propagation (BP) based design.

- **A new CNN design methodology**
  - One-pass feedforward (FF) design
  - Interpretability
  - Complementary reference design
- **Performance comparison between FF and BP designs**
  - Classification accuracies
  - Vulnerability against adversarial attacks
- **Connection between FF and BP designs**
  - Cross-entropy study at intermediate layers

Feed forward Design Methodology:

Now-a-days, all the data is high dimensional. In images, one pixel is one dimension. Say the input image is 32x32 RGB, then the input dimension will be 32x32x3 = 3072. The output dimension depends on how many classes are there. Say there are 10 classes, then have 10 dimensional vector. Middle part are cuboids of certain high dimension. Every processing in each layer is actually a transformation of high dimensional vector to the next set of high dimensional vector. The training helps to determine how to do the transformation. Along with this dimension reduction is also needed, along with some optimization.

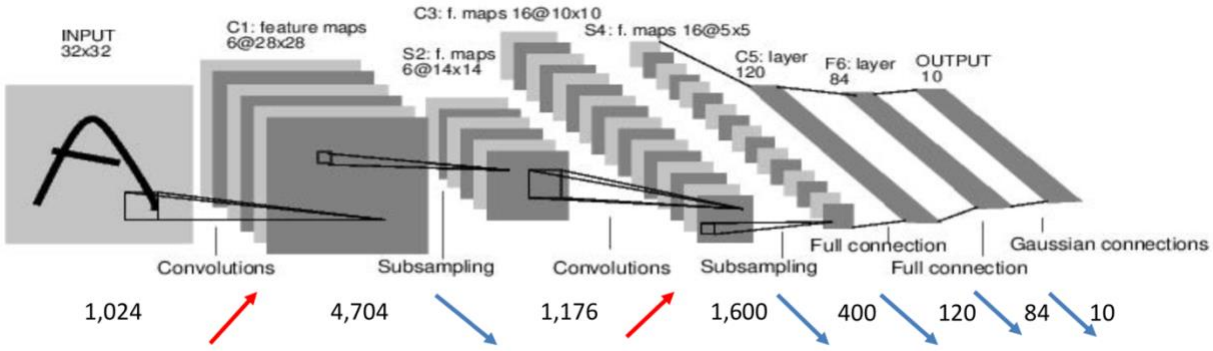
Vectors in high-dimensional spaces

- A unified framework of image representations, features and class labels
- Example: Classification of color images of size 32x32 to 10 classes
  - Input space of dimension 32x32x3=3,072
  - Output space of dimension 10
  - Intermediate vector spaces of various dimensions

FF design basically talk about the filter weight in the one pass version with the dimension reduction techniques. In the vector space, the data does through transformations to get the next response for each step and finally getting 10 decision vectors. Here in the process, we have some criteria: a) preserve as much as information/ fidelity, because in dimension reduction loss some data. It is a lossy process. But preserve the critical and important information. b) preserve the discriminant power of the dimensions i.e. keep/ enhance the discriminant ability of

remaining dimensions. So in all reduce the multiple high dimension vector but at the end keep the remaining dimension which has strong discriminant power.

- **Sequence of vector space transformations**
  - Input space ➟ output space
    - Dimension expansion (sometimes)
    - Dimension reduction (most time)



Input is 0 to 9. The output is 10D probability vector for the 10 digits. The first conv layer has the dimensions 28x28x6 from the input 32x32. Thus the dimension increases from 1024 to 4704, denoted by the red arrow above. Then implementing spatial sampling, dimension reduces to 1176 denoted by the blue arrow. Do another filtering where dimension increases again and further dimension is only reduced i.e. dimension expansion. This is because want to find the dimension which has strongest discriminant power. Spatial filtering is done, because it looks through a patch not a pixel, the filter tries to capture some pattern in the patch so do not use very small patch. Hence require multiple filters to capture multiple patterns. Use this filter and scan through the whole 28x28 image and see the response. If use larger patch in the first one, the use a smaller patch for the filter in the next layer. Perform dimension reduction which is from the input representation or feature to the output decision labels, along with keeping the discriminant power, using the PCA and max pooling, also doing the non-linear activation so as to increase the discriminant power.

Why dimension expansion?

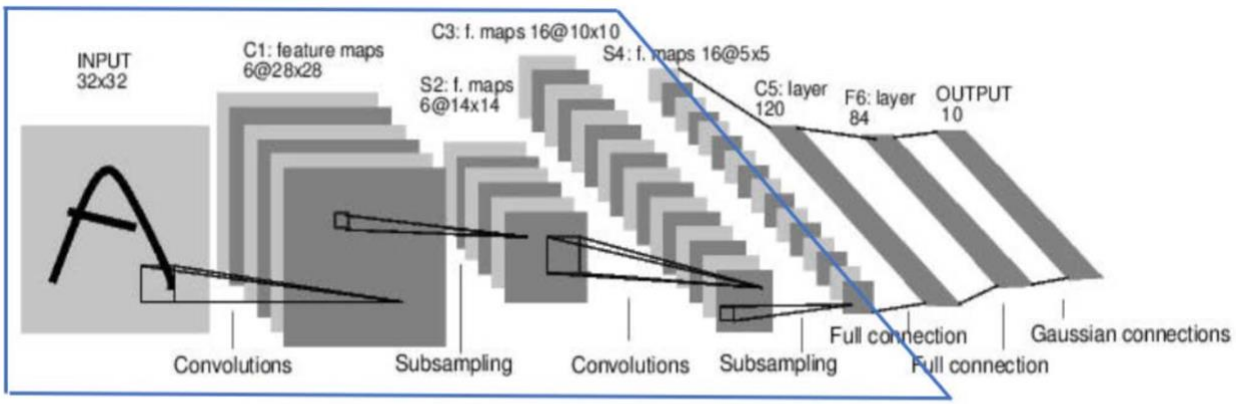
- To contain dimensions that carry stronger signature
  - Example: spatial-spectral filtering using a small stride

Why and how dimension reduction?

- (Why) bring input space to output space
  - Representations/features → decision labels
- (How) keep informative dimensions
  - Examples: PCA and max pooling
  - Example: Hierarchical decision process in FC layers
- (How) enhance dimension's discriminability
  - Example: nonlinear activation, etc.

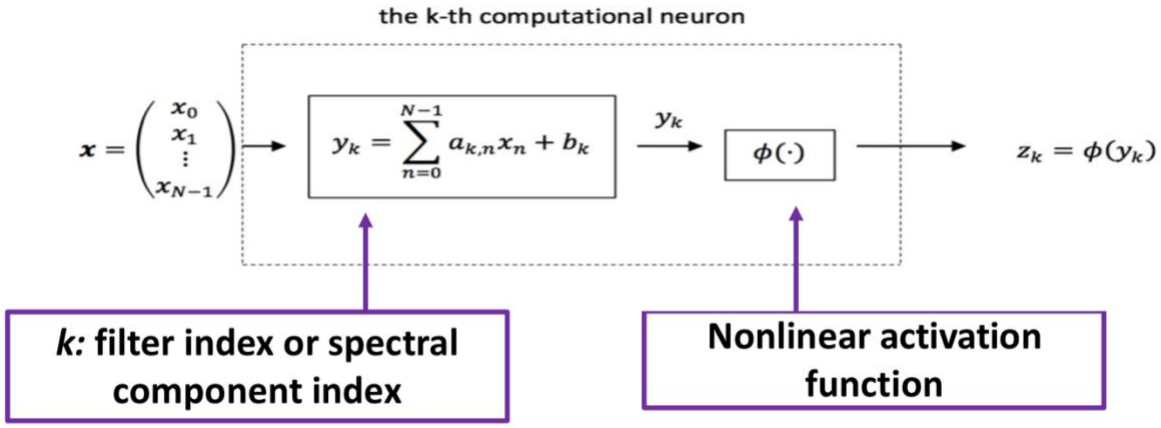
Part 1: Convolution layer

There are two layers, consisting of spatial filter, non-linear activation and pooling.



2 convolutional layers + 2 FC layers + 1 output layer

The convolutional filter is shown below:



Two challenges:

- Non-linear activation – difficult to analyze
- A multi-stage affine system is complex

For the interpretation of filter weights in the convolution layers:

1st viewpoint (training process, BP)

- Parameters to optimize in large nonlinear networks
- Backpropagation – SGD

2nd viewpoint (testing process)

- Filter weights are fixed (called anchor vectors)
- Inner product of input and filter weights -> matched filters
- k-means clustering

3rd viewpoint (testing process, FF)

- Bases (or kernels) for a linear space
- Subspace approximation

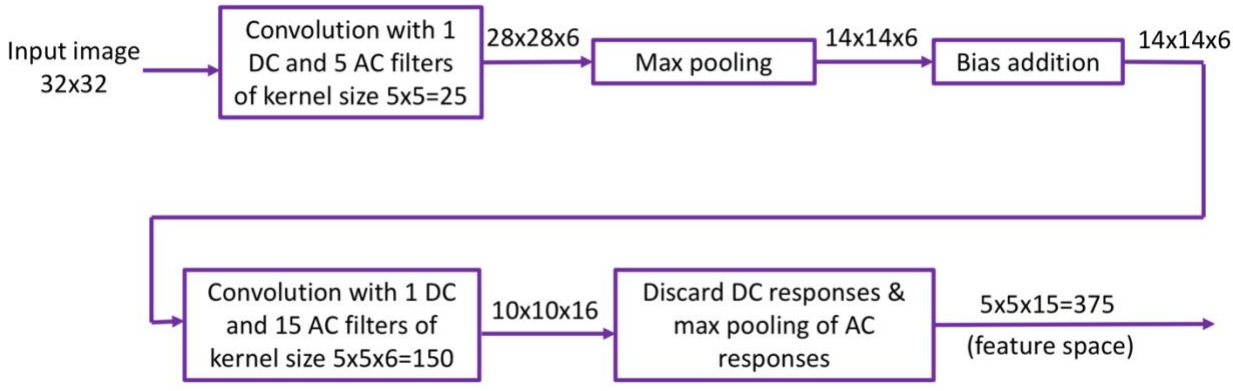
- **The sign confusion problem**
  - When two convolutional filters are in cascade, the system is not able to differentiate the following scenarios:
  - **Confusing Case #1**
    - A positive correlation followed by a positive outgoing filter weight
    - A negative correlation followed by a negative outgoing filter weight
  - **Confusing Case #2**
    - A positive correlation followed by a negative outgoing filter weight
    - A negative correlation followed by a positive outgoing filter weight
- **Solution**
  - Nonlinear activation is used as a constraint to block the 2nd case (i.e. a rectifier)

Non-linear activation used can be sigmoid, relu or leaky relu.
Spatial redundancy is controlled by the stride parameter.
Small stride → Overlapping windows → Dimension expansion

Spatial pooling → Maximum pooling: maximizing the correlation between inputs and a filter.
PCA → spectral pooling, max pooling (subsampling) → spatial pooling

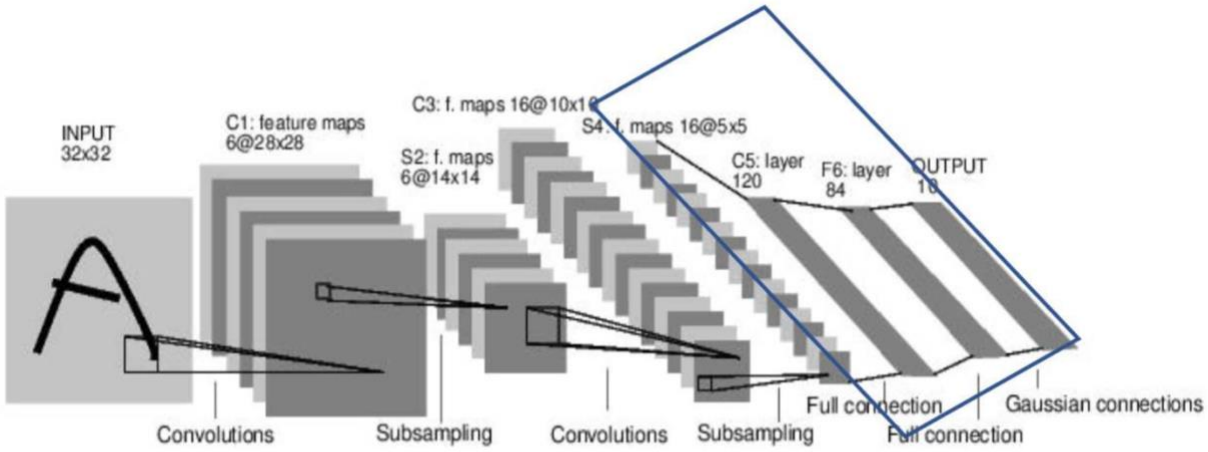
Limitation of one-stage filtering: Diversity increases rapidly with a larger window size
Power of two-stage filtering: 1st Conv Layer (patches of size 5x5): simple strokes, 2nd Conv Layer (patches of size 14x14): More discriminant as window sizes go larger

Block diagram of 2 Convolutional layers is shown below:



Part 2: Fully connected layer

There is one output layer and 2 hidden layers in the FC part.



2 FC layers (120D, 84D) + 1 output layer (10D)

For the interpretation of filter weights in the FC layers:

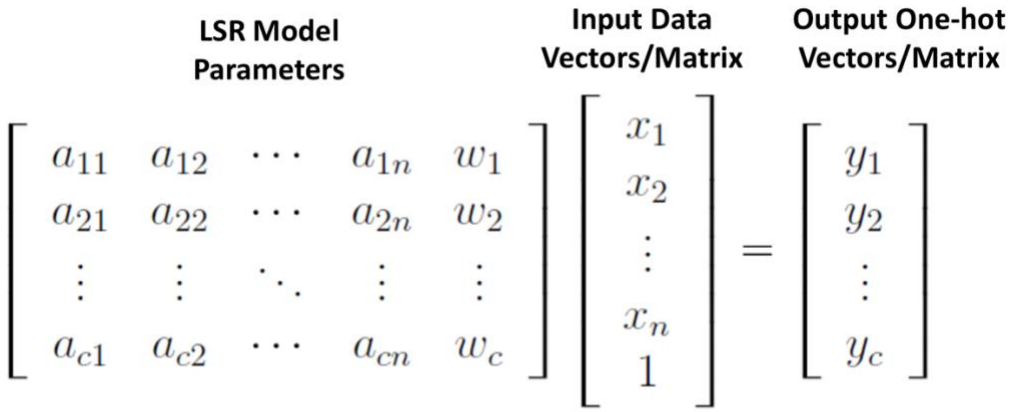
1st viewpoint (BP)

- Parameters to optimize in large nonlinear networks
- Backpropagation – SGD

2nd viewpoint (FF)

- **Parameters of linear least-squared regression (LSR) models**
- **Label-guided linear LSR**
  - True label used in the output layer
  - Pseudo label used in intermediate FC layers

Filter weights determination via LSR (Least Square Regression) is shown below:



$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & w_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & w_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{c1} & a_{c2} & \cdots & a_{cn} & w_c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_c \end{bmatrix}$$

- **Intermedia FC Layer: using pseudo labels with c=120 or 84**
- **Output layer using true labels with c=10**

Use the training data input and output data parameters such that output matrix is of 0 and 1 and input matrix if made of features, we find the LSR model parameters.

Part 3: Performance comparison

FF uses LSR, Saab and all and gets quite good accuracy. It is interpretable and also very fast as only one pass, no backward propagation, rather than using the optimization.

Testing Accuracy

Dataset	MNIST	CIFAR-10	
FF	97.2%	62%	Decision Quality
Hybrid	98.4%	64%	Feature Quality
BP	99.1%	68%	

Hybrid: Convolutional layers (FF) + FC layers (BP-optimized MLP)

Consider multiple FF CNN and then ensemble the result. The other idea is data partitioning. We see that the performance is enhanced even more for FF.

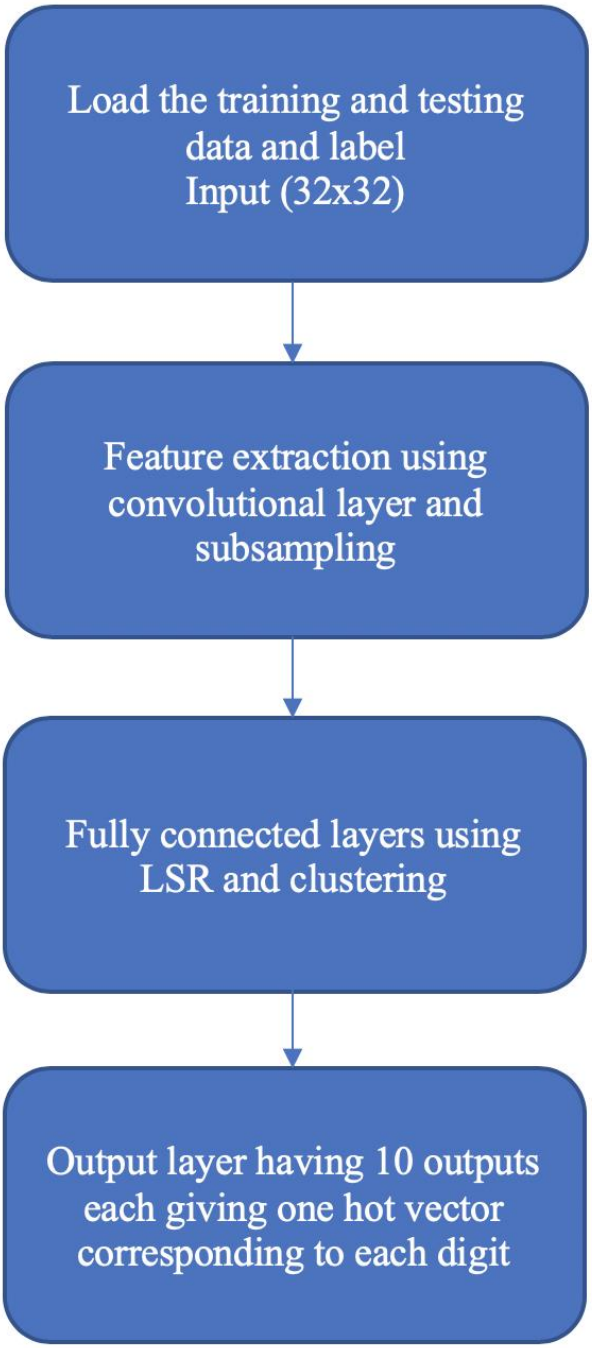
Easy, Hard, FF: A simple ensemble system evaluated on the easy set, the hard set, and the entire set
Hard+, FF+: An advanced ensemble system evaluated on the hard set and the entire set

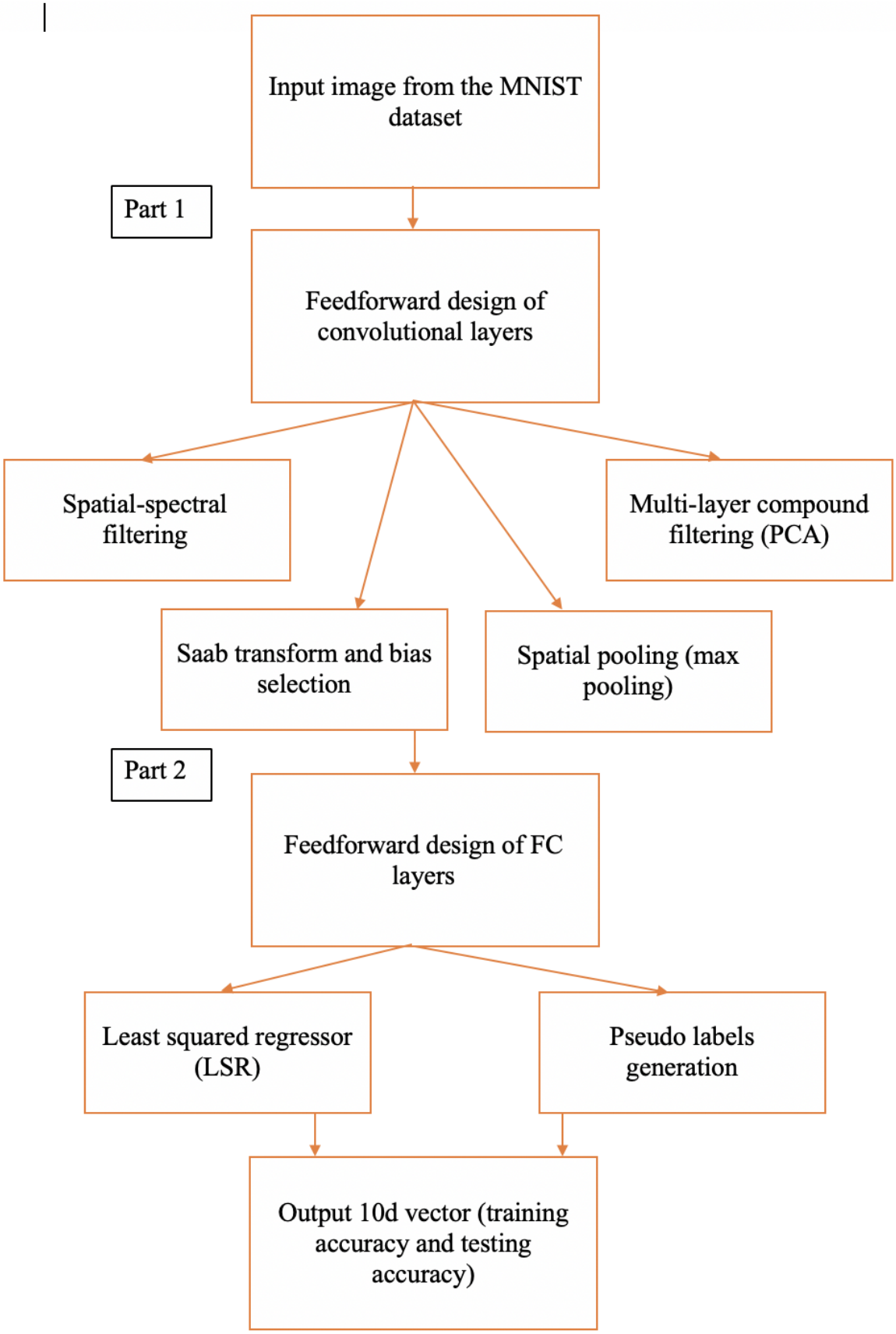
		Easy	Hard	Hard+	FF	FF$^+$	BP
MNIST	Train	99.9	90.0	98.2	98.9	**99.7**	
	Test	99.9	88.0	93.6	98.7	**99.3**	99.1
Cifar-10	Train	99.9	73.5	82.3	80.1	**87.2**	
	Test	98.2	66.2	68.8	74.2	**76.2**	68

III. Discussion

An FF-CNN consists of two modules in cascade: 1) the construction of conv layers using the Saab (Subspace approximation with adjusted bias) transforms, and 2) the construction of fully-connected (FC) layers using the multi-stage linear least squared regressor (LSR).

FF-CNNs summarized with a flow chart is shown below.


Load the training and testing data and label
Input (32x32)
Feature extraction using convolutional layer and subsampling
Fully connected layers using LSR and clustering
Output layer having 10 outputs each giving one hot vector corresponding to each digit


Input image from the MNIST dataset
Part 1
Feedforward design of convolutional layers
Spatial-spectral filtering
Multi-layer compound filtering (PCA)
Saab transform and bias selection
Spatial pooling (max pooling)
Part 2
Feedforward design of FC layers
Least squared regressor (LSR)
Pseudo labels generation
Output 10d vector (training accuracy and testing accuracy)

The CNN layers correspond to a vector space transformation. This transformation is based on two parts: one is dimension reduction through subspace approximations which is used in convolutional layers and other is training sample clustering and high to low dimensional remapping used to build FC layers. The convolutional layer has back to back spatial spectral filtering operations. To balance the loss from the use of spatial resolution, we use spectral representation and PCA. We also use Subspace approximation with adjusted bias (Saab), which is a variant of PCA. For the FC layers, each dimension of the output space corresponds to ground truth label of a class. Say there are 3 levels – feature space, sub-class space and class space. The first fully connected layer is constructed using linear least squared regressor (LSR) with pseudo labels from the feature space to the sub-class space. In the second FC layers, these pseudo labels are used as features and another LSR is created with true labels from sub-class space to class space. These FC layers in combination make the multi-layer perceptron (MLP), using multi-stage cascaded LSRs. This FF CNN not only decreases the dimension but also enhances discriminability. Through these various transformations, it gives the output using the strongest discriminability. In the spatial-spectral filtering, use the PCA to find the anchor vectors, and use this to find the neighborhood pattern, representing them as a linear combination of the anchor vectors. This is done mainly to enhance the discriminability. PCA does not need image labels, this is another advantage of the PCA based sub-space approximation. In the CNN, redundancy is controlled by the stride parameter.

Subspace approximation with adjusted bias (Saab)

- Subspace decomposition

$$\mathcal{S} = \mathcal{S}_{DC} \oplus \mathcal{S}_{AC}$$

- DC subspace is spanned by constant-element vector d $(1, \ldots, 1)$
- AC subspace is its orthogonal complement

Relaxing the assumption --- b_k =0 **for all** k

- The bias vector resides in the DC subspace
- The PCA analysis holds for the AC subspace

Summary of filter weight selection:

PCA conducted in the AC subspace of input vectors

$$y_{d,k} = \mathbf{a}_k^T \mathbf{x} + b_k$$

$\boldsymbol{a}_k$, k=1

- *The constant-element vector -> remove DC*

$\boldsymbol{a}_k$, k=2, … , 6

- *The first 5 principal components of input AC subspace*

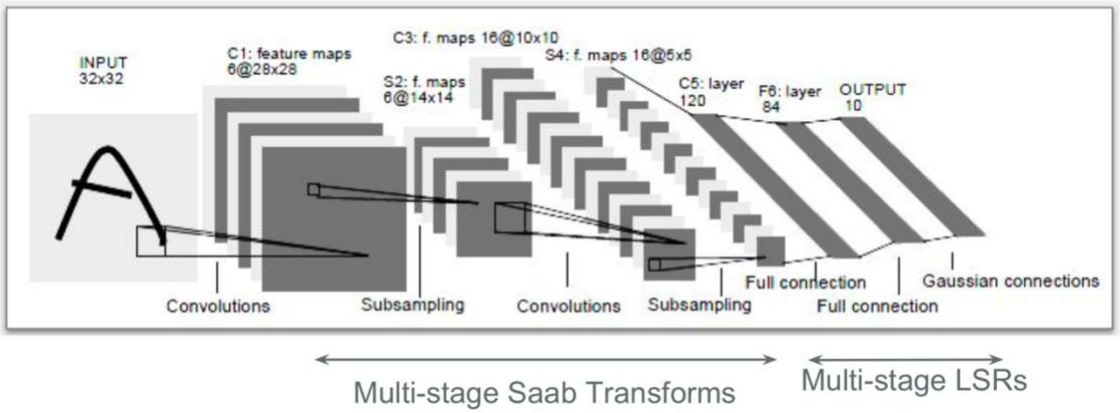
Bias, $b_k \geq \max ||\boldsymbol{x}||$

In the FC layer, there are 2 hidden layers of dimension 120 and 84 and output layer having 10 output nodes with 10-dimensional vector. In FF CNN, each FC layer is treated as linear LSR. We use k means clustering grouping different number inputs into different clusters and then generate one hot vector output such that there is 1 only for one number and 0 for the other elements. Each input has 2 labels, namely original class label and new cluster label. K pseudo labels are created for each input by combining the class and cluster labels. From these k pseudo labels, define the one hot vector corresponding to each digit 0~9, and using this set the linear LSR problem. To get the LSR, need a set of linear equations showing relationship between the input and output variables. Use the ensemble method, by combining the classification output vectors of multiple networks to enhance the classification accuracy. The cost for building the ensemble system in FF CNN is lower as compared to BP CNN and helps to gain better results.

There are 2 parts in this – one is how to compute the parameters in the convolutional layers and another part is how to compute the parameters in the FC i.e. fully connected layers.

Interpretable feedforward design of Convolutional Neural Networks

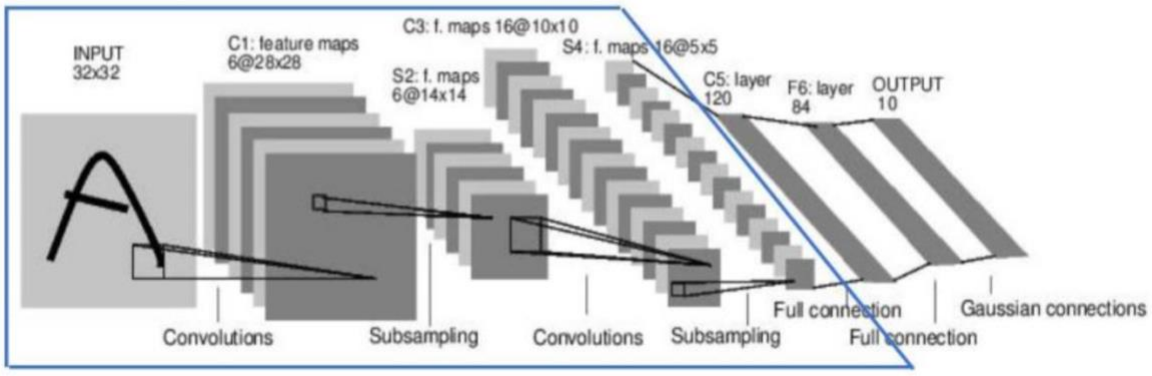
- Derive network parameters based on data statistics from the previous layer in an onepass manner
- Offer valuable insights into CNN architectures

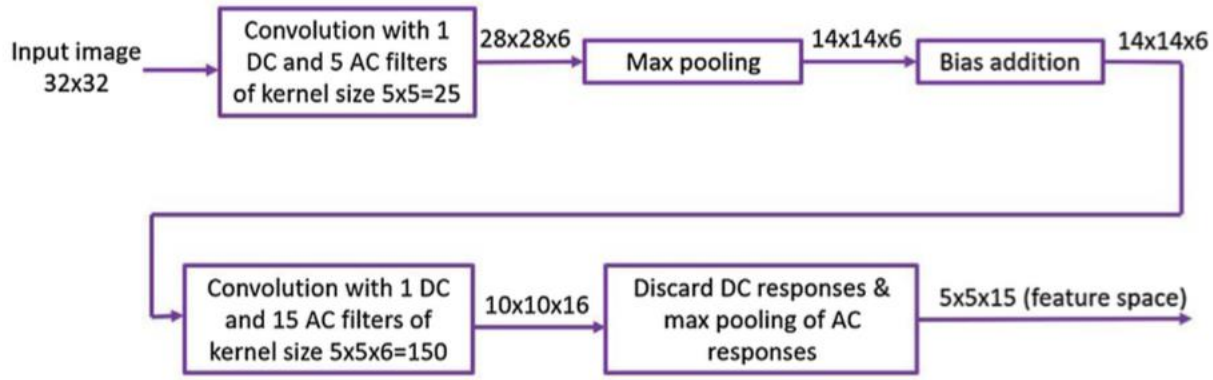


In this first problem, it is asked to describe the algorithm for the FF-CNN. So we separate these two parts and describe each part. We use a similar architecture to LeNet5 and use 2 convolutional layers and 3 fully connected layers. Firstly, we talk about the first modulo which is to try to compute the parameters in the convolutional layers. So in the feed forward design system we call this construction part multi-stage Saab transform. Below I talk about what is Saab transform and how to learn about this Saab transform.

Two modules of FF-CNNs:

- The convolutional layers construction: multi-stage Saab transform





Saab transform stands for Subspace approximation with adjusted bias transform. Here, we learn all the parameters in the convolutional layers through computation based on some statistics in the input data. So one stage Saab transform corresponds to one stage convolutional layer so since we have 2 convolutional layers in the LeNet5 architecture, thus we need 2 stages of Saab transform. So to learn those parameters in the Saab transform, the explanation is shown below. In the paper I referred for the feed forward design CNN, few terms are explained below. Talking about the transform kernels in the Saab or Saak transform, and also about the anchor vectors in both transforms and also familiar with the filters used in the convolution network previously. The goal is to try to learn the parameters of the filters and also for the transform kernels and anchor vectors. For the Saab transform, we separate the transform kernel into 2 categories, one is called DC anchor vector and other is AC anchor vector. The DC anchor vector tries to calculate the mean of the certain patches with some certain receptive field size. So the parameters for the DC kernel or DC anchor vector is listed below. It should be normalized to make the norm of the anchor vector to be 1, so divide by some square root of N. Another category we call it as AC anchor vector or AC kernel. So the parameters for the AC anchor vector or AC kernels are calculated using the PCA. First remove the mean-removed input data and then calculate the covariance matrix of all those mean-removed inputs. Then find the eigen vector and eigen values, choose the top eigen vector with the top eigen value. And those eigen vectors are just the AC anchor vectors. The only thing we need is those mean-removed input data and then based on those input data, we can compute all the anchor vector which means we decide the filters in the convolutional layers. In this way, we learn the weights in the conv layer.

Separate the transform kernels (anchor vector) to two categories

- DC anchor vector: $\mathbf{a}_0 = \frac{1}{\sqrt{N}}(1, 1, \cdots, 1)^T$
- AC anchor vector: PCA on the set of the mean-removed inputs

$$p_k = \mathbf{a}_k^T \mathbf{f}, \quad k = 0, 1, \cdots K.$$

$$\tilde{\mathbf{f}} = \mathbf{f} - p_0 \mathbf{a}_0$$

After we learn the anchor vectors, the next thing we decide is the bias term for each conv layers. So the bias term below is using only 2 constraints. When there are constraints, make sure that the output of each convolution layer operation should always be positive. Another constraint is to try to simplify the computation with the bias for each anchor vector making it same. So we can just decide the value for the bias based on the two constraints. So now determine the bias term in the conv layer. After deciding all the weights for the conv layer then compute the output for the first conv layer then compute the output for the second conv layer. And after each Saab transform, following the LeNet5 architecture, we also need some max pooling layer after that. The reason we need max pooling is that max pooling can reduce our spatial size, it reduces the feature dimensions we extract through the transform. Another thing that max pooling can make the feature extraction part more stable and robustness. Because there are some variations in the input but since we do max pooling, it always tries to get the maximum response in some spatial size and this way we are able to get more stable output for each layer. This is about the Saab transform part.

Bias selection:

- Affine transformation
- Two constraints:
  - None negative response constraint $y_k = \sum_{n=0}^{N-1} a_{k,n} x_n + b_k = \mathbf{a}_k^T \mathbf{x} + b_k \geq 0,$
  - Constant bias constraint $b_1 = b_2 = \cdots = b_K \equiv d\sqrt{K}.$

Max-pooling after each transformation

Max pooling: The next step is pooling or subsampling. However, do not use the mean, for every 2x2 location, choose the largest one so called maximum pooling. Also this is non-overlapping. Thus after the pooling, get the 28x28 to 14x14 response, because after do the pooling will partition 28x28 to 14x14 non overlapping blocks. Do this for each filter, doing the spatial pooling. This convolution and pooling parts are called as one layer which takes input 32x32 and gives the output 14x14x6. Each filter is a spectral component here the number 6 denotes that. Thus the output obtained is called a tensor.

We do not do pooling in the spectral, only in the spatial domain it is performed. Above we see that 224x224 becomes 112x112. It becomes one half of each dimension. There are different types of pooling – Average pooling, max pooling or min pooling. To do this, we use maximum pooling as below.

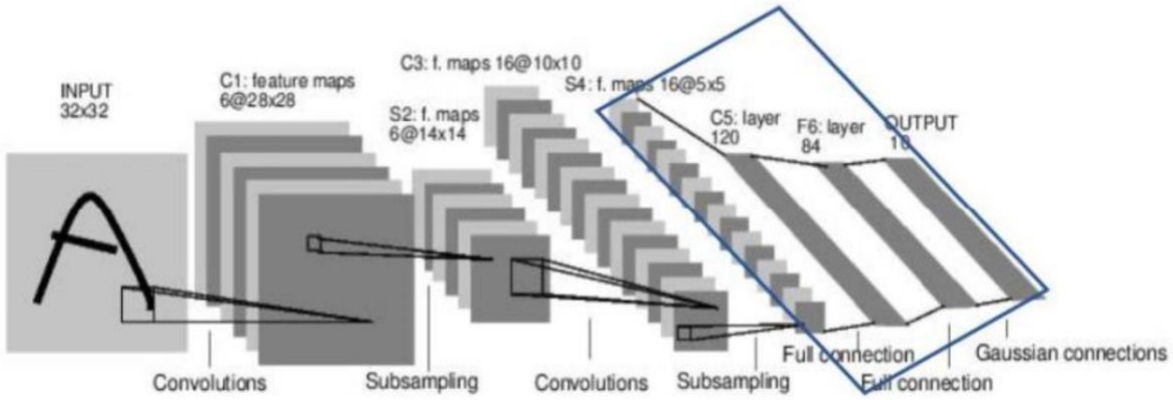
For every 2x2 block in the spatial domain, choose the maximum value. Repeat this for each filter spatially take 2x2 blocks and replace by one maximum value. We do not do average as sometimes linear is not good in adapting and learning. So we use non-linear. Maximum pooling performance is much better than non-linear pooling. Thus, choose the point which has maximum representation capability.

Max pooling layer is added post the convolutional layer along with the activation function like relu, sigmoid, tanh, leaky relu. It reduces the number of parameters to be trained by reducing the size of the spatial representation, thus it also helps to prevent or reduce overfitting of the network.

Now we will talk about how we develop the parameters in the FC layer through one feed forward pass. The fully connected layers development is like multi-stage rectified least squared regressors. The output of the conv layer will have some spatial dimension by spatial dimension and we can just reshape that cuboid output to make it a 1D vector but it is still larger and we want to further reduce that 1D vector and try to map it to the final object class space and we try to do it gradually like first is to reduce the feature to become 120 dimension by using one full layer and we further reduce the dimension to 84 and finally become 10.

Two modules:

- The Fully-connected layers development: multi-stage rectified least-squared regressors



So for one FC layer to do this, we do some clustering which we can use k means to cluster all the input features. In this way, the number of k means clusters should be the number of the units we want like in first stage we want 120 as our output dimension so we need to generate 120 clusters by using k means. Another trick to do k means is that we don't perform k means on the entire data set but do k means for each object class so for example, for the MNIST dataset we have 10 object classes like from 0 to 9. Here we do k means for each object, so we do k means for 0 digits, and in this case, we need to say the cluster number to be 12 for 0. So we can pursue some sub class for 0, since we have 12 different kinds of 0. Same thing we repeat for 1, i.e. we have 12 sub class for 1 and we have 12 different 1s. Repeat this procedure for all 12 digits. The reason we do this is to capture the diversity of single class, so for 0 class we have different writing style for 0 so one is do k means clustering and try to separate all these different kinds of 0 even though they belong to the same object class. After that we set a target for the different clusters. Like mentioned above, for 0 there are 12 different 0s, same for 1s and all other digits. So if we combine all these then we get 120 clusters so for each cluster we can set a 1 hot vector which is like for the first cluster, the one hot vector has the dimension of 120. And for the first cluster it will be 1 in the first dimension and other 119 dimension show this set to be 0. For the second cluster, second dimension should be set 1 and other 119 dimensions should be set 0. This is what we call the one hot vector encoder, for each cluster. In this way, for all the training data get some encoder target which is one hot vector, based on the k means clustering results. After that try to set the target for the LSR problem. LSR stands for least square regressor. Thus we set the LSR with the target y and input is x being the original features which we get from the conv layer. And then there is weight matrix, which is computed

through LSR. This weight what we get is the weight for the first FC layer. And after we compute the first FC layer, we get some Relu activation just like we did in the LeNet5 model. Repeat this procedure to get the weight of the second FC layer. For the second FC layer, we know that the input feature is like the feature of 120 dimensions and use the k means clustering to generate 80 different clusters and for each cluster we give the one hot vector as the target and we repeat this same thing to build the least square regressor for the second FC layer. Repeat this process to get the final object classes to get the final output.

Pseudo-labels generation
- Cluster training samples of the same class
- Generate pseudo categories
  - Capture the diversity of a single class
- Encode the one-hot vector as targets of LSR problem

ReLU after each LSR

Transform samples to final decision space gradually

In the FF CNN, each FC layer corresponds to linear LSR, final output of the last FC layer is a one hot vector. We use k means clustering on the input where each input has 2 labels original class label and new cluster label. We then combine these cluster labels with the class and create k pseudo labels for individual input. Then set up the linear LSR using one hot vector which is defined by k pseudo labels and do this for multiple stages. Hidden layers and pseudo label generation is added in this model to make it an advantage.

The last conv layer's output has a physical explanation in the FF CNN design. This output is the spatial-spectral transform of the input image. In the later layers, this provides as a feature vector for the classifier. It is fed to the fully connected layer and a mathematical model aligns the input and output feature spaces by mapping the samples from a pseudo class to one of orthogonal unit vectors. This spans the output feature space. The feature space alignment serves the purpose to reduce the cross-entropy value and generate more number of discriminant features. Hence, this cascade of least squared regressors (LSRs) present in multiple fully connected layers is like a sequential feature space alignment procedure.

The similarities and differences between Feed forward designed CNNs (FF-CNNs) and backpropagation-designed CNNs (BP-CNNs) is explained below.

Based on various properties and parameters, I have compared the FF-CNNs and BP-CNNs below.

Principle – The BP design has 3 parts: set of training and testing data, network architecture, cost function at output side for optimization purpose. These factors need to be decided. Novel network architectures can be implemented, also compare various cost functions and use the one which gives best performance. In contrast, FF design has a combination of spatial-spectral transformation in conv layer serving 2 things like discriminability and dimension reduction. They use PCA and LSR, clustering to get good performance with correct output. Thus, it uses data statistics while BP uses optimization.

Mathematical tools – BP uses stochastic gradient descent technique in optimizing the cost function, whereas FF uses linear algebra and statistics for training data collection and labelling.

Interpretability – It is difficult to interpret the BP CNN, because we do not get a full understanding of what the model does inside. However, FF CNN on the other hand, it is quite easy to interpret as it has one pass and all the parameters in the model used are known. We can explain about the filter weights, multi-layer CNN architecture. Also the connection between the FF and BP, using the cross entropy values of the intermediate layers can be studied.

Modularity – BP depends on the input data as well as output labels. The model parameters are influenced by the information at both the ends, using the input and output details, the parameters in the intermediate layer are varied to get fine output. Shallow layers capture low level image features and deeper layers capture semantic image information. The BP design is end-to-end. For FF, there are 2 parts in the whole design, conv layers is about the feature extraction using Saab and Fc layers determines the decision labels using classification.

Robustness – Both BP and FF design CNNs are vulnerable to adversarial attacks, as shown above. This can be less attacking when consider the ensemble system by fusing the results of multiple networks to get the best outcome.

Training complexity – BP is an iterative optimization model which processes all the training samples in one epoch and takes 100s of epochs for the network to converge. Whereas, the FF design CNN is quite fast compared to BP CNN. On a small training data, PCA and LSR can be conducted, is we consider the conv layer, we see that the

covariance matrix converges quickly. Also the regression analysis with the selected samples is quite better.

Architecture – BP network architecture has constraint, to enable the end to end optimization. FF network architecture does not have any architecture constraint.

Generalizability – In the BP design, there is a tight coupling between the data space and the decision space. Because of this, we need to design different networks to implement different tasks as the cost function is different for each application, for the same input sample space. In the FF design, all the tasks can use the same convolutional layers since they depend only on the input data space. For the different tasks, design different FC layers and join the information from different techniques.

Performance – The BP design offers the state of art performance by using the end to end optimization, and the choice of cost function and network architecture is very important along with this to determine the performance. FF design is still small scale and needs enhancements in the target performance. One of the solutions is to implement the ensemble system. Multiple classifiers are used after the feature extraction stage to get different outputs which are then ensembled to get the best output.

Differences in extracted features:
BP – Can generate the spatial patterns to tailor to the salient image parts like discriminant region and frequently occurred pattern, if some pattern occurs quite often then design the matched filters.
FF – Do not generate special patterns but project them to principal components, since there is no optimization in the backward direction, so just try to preserve energies. Correlation responses tend to be spread out and weaker, use all the coefficients. Features are less discriminant that those obtained by BP. Thus, the filters designed for FF are quite different from that of the BP weight.

Based on the characteristics of signal representation, the comparison of BP and FF is shown below:

Representation	FF	BP
Analogy	Projection into linear space	Matched Filtering
Search	PCA and LSR	Iterative Optimization
Sparsity	Denser	Sparser
Redundancy	No	Yes
Orthogonality	Yes	No

Signal correlation	Weaker	Stronger
Data flow	Broad-band	Narrow-band
Cost	Less	More

Comparison of the BP and FF designs:

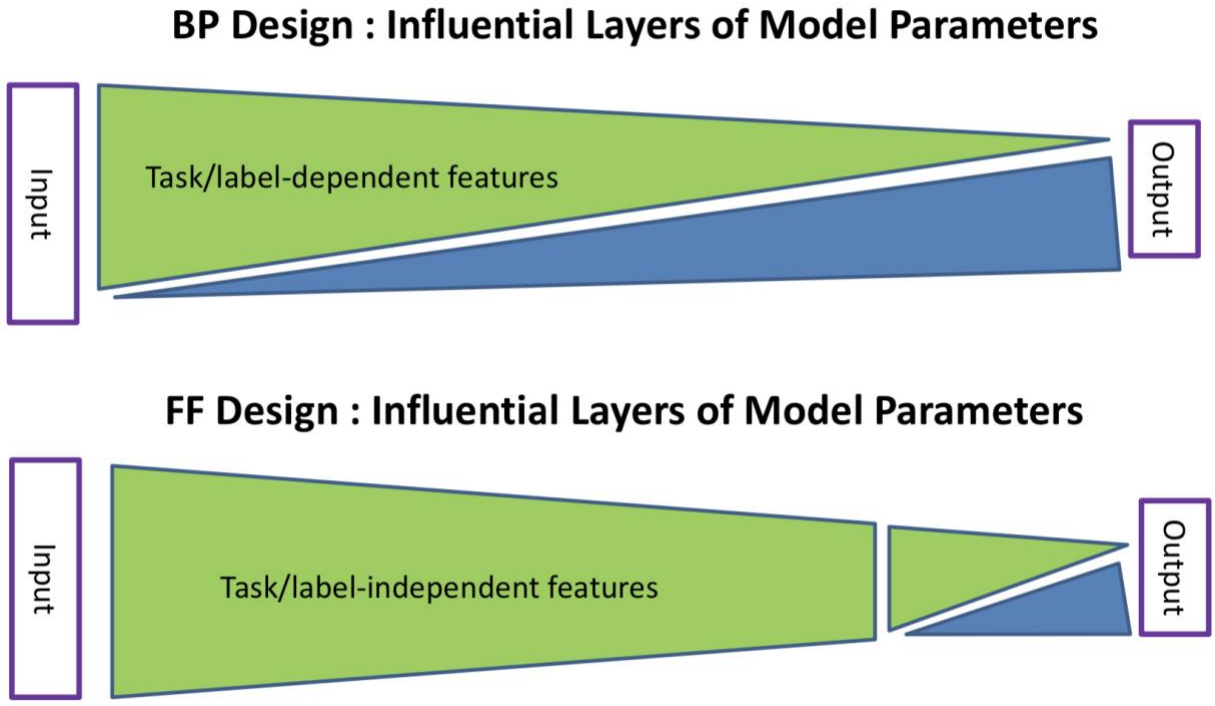
BP design

- Pros: state-of-the-art performance
- Cons: vulnerability to adversarial attacks, robustness to noise, generalizability (domain adaptation, weakly/ unsupervised learning)

FF design (still in infancy)

- Pros: Interpretability, robustness to noise, generalizability, etc.
- Cons: lower correct classification rate, vulnerability to adversarial attacks

Design	FF	BP
Principle	Data statistics centric (linear algebra and statistics)	System optimization centric (non-convex optimization)
Interpretability	Easy	Difficult
Modularity	Yes	No
Vulnerability to attacks/ Robustness	Low	Low
Training complexity	Lower	Higher
Architecture	More flexible	End-to-end network
Generalizability	Higher	Lower
Performance	To be further explored	State of the art
Hidden state	Not present	Present
Optimization	Not needed	Needed
Math tools	Linear algebra and statistics	Non-convex optimization
Ensemble learning	Lower complexity	Higher complexity



BP is global impact the label will impact all the filter weight. The input image pixels will also infer the filter weight. It is very complicated action between the both. Need to look at the error, backward propagate, again see the error and backward propagate and continue this process.

In the FF design, the first part at the input has nothing to do with the labels. Feature extraction just look at the pixels. Once find the Saab features, the second part is need the label to help do the clustering, can do the one pass.

The traditional CNN is the backward propagation (BP) CNN. It is a non-convex optimization problem. It is optimization based, try to minimize some cost function and use BP algorithm to do the parameter search. This does end to end optimization. In this we set the parameters and weights and find the features along with some error so we back propagate to minimize the error and optimize this model to gain a good output.

The feed forward (FF) design CNN has no hidden step, everything is explicit. This uses the approximation theory, try to change the signal representation but want to approximate well so control the loss, when going from layer to layer, do not want to lose much information. Another thing is to do the classification problem, to reach the end goal, minimize the cross entropy, reduce it or increase the recognition, discriminant power of the output. In this process, we just have one pass, thus we

have to set all the parameters correctly, in one go in the first pass and create a good output at the end. Thus the model with fewer parameters and giving same accuracy as the one with more parameters, is more robust and can be generalized.

The FF CNN is transparent and interpretable, no hidden states. The model is of the right size. It can accommodate data diversity, but will not be over parameterized, so that it is valuable.

Robustness against adversarial attacks

Generate these adversarial attacks on the FF and BP designs individually. The deapfool provides attacks of least visual distortion. The quality of the images attacked by FGS and BIM is poor. The performance of the BP and FF designs drops against these attacks mentioned. Hence, once parameters are known, it is easy to design powerful adversarial attacks of good visual quality.

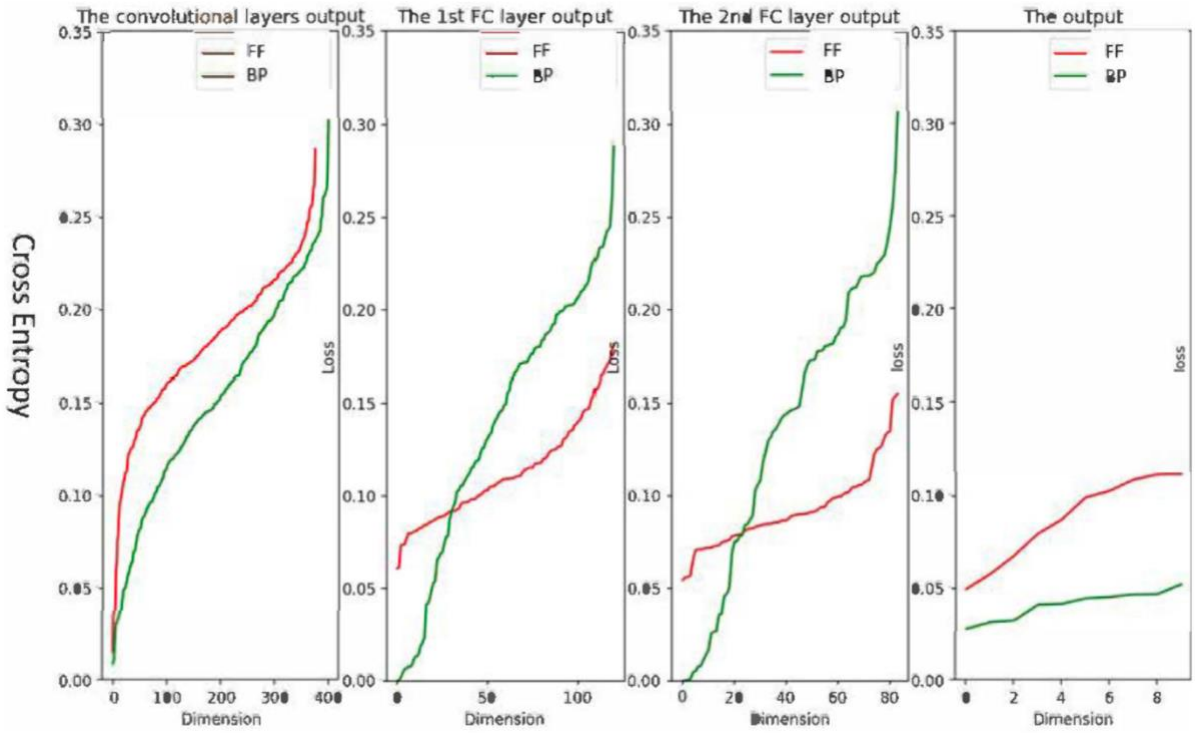
Table 3
Comparison of testing accuracies of BP and FF designs against FGS, BIM and Deepfool three adversarial attacks targeting at the BP design.

Attacks	CNN Design	MNIST	CIFAR-10
FGS	BP	6%	15%
FGS	FF	56%	21%
BIM	BP	1%	12%
BIM	FF	46%	31%
Deepfool	BP	2%	15%
Deepfool	FF	96%	59%

Table 4
Comparison of testing accuracies of BP and FF designs against FGS, BIM and Deepfool three adversarial attacks targeting at the FF design.

Attacks	CNN Design	MNIST	CIFAR-10
FGS	BP	34%	11%
FGS	FF	4%	6%
BIM	BP	57%	14%
BIM	FF	1%	12%
Deepfool	BP	97%	68%
Deepfool	FF	2%	16%

Comparison of rank ordered cross entropy values of BP and FF designs


Cross Entropy
The convolutional layers output
The 1st FC layer output
The 2nd FC layer output
The output
FF
BP
Loss
loss
Dimension

Problem 2: Image reconstructions from Saab coefficients

I. Abstract and Motivation

In the feed forward design CNN, there are two parts, one is data representation corresponding to the convolutional layer, where get low dimensional data from high

dimensional data. Do spatial spectral filtering and do the Saab transform, also can compute the inverse, can go back and reconstruct the whole image. This dimension reduction is very compact. Then the second part is the classification process to the output. Parameters chosen cannot be too small such that they cannot capture data diversity. Thus not very large parameters are needed to build a good model.

(Source: Lectures by Prof Kuo dated March 2019)

Subspace approximation with adjusted bias (Saab)

- Subspace decomposition

$$\mathcal{S} = \mathcal{S}_{DC} \oplus \mathcal{S}_{AC}$$

- DC subspace is spanned by constant-element vector d $(1, \ldots, 1)$
- AC subspace is its orthogonal complement

Relaxing the assumption --- b_k =0 **for all** k

- The bias vector resides in the DC subspace
- The PCA analysis holds for the AC subspace

Summary of filter weight selection:

PCA conducted in the AC subspace of input vectors

$$y_{d,k} = \mathbf{a}_k^T \mathbf{x} + b_k$$

$\boldsymbol{a}_k$, k=1

- *The constant-element vector -> remove DC*

$\boldsymbol{a}_k$, k=2, … , 6

- *The first 5 principal components of input AC subspace*

Bias, $b_k \geq \max ||\boldsymbol{x}||$

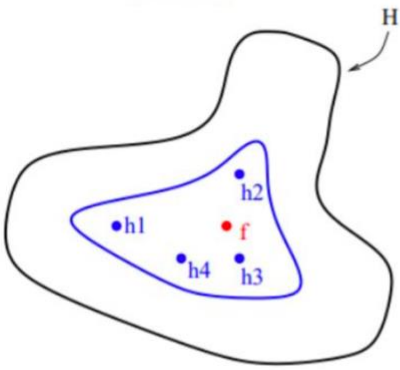
For homework problem 1, the explanation for the Saab transform is given and above as well. For problem 2, we need to find the Saab coefficients and also do the reconstruction from the Saab coefficients. For the Saab transform, there are several steps. The first step is about the PCA part. Also, do the inverse of the PCA, even though we cannot get 100% reconstruction, because we only choose top several PCA components. This is reported in the report. Then we add some bias term. This part is

completely reconstructed by after adding the bias, when do the inverse just need to subtract that bias term. Thus, the reconstruction algorithm is written.

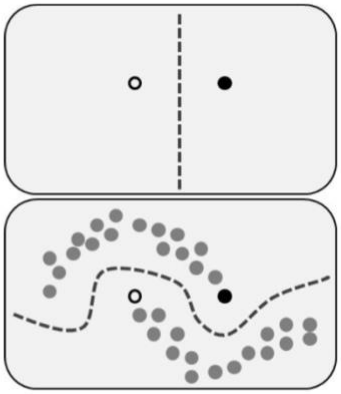
Further improve Saab-transform-based decision system

- Remove spatial-dimension redundancy
- Develop an ensemble system

Extend to semi-supervised decision system



Ensembles [11]

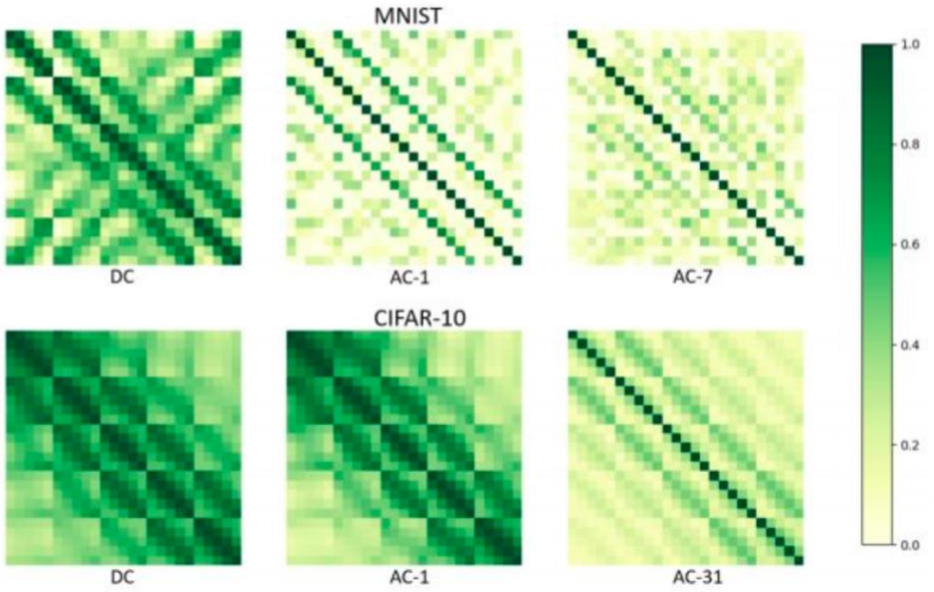


Semi-supervised learning [12]

Now, we talk about the last part of the problem 3 i.e. the ensemble part of the CNNs. We can extend the design of the FF-CNN, trying to make it better, so one way is we try to remove some spatial-dimension redundancy and the other way is to develop the ensemble system. Focus on the second part as the first part is not needed to implement in this homework. For the feature redundancy in the spatial dimensions, means after two stages of conv layers we get cuboids as representation for the input data which is 5x5x16 for MNIST images. We talk about the 5x5 spatial dimensions here, this has some redundancies and we want to remove them. So we try to apply another PCA but in this case that PCA is just compute for each channel for the outputs of the cuboids. This means we will apply 16 different PCAs for the output cuboids. The images shown below try to visualize these feature redundancies.

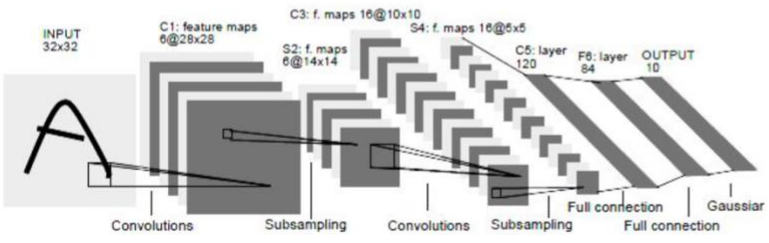
Feature redundancy in the spatial dimensions

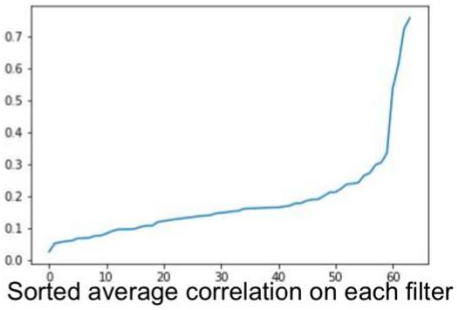
Stronger correlation in low frequency components



Say this is a 25x25 covariance matrix, which is each position in the 5x5 spatial size for different filters or different anchor vectors. The green above means a larger value which means a strong correlation, which we can say like for the DC term or some AC term. We can see that for both the datasets, there is some correlation in the 5x5 spatial size and we want to remove this. Before we try to apply PCA for each channels.

- Apply PCA to spatial dimensions on each filter
- Keep more energy for low frequency components





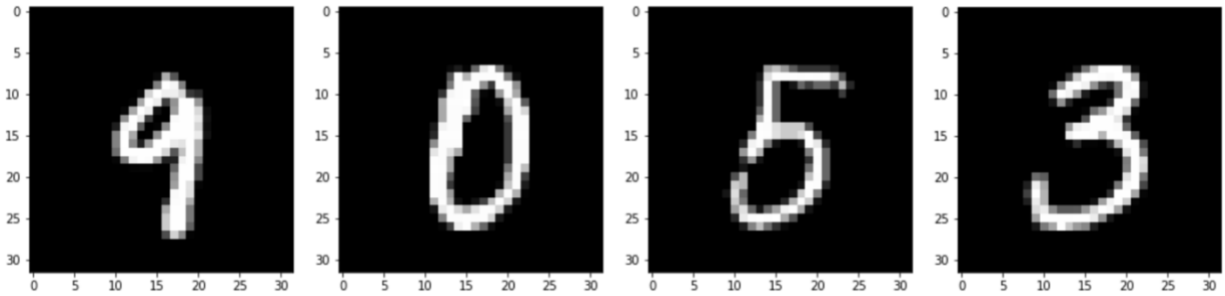
II. Approach and Procedure

Apply Saab transforms to images in the MNIST dataset.

- Compute the Saab coefficients of four handwritten digits images as shown in Figure and implement the reconstruction algorithm to transform the Saab coefficients back to images.
- To evaluate the reconstruction results, need to show the reconstructed images and compute PSNR scores between original images and reconstructed images.

Architecture setting:

In this problem, use two stage Saab transforms where the spatial size of the transform kernels is 4x4. The stride of each stage is 4 (non-overlapping). Thus, at the output, the dimension of your Saab coefficients of an image should be 2x2xN, where N is the number of transform kernel in the second stage. Need to evaluate on four different settings (different kernel numbers of each stage) and discuss the results.



For problem 2, all the parameters settings are fixed as mentioned, means we use 4x4 filters and we have non-overlapping strategies which means stride is equal to 4, when collecting all the patches or do the transform. For the reconstruction part, we do not have max pooling. After set the non-overlapping strategy, try to simplify the inverse part, to transform the collection of patches to the whole images. Another thing about the inverse transformations is, it is better to use PCA and inverse PCA. Use the building function in the PCA and inverse PCA. For the PCA algorithm, remove the feature mean of the data matrix and compute the kernel. This is different from the DC coefficient compute. This is feature mean. Here first remove the feature mean of the input vector and then apply the matrix multiplication of the transform kernel as soon as get the final coefficient.

(Source: Discussion lecture dated 04/16/2019)

In this homework, we apply the Saab transform on the MNIST dataset. Sometimes we may need to pad the images to change the size to become 32x32. Another problem is about the computation memory. When calculate the weights or parameters of the convolutional layer, this is an unsupervised process which is using

the Saab transform, so this part does not need so much data to compute the PCA and get the Eigen vectors. So can use just a subset for example say 10k images are enough to get good weight for the conv layer. But for the later part, the weight for the FC layer, to get the final classification results, this is a supervised process. Suppose if use the whole training images to learn the classifier part, this may not get good results, as reported in the paper. So need the whole training data set to get the correct results. But the online code uses the whole training data and stores all the features which may cause memory issues and thus can compute a batch of features say 1k at a time and run a for loop to go over all the training images and store that data. Also there is no problem of memory to save the features but might be problem to process very large matrix like here 60k training images, that is too big for the computer.

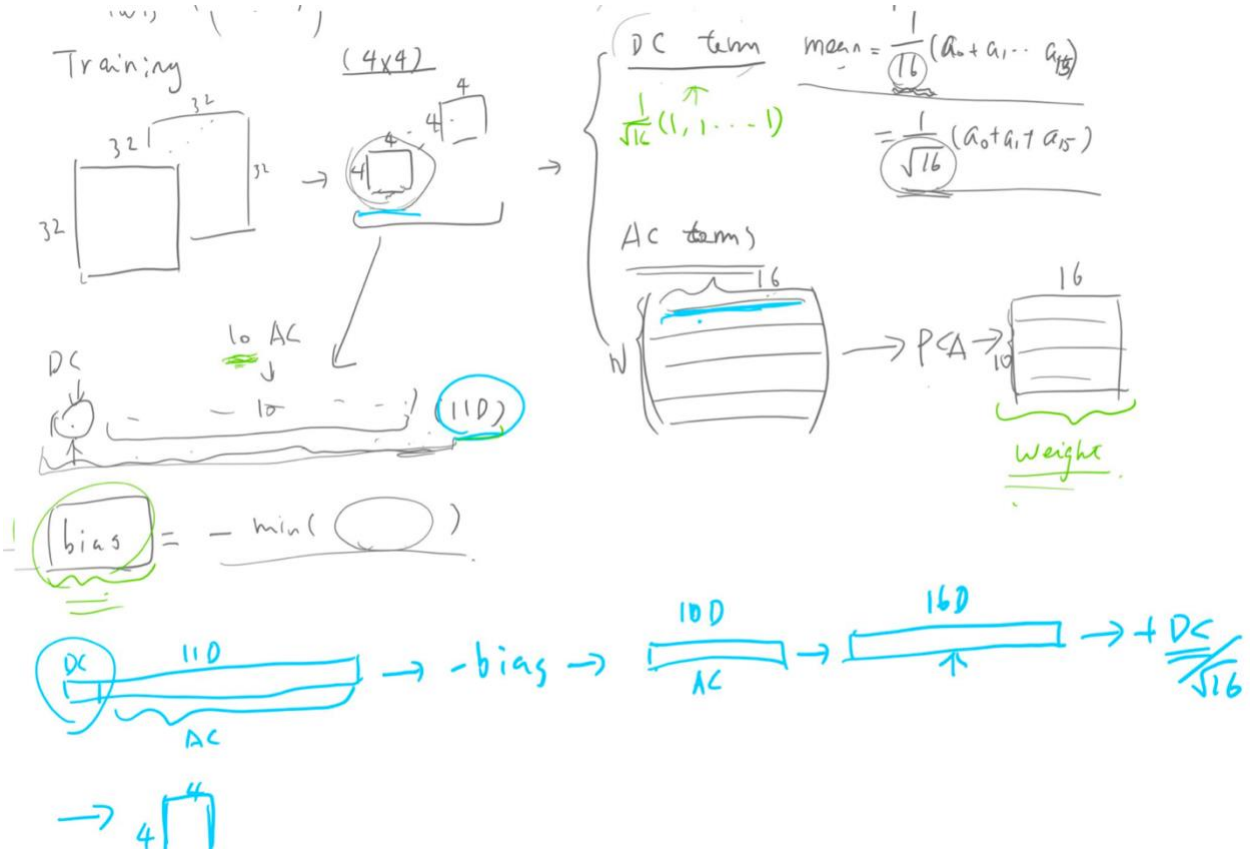
Application of the MNIST dataset

- MNIST dataset
  - 60k training images, 10k testing images
  - 10 classes, 28x28 -> 32x32
- Batch process

We try to process the data in the MNIST dataset which is 32x32 images. So first need to separate the training and testing process. For the training process, we have a lot of training data. For the problem 2, for the reconstruction problem, also need a training set to learn the kernels, not just learn the PCA only on one image like given. So need the training set but also in that case, do not need the whole training dataset, can select only a subset, it does not influence too much. When we get the training dataset, we collect some patches from this training set so we can just use the example, say filter size of 4x4 so we need to collect the 4x4 patches. Also while collecting the 4x4 patches, can do overlapping collection and non-overlapping collection. If we do non-overlapping, just one image will give like 34 patches. So if we have a lot of training images, we have a lot of patches too. These patches are used to compute the weights of the first conv layer based on the PCA. So we have 2

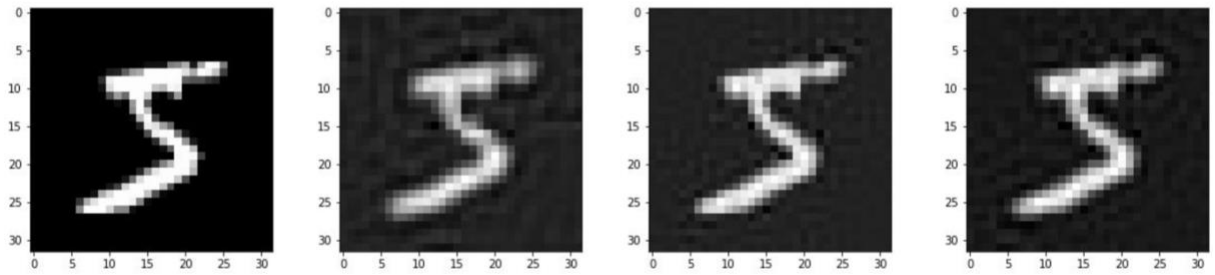
terms here – one is a DC and other is AC. The AC term is the PCA components learnt through the PCA calculation. The DC term is the mean of the patches. This means for each 4x4 patch, we calculate the mean i.e. for the pixel values of all the 15 pixels. These are DC coefficients which we get. The DC kernel has been provided above. The coefficients of the first convolutional layer is generated like a 1D vector, so the first term is the DC coefficient and the others are say 10 AC coefficients. We want them in the same scale. So if look at the DC kernels, they are the square root of the number of pixels. This is shown below. So these are the real DC coefficients we have after the one stage transform. For the AC terms, we know its computed from the PCA. So first thing is to prepare the data matrix, and calculate the covariant matrix of that data matrix, so the data matrix here is like rows represent number of data samples and columns of the matrix represent the dimensions for each sample i.e. 16. Also we need to remove the mean when calculating the PCA. So all the data inside the matrix should be 0 mean data which is each row in this data matrix. From this data matrix, compute the PCA and select some PCA components, dimension is 16 and rows number is say 10 here, so discard some high frequencies. This matrix is the weight or the parameter for the conv 1 layer and there is another corresponding to the DC term. For each 4x4 patches, we can transform it to DC, AC after one stage transform. The first one is DC coefficient and the other right ones are AC computed using the matrix. This gives the 11D features vector for each original patches. Next thing is need to compute the bias. There are 2 constraints for the bias term. One is we need to make the feature vector i.e. the output for the conv layer always positive and another one is want to have a single value to each dimension in the vector above. Thus make the bias term equal to the minus the minimum value for all the coefficients got from the training data. So if just add the bias term, all the coefficients will be positive and we only add just one number. This is the simplest way to add this term. The parameters to be remembered for reconstruction are weight and bias. Hence upto here, we finish oen stage of Saab transform and we calculated everything needed. Next is to build the reconstruction for this stage. For the reconstruction, the input is the Saab coefficients i.e. the 11D feature vector for each 4x4 patches. First step is to minus the bias to remove the effect as mentioned above, to do inverse of the above bias term part. After we remove the bias, we go to the PCA part. First thing is we need to treat the DC and AC separate. We just use the AC part which is 10D by using the inverse PCA, to inverse the 10D AC components to become like 16D original features which are located in the pixel space. For this one, we have removed the mean previously, so add the mean back. Next step is to add the DC term, also divide some scale. After add the DC term back to the 16D vector, finish the reconstruction for the 4x4 patches. Do this reconstruction for the whole image i.e. 64 patches and combine them and get the final output image. Below, I have shown the reconstructed images and also computed the PSNR values to evaluate the

reconstruction results. Above we have shown the whole process about one stage Saab transform and the reconstruction process. Similarly have the second stage, just repeat to have the two stage Saab transform and then go back to have two stage reconstruction step and thus get the final result.



The problem asks to use different parameter settings like say the filter number, which is set to 11 in the example given above. Basically change the number of AC like 5, 10, 12. Same thing for the second stage as well, try different number of AC terms from the PCA. Below are some results shown, N1/N2 represent the number of kernels or filters from the conv 1 or conv 2 layers. Also it shows the reconstruction results. And the second value shows the PSNR value.

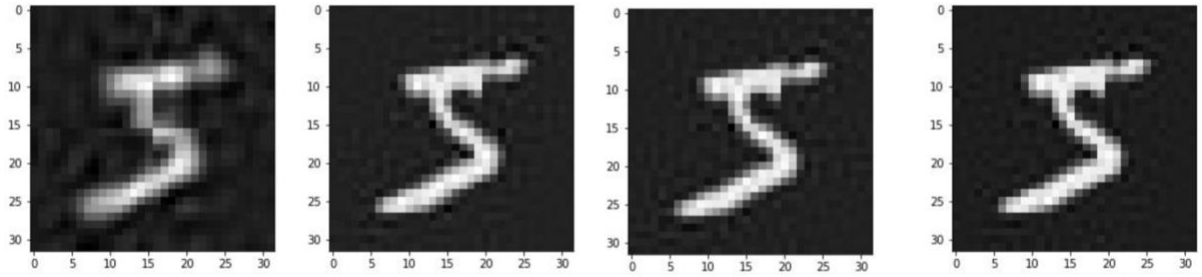
❖ Results: N1/N2



original 10/60, 22.3 10/100, 25.3 10/140, 28.6

If keep more AC kernels, it causes better reconstruction from the reconstruction or the PSNR value. Below is shown for some more different values.

❖ Results: N1/N2



12/40, 19.8 12/120, 26.8 12/140, 28.4 12/160, 30.5

In the report, show the trend of the PSNR values which is say PSNR becomes better if use more kernels in each stage. Below is shown the equation for the PSNR computation.

Evaluation

➢ PSNR

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

PSNR is related to the mean square error (mse) i.e. calculate the difference between the original images and the reconstructed images pixel wise and add all mse between 2 images and then use the equation above. The MAX is the maximum pixel value and the bottom part is the MSE.

Take an example of kernel size is 5*5 in both two stages, and kernel number is 10 in stage 1 and 16 in stage 2. We can get the output feature having the dimension of 5*5*16. 5*5 is the spatial dimension, and 16 is the spectral dimension. The C-PCA is to reduce the 5*5 into a lower dimension, for example 10D.
C-PCA is computed using each channel or each spectral dimension. PCA in Saab is computed using all channels.

'Layer_x/feature_expectation' these are mean values of each feature in the respective layer. It is the mean of each column of [number of patches of all images vs 16(4x4) filter size] matrix.

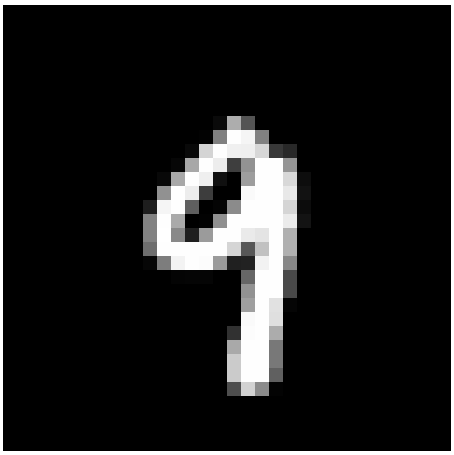
Need final saab coefficients and pca_params, for reconstructing image from saab cooeficients.

For reconstruction, you don't need maxpooling after each stage of saab transform.

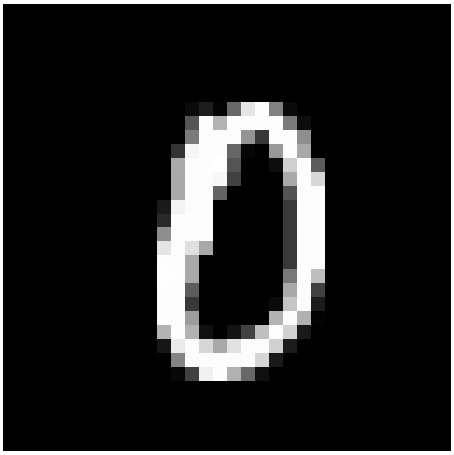
Saab coefficients are the features you get using Getfeature.py.
Get pca_params from Getkernel.py, then use it and the 4 images as the input of Getfeature.py to gain the feature of the 4 images, which is said as saab coefficients.
Next, reverse the feature to images and compare them with the original images.
Need to compute inverse saab transform, use the features from Getfeatures, and kernels, bias and pca_mean from Getkernel.

The scale factor is used to normalize the DC kernel (e.i. [1,1,...,1]/sqrt(16) , 16-D). Need 1/sqrt(16). AC kernels don't need scale factor, because PCA already normalizes all kernels.

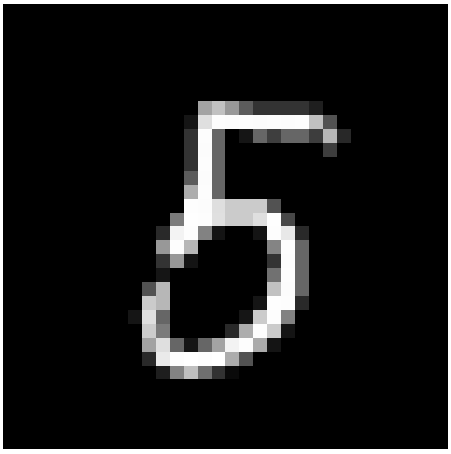
III. Experimental Results



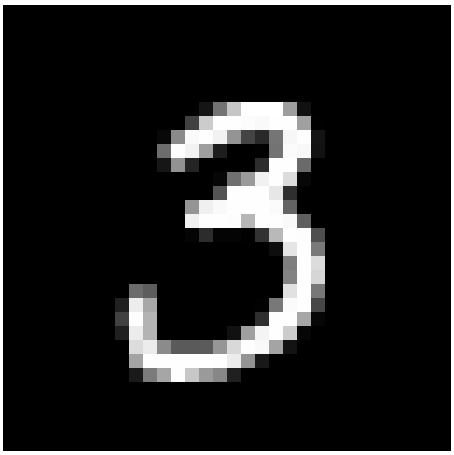
Input image



Input image



Input image



Input image

Setting 1: (different kernel numbers of each stage)
Kernel number: 5, 15
PSNR: 10.76

Setting 2:
Kernel number: 5, 19
PSNR: 12.32

Setting 3:
Kernel number: 8, 15
PSNR: 11.89

Setting 4:

Kernel number: 8, 19
PSNR: 13.45

IV. Discussion

I have referred the code mentioned in the link above for implementing problem 2. There are different python files in the MNIST_FF folder. The getkernel_compact.py is used to get the DC and AC kernels. It imports and loads the training data and labels as well as testing from the MNIST dataset. It uses the function Saab to create multi stage Saab transform. The key parameters to use are the kernel size, kernel number i.e. AC number. Tune these two parameters for this problem. The kernel size is 4,4 and kernel number is 5,15 for first setting. Output for this function is the parameters of the PCA and also the parameter for the conv layer, so it saves it in a dictionary for later use. Pca parameters are stored in a pickle file in the folder.

The second code is getfeature_compact.py which is used to get features. The feature which is output for the two stage Saab transform. It consists an imread image part where the 4 png image files provided are taken as input i.e. training images here. Again here it loads some data and also load the parameters to calculate through the previous files. Then use the saab.initialize function to give input data of training images (9,0,5,3) and another input is the pca parameters. This function will give the final output feature of some dimension. Then save this feature in the pickle file for

later use. Also, comment the part for max pooling in saab.py since there is no max pooling used because the image should be reconstructed properly, however, if max pooling is done, we lose a lot of information data and hence it is difficult to reconstruct the image back. The stride parameter is taken as 4.

The last code getweight_compact.py gets the weight for the FC layer. First load the feature calculated in the previous file and reshape the feature to become a 1D vector so as the input for the FC layer. Further k means is used to compute the sub-class, different clusters. Then compute the one hot vector part. Also compute the least square regression part. After we solve this, we get the weight. So this weight and bias is for the FC layer. And finally find some training accuracies. Also then save the weight and the bias. Basically, all these 3 files try to calculate the weight for the conv layer, FC layer, and also can learn the training accuracy here. One part added by me is I modified the file to add the testing accuracy.

Thus, the Saab coefficients are calculated for doing the feature extraction and now I write the code reconstruction.py to reconstruct the images, transform the Saab coefficients back to images.

Another part is for the problem 2 about the reconstruction, required to use some different architecture setting than the LeNet5 and in this one there's no need of max pooling because it tries to simplify the reconstruction part. The get kernel file already gives the max pooling so need to modify the saab.multi_Saab_transform for problem 2 and write the reconstruction algorithm. For MNIST dataset, there are 60k images, which is quite a large dataset, can also run 10k images, depending on the computer processor. For the kernel for the Saab transform actually gets a good filter weight but may require full dataset to get reasonable results.

In the code reconstruction.py, use the features and pca parameters pickle file and the 4 input images. Here, we have the Saab coefficients and we need to reconstruct the image from these features thus we do the inverse of all the steps we did from input to features, and go backwards. The output I got from running getkernel.py and getfeature.py, the transformed image features, now use this as the input for the reconstruction method. Add the bias which was subtracted previously, take inverse and multiply the kernels which were multiplied to input before. Perform the inverse operation for the AC and DC components. Add the feature expectation and reshape the layers output sample patches. Repeat this procedure once more as there were two conv layers. Stack them and display the output in grayscale. Also can do normalization to get better output.

To find the PSNR value, define a function which calculates the mse for the original image and the output reconstructed image and returns and prints the PSNR using the formula for each of the 4 images.

I have shown the reconstructed images for different settings of kernel number above along with the PSNR values between the original and the reconstructed images.

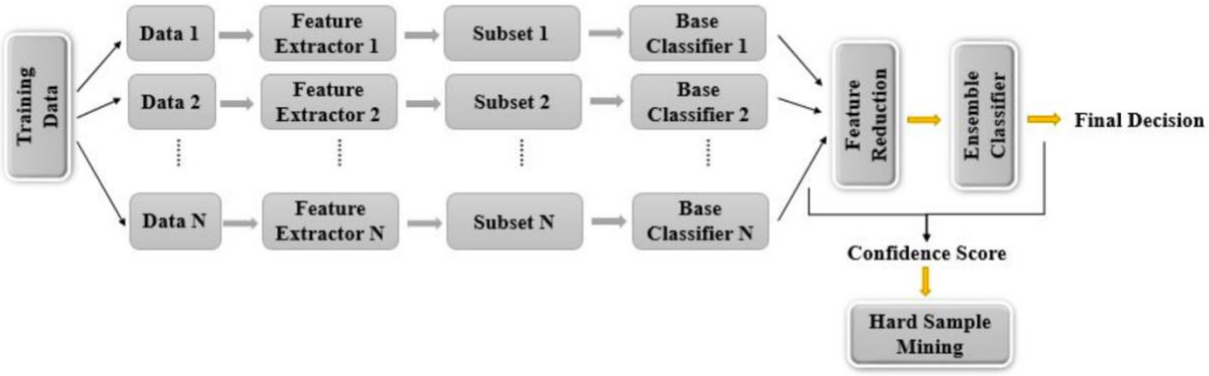
N1/N2 represent the number of kernels or filters from the conv 1 or conv 2 layers. If keep more number of AC kernels, it causes better reconstruction from the reconstruction and higher the PSNR value. The trend of the PSNR values shows that the PSNR becomes better if use more number of kernels in each stage.

Problem 3: Handwritten digits recognition using ensembles of feedforward design

I. Abstract and Motivation

Now we talk about the second part i.e. how to improve the FF CNN design, try to build the ensemble system. Below shown is the architecture to be used in the problem 3. Firstly, need to build a different base classifier. So for each base classifier, each is different than the other, and each base classifier is a single or individual FF-CNN. So the difference can be a lot. For example, it will have different architecture for the network, or change in the filter size or filter number or use of different feature got from the conv layer or use different input data i.e. different input data representation. For example, for the MNIST dataset, one can learn the Laws filter before and can use the input data as the Laws filter response maps and can get different base classifiers. Here, we need to build 10 different base classifiers and need to describe what is the variation for the base classifier and need to give the reason why chose they 10 different classifiers. After building the 10 different base classifiers, for each classifier we get 20 prediction vectors, and use these 20 prediction vectors as new features. In this case we get 100D new features. Further need some feature reduction with another PCA to reduce the 100 dimension to smaller i.e. 30 or 40. After reducing it to say suppose 40D feature vector, we can build the ensemble classifier, can use SVM as the ensemble classifier, try to get a better performance. So the ensemble classifier finally gives the final decisions for the input images. So this is the overview of the ensemble system need to build in problem 3.

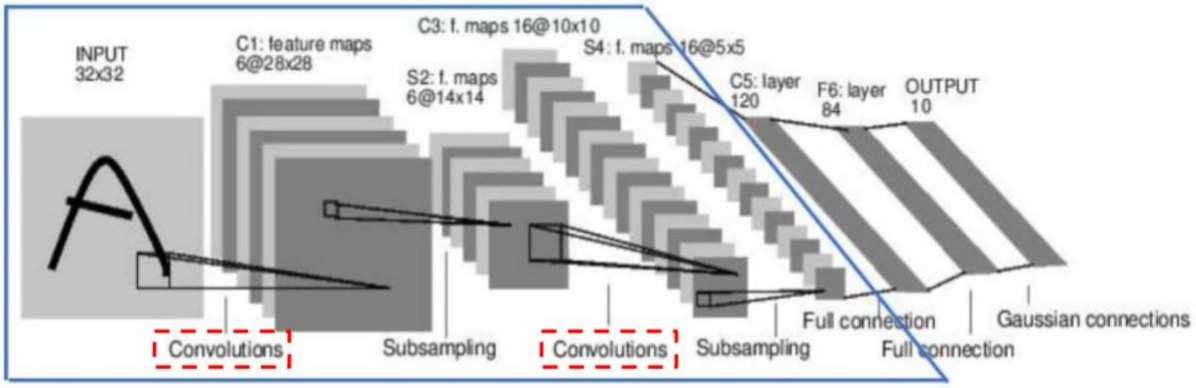
- A simple but effective ensemble method
- Base classifiers: FF-CNNs, ensemble classifier: SVM
- Diversity is the key to success



Below is the introduction for the ensemble paper. The paper introduces 3 types of diversity sources. Diversity is the key to get a good ensemble system. Below given are the different types of diversity sources that can be used. Also we can use different filter size for conv1 and conv2 layers. Like in the LeNet5, we used filter size 5x5, now can change it to 3x3.

S1: Different filter size settings

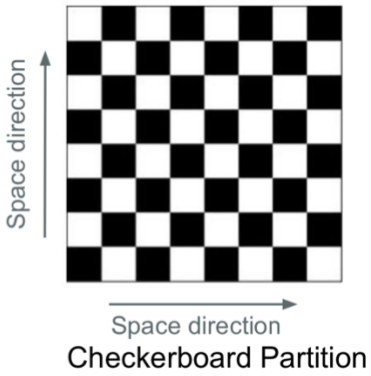
- (5x5, 5x5), (3x3, 5x5), (5x5, 3x3), and (3x3, 3x3)



Another thing is use the subset of the available feature set. Our feature set can be from the conv1 outputs and the conv2 outputs. So randomly select some features from this feature sets.

S2: Subsets of the Available Features $\mathbf{F} = [\mathbf{F}_{conv1}, \mathbf{F}_{conv2}]$

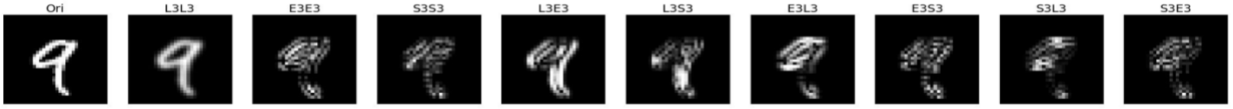
- Randomly select features from Conv1
- Randomly select features from Conv2
- Checkerboard partition of features from Conv1 in the spatial dimension



Checkerboard Partition

Another one is to use the different representation of the input images. So, for the MNIST part, try to use the response from the 3x3 Laws filter bank. Then can have 9 different response maps as to new input images. This is sent to the FF CNN design system and learn and get one hot base classifier.

- S3: Different Representations of Input Images
  - RGB, YCbCr and Lab color spaces
  - 3x3 Laws filter bank



Selection hard samples

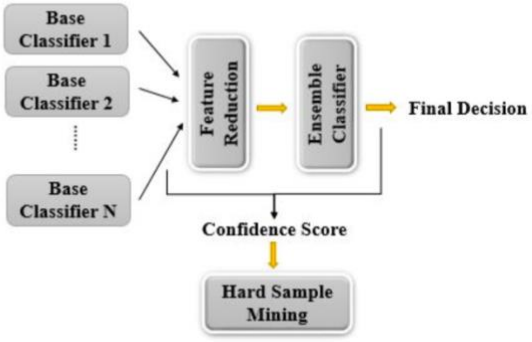
- Confidence score

$$CS1_i = max(P_{final}(\mathbf{y}|\mathbf{x_i}))$$

$$CS2_i = N_i/N_{all}$$

$$\mathbf{X_{hard}} = \{\mathbf{x_i} : CS1_i < th_1 \quad and \quad CS2_i < th_2\},$$

A new FF-CNN ensemble targeting at the hard samples set



Below given is the introduction of the dataset.

Datasets:

- MNIST
  - Handwritten digits 0-9
  - Gray-scale images with size 32x32
  - Training set: 60k, Testing set: 10k
- CIFAR-10
  - 10 classes of tiny RGB images with size 32 × 32
  - Training set: 50k, Testing set: 10k

Evaluation:

- Top-1 classification accuracy

Adopt the LeNet-5 architecture

Below given is the comparison, to evaluate the first type of diversity which is different filter size here. So for the MNIST dataset, by combining 3 types of FF design model we can improve by 1% on the testing accuracy.

S1: Convolutional-layer parameter settings

- Filter size: (5x5, 5x5), (3x3, 5x5), (5x5, 3x3), and (3x3, 3x3)
- Ensembles help to improve the performance

Table 1. Comparison of classification accuracy (%) of BP-CNN, four FF-CNNs and their ensemble on MNIST and CIFAR-10.

	BP	FF-1	FF-2	FF-3	FF-4	Ens.
Filter Size	(5,5)	(5,5)	(5,3)	(3,5)	(3,3)	-
MINST	**99.1**	97.1	97.0	97.2	97.3	98.2
CIFAR-10	68.7	63.7	65.3	64.2	65.9	**69.9**

Another type of diversity is different feature subsets. We try different combinations for the subsets and we can find it also tries to improve about 1% when combine different base classifiers.

S2: Different feature subsets

- Randomly select 3/4 of the features
- ED-1: combination of Conv2, Conv1-1, Conv1-2
- ED-2: combination of six different Conv1-RD
- ED-3: combination of twelve different Conv2-RD
- ED-4: combination of six different Conv1-RD and twelve different Conv1-RD.

Table 2. The testing classification accuracy (%) on MNIST and CIFAR-10 using feature subset diversities.

	Conv2	Conv1-1	Conv1-2	Conv1-RD	Conv2-RD	ED-1	ED-2	ED-3	ED-4
MINST	97.1	95.4	95.3	96.8	95.2	97.7	97.6	97.2	**98.0**
CIFAR-10	63.7	64.3	64.4	62.3	64.2	68.7	66.0	68.4	**69.3**

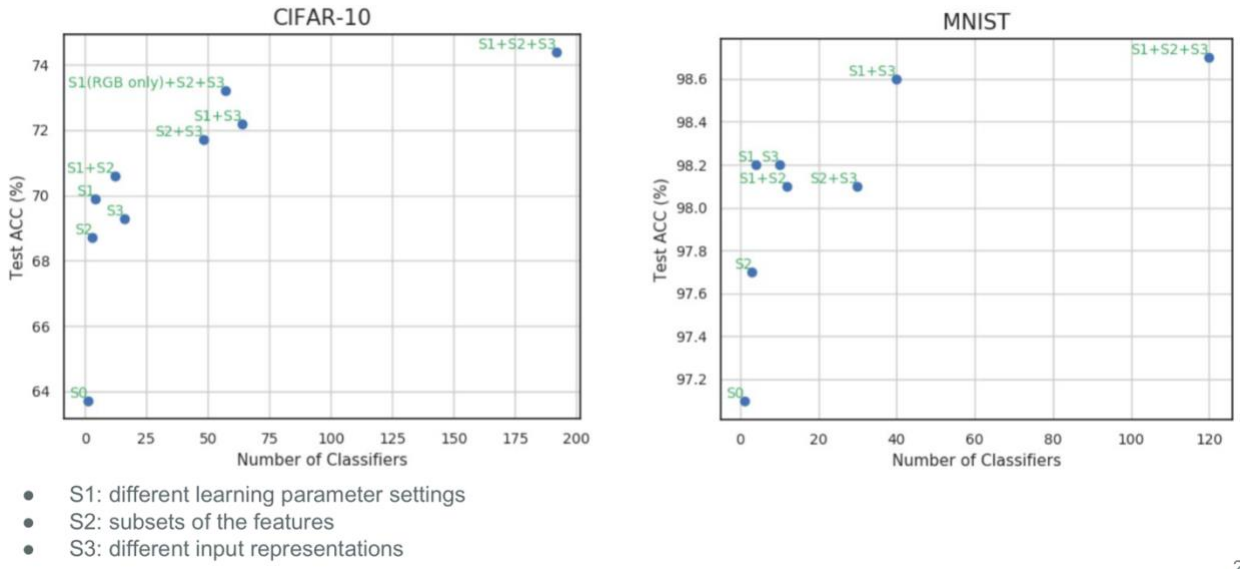
Another one is for the different representation of the input images, we do similar stuff. For MNIST dataset, we combine with 9 different base classifiers by using 9 different Laws filter response maps. And it also gets similar results as we below.

- S3: Different representations of input images
  - Treat the three channels separately for YCbCr, Lab
  - L1 to L9 denote the filtered maps by L3L3, E3E3, S3S3, L3S3, S3L3, L3E3, E3L3, S3E3, and E3S3 Laws filters

Table 3. The testing classification accuracy (%) on MNIST and CIFAR-10

	RGB	Grey	YCbCr	Lab	L1	L2	L3	L4	L5	L6	L7	L8	L9	ED
MINST	-	97.1	-	-	97.0	95.1	87.8	92.6	93.7	94.9	95.6	93.1	92.6	**98.2**
CIFAR-10	63.7	-	54.0/41.4/41.1	53.2/40.0/41.0	50.6	44.8	44.5	46.3	48.3	44.9	47.6	43.0	45.8	**69.6**

Below is the summary of all different types of diversity. For the MNIST dataset on the right side, we can refer the image below to say how we can improve the performance by ensemble different number of base classifiers. There is no need to go to a larger number of classifiers, observe the improvements by the design.



- S1: different learning parameter settings
- S2: subsets of the features
- S3: different input representations

Diversity is the key to get good ensemble systems and that's the reason why we test different types of diversity, try to make the base classifier different from each other and compensate the results with each other. So below, introduce the two types of diversity measures. In the paper, one is called the Q-statistic and other is called entropy measure. So the table below, S1, S2, S3 are the different types of diversity and by combining all those 3 types of diversity by using the diversity, we can prove that when combining more different base classifier, can get more diversed system. This explains the improvement in the performance. For this homework, it is not needed to compute this.

Two diversity measures

- Q-statistic: 1 indicates no difference and 0 indicates the highest possible diversity
- Entropy measure: 0 indicates no difference and 1 indicates the highest possible diversity

Table 4.7: Diversity measurements on CIFAR-10 training set. Three types of diversity sources are evaluated separately in the first to third columns, and the last column reports the measurements on all base classifiers in an ensemble.

	S1	S2	S3	ALL
Q-statistic	0.88	0.89	0.66	0.61
Entropy measure	0.21	0.24	0.47	0.49

II. Approach and Procedure

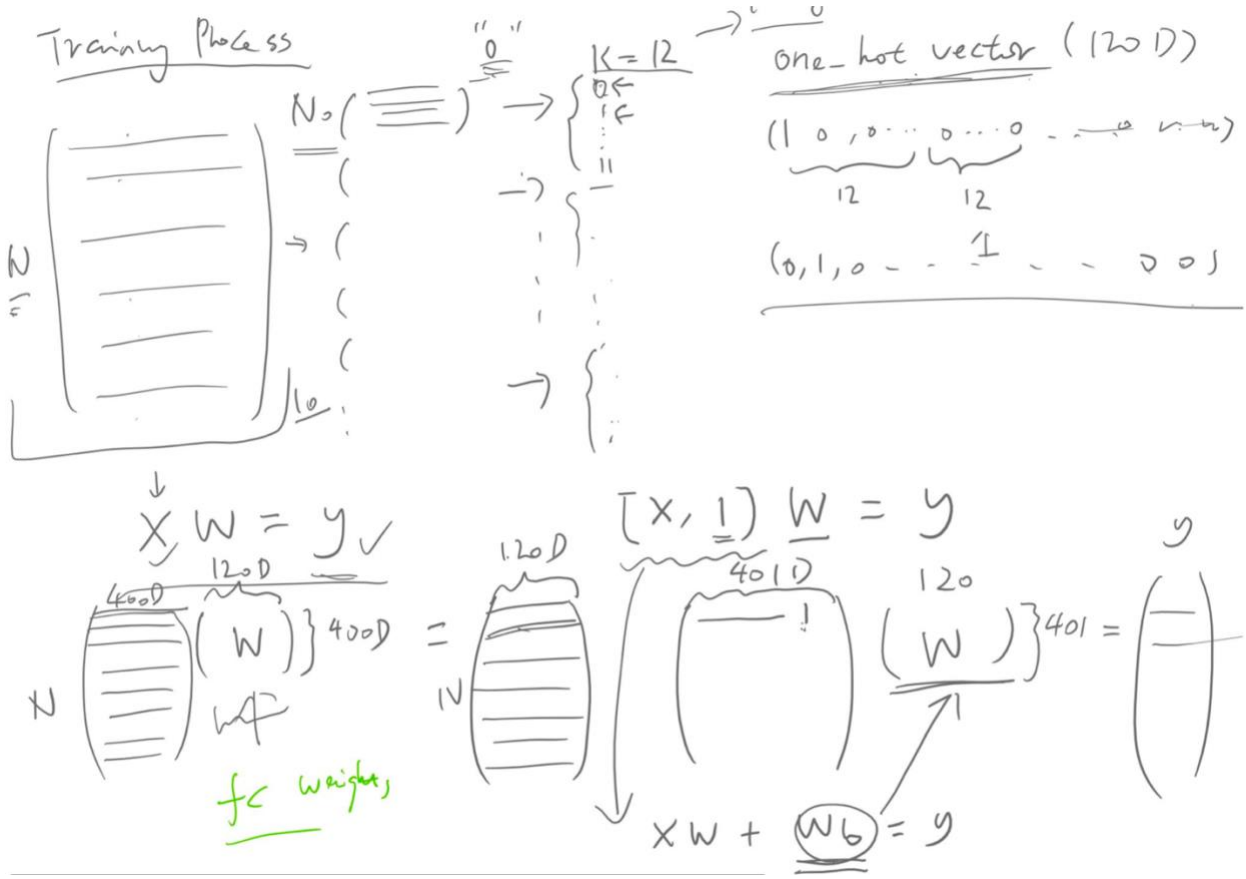
This problem is the ensemble problem and I have referred the github code online for the same. To do the ensemble, change some input data, don't need to change the algorithm shown, and collect the final result i.e. 20 prediction vectors and do the classification part. Try different features for the ensemble, which means to collect coefficients using the code and input different feature subset to the FC layer using some part of the code to do that or change some kernel size or number of kernels that can change for the current version. This has helped to simplify the implementation.

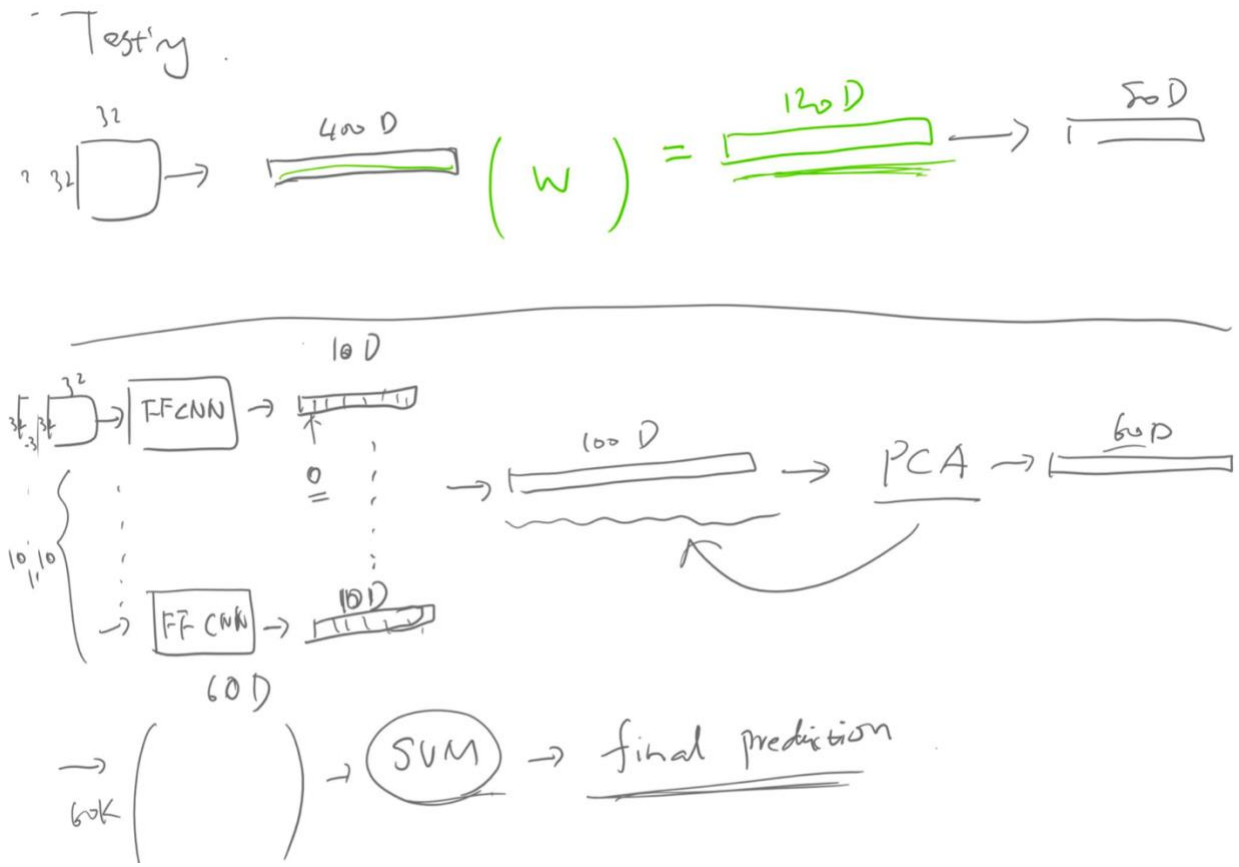
In this problem, apply an FF-CNN to solve handwritten digits recognition. Train an FF-CNN using the 60,000 training images from the MNIST dataset. Adopt the LeNet-5-like architecture where the filter numbers of the first- and the second-conv layers and the first- and the second-FC layers are 6, 16, 120 and 80, respectively. The spatial size of the transform kernels is 5x5 and the stride is 1 for each conv layer. To reduce the spatial dimension, max-pooling layer is adopted.

- Report the training and testing classification accuracy for individual FF-CNN on the MNIST dataset.
- One way to improve the performance is building the ensemble systems of FF-CNNs. Train ten different FF-CNNs and ensemble their results following the method. Diversity is the key to have successful ensembles, and paper introduces three strategies to increase diversities in an ensemble of FF-CNNs which can refer to. Explain and justify the strategies to generate various FF-CNNs in an ensemble and report the training and testing classification accuracy of the ensemble system.
- Error analysis: Please compare classification error cases arising from BP-CNNs (use best result in HW#5) and FF-CNNs. What percentages of errors are the same? What percentages are different? Please give explanations to your observations. Also, please propose ideas to improve BP-CNNs, FF-CNNs or both and justify the proposal.

For the FC layer construction, the input for the FC layer, say we use LeNet5 architecture for the input 32x32, after 2 stages of the transform get 5x5 spatial size and 16 spectral size, changes the image to become feature vector. Here 1 is DC and rest 15 is AC. The final representation after the two stage transform is shown below. Finally need to shape it to become a 1D vector, so 400D feature vector for each input images. So we talk about the training process and also to learn all the parameters in the FC layer. For the number of images we have, there is a data matrix. First thing to do is the k means, and k means for each class. Since this is training, we already have the label for each data vector, so we can just group it to become 10 data matrix.

For example, we have 0 and then we apply k means on the 400D feature vectors. And for the parameter settings, we want to reduce the dimension become 120D after the 1 FC layer so set the number of k equal to 12. And we have a Saab category for the original, the 0 object class. This shows the k means result and we repeat the k means 10 times to generate the sub class. Next step is get the one hot vector. We have the one hot encoding process. Actually we need to prepare 120 one hot vectors corresponding to each Saab class. The reason to prepare the one hot vector is we use data as target for the LSR computation. We build equations to learn the weights for the Fc layer. Say below XW = y. Rows are say 400D and column is 120D for W. For X, both are N, 400D. Then for y, Nx120D. y is the target vector from the one hot vector. For each data sample, we know it will be aligned to which sub class and find the corresponding one hot vector and put data below. Thus we have a lot of samples and the corresponding one hot vector, so we know y and X and do some matrix inverse and calculate W. W is the FC layer weight. Here we finished the training part for one stage of FC layer. If want to add bias, one common step to do is like extend the X with another new dimension and set it to 1, and again sensing it with W to get y, but out here for each feature vector we have extra dimension 1. So X would have 401D feature vector. Corresponding to this the weight matrix will become rows are say 401D and column is 120D, and y doesn't change, it is the original y. So extend the original feature to get another dimension and set the value to be 1, to get the bias term. So the new equation becomes XW + Wb = y, so Wb is like a 1D vector, which is bias term. So this was about the training process for the FC layer. Now to do the testing for the FC layer, input will be like some testing images and go over 2 stage Saab transform, use all the weights and bias got from the previous computation and finally get the 400D feature vector. When do the testing, no need to do k means anymore, we just require W. So we do matrix multiplication using the 1D vector and W, so it will give the result which is 120D feature vectors. This is the output for the first FC layer. Repeat this process for the second stage, for which we need to change to become 80D. So the data matrix is changed to become 120D and change K to become 8. And for the final stage, don't require k means, just use the ground truth to generate the one hot vectors. The diagramatic form of the explanation above is shown below.


Training Process
"0"
K = 12
one_hot vector (1201)
(1 0 , 0 ··· 0 ··· 0 ···
12
12
(0, 1, 0 ··· 1 ··· 0 0)
X W = y
1200
400D
400D
N
fc weights
[X, 1] W = y
1201
120
401
XW + Wb = y

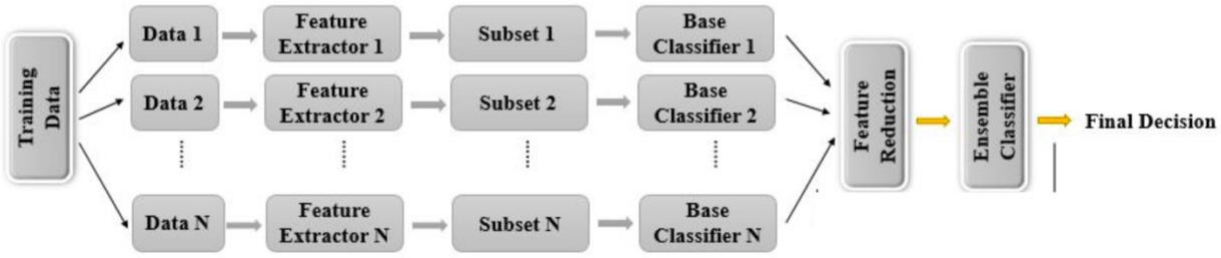


Next part shown above is for the ensemble part for problem 3. For each FF CNN, each 32x32 input data will become 10D vector, can treat this 10D vector as prediction vector. And each dimension corresponding to one class, say first one corresponds to 0 and value here shows the probabilities to belong to class 0 and so on. If we have a lot of different FF CNN, we will have a lot of 10D vectors, so the simple way to fuse is to concatenate along the vectors. Say there are 10 different FF CNNs used, so will have 100D new feature vectors. From here we can have some standard classification process. So first we try to reduce the feature dimensions and try to improve the performance so use another PCA to apply to this 100D feature vector to do the feature reduction and may be get some reduced dimension like 60D. After this, need to fit this 60D feature vectors, use all of the training data, so all the data goes to the SVM classifier, and use some building functions in the SVM classifier and it will give the final prediction. Then can evaluate the results, like with the testing accuracy and training accuracy.

Testing of an FF-CNN

- ➢ Transform images using learned Saab kernels
- ➢ Use learned Kmeans and LSR
- ➢ Output: 10 prediction vectors

❖ Ensemble method: fuse of prediction vectors



For the python, use some sklearn package, for the PCA computation and also for the SVM classifier. In the problem, do not need to worry about the parameters in the SVM classifier, use the default settings. SVM computation does take some time, so no need to get the optimal parameters inside the SVM.

New features: concatenate all 10D prediction vectors

Feature reduction: PCA

sklearn.decomposition.PCA¶

class sklearn.decomposition. **PCA** (*n_components=None, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None*) [source]

Ensemble classifier: SVM

sklearn.svm.SVC

class sklearn.svm. **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) ¶ [source]

Using the training dataset, learn all the parameters in the FF CNN, apply that and learn the FF CNN on the testing data and collect all the prediction vectors. Also use the training dataset to learn the PCA for the feature reduction and also learn the SVM classifier and apply those learned PCA and learned SVM classifier to the testing. This will give the final result.

Testing

- Compute 10D prediction vectors for all FF-CNNs
- Concatenate
- Apply PCA
- Apply ensemble classifier

Evaluation:

- Accuracy: # correct samples/ # total samples
- Error analysis

The ensemble model is as follows:
1. Run all three files – getkernel.py, getfeature.py, getweight.py.
2. Comment out normalization in getweight.py.
3. Run the Mnist data test set using mnist_test.py for testing accuracy. Comment out normalization in mnist_test.py as well.
4. Perform and repeat this for 10 different settings.
5. Each time when run, save the feature variable in mnist_test.py and getweight.py. These represent the training features and the testing features.
6. Write code to concatenate all features side by side so your total rows are 60000 i.e. the number of data points and total columns are 100 (10 settings each having 10 features).
7. Run PCA.
8. Run SVM.

Also, because of the memory usage problem, can do training of the convolutional layer parameters using 10,000 training data samples and for the training of the FC layer parameters, use full 60,000 training data samples.

III. Experimental Results

The training and testing classification accuracy for the individual FF-CNN on the MNIST dataset is given below.

Training accuracy – 97.23%
Testing accuracy – 96.40%

The training and testing classification accuracy for the ensemble FF-CNN system on the MNIST dataset is given below.

	FF1	FF2	FF3	FF4	FF5	FF6	FF7	FF8	FF9	FF10	Ensemble
Filter size	(3,3)	(3,5)	(5,3)	(5,5)	(5,5)	(5,5)	(5,5)	(5,5)	(5,5)	(5,5)	-
Kernel number	(5,15)	(5,15)	(5,15)	(5,15)	(6,18)	(8,19)	(6,19)	(8,18)	(9, 19)	(9,15)	-
Accuracy	97.30	97.28	97.06	97.12	96.81	96.48	97.14	96.99	97.65	97.15	97.81

Training accuracy – 98.76%
Testing accuracy – 97.81%

IV. Discussion

I have referred the code mentioned in the link above for implementing problem 3. There are different python files in the MNIST_FF folder. The getkernel_compact.py is used to get the DC and AC kernels. It imports and loads the training data and labels as well as testing from the MNIST dataset. It uses the function Saab to create multi stage Saab transform. The key parameters to use are the kernel size, kernel number i.e. AC number. Tune these two parameters for this problem as well as other parameters in saab.py too. Output for this function is the parameters of the PCA and also the parameter for the conv layer, so it saves it in a dictionary for later use. Pca parameters are stored in a pickle file in the folder.

The second code is getfeature_compact.py which is used to get features. The feature which is output for the two stage Saab transform. It consists an imread image part where the 4 png image files provided are taken as input i.e. training images here. Again, here it loads some data and also load the parameters to calculate through the previous files. Then use the saab.initialize function to give input data of training images (9,0,5,3) and another input is the pca parameters. This function will give the final output feature of some dimension. Then save this feature in the pickle file for later use. The stride parameter is taken as 1. Max pooling is also used after both the convolutional layers to reduce the dimensions.

The last code getweight_compact.py gets the weight for the FC layer. First load the feature calculated in the previous file and reshape the feature to become a 1D vector so as the input for the FC layer. Further k means is used to compute the sub-class, different clusters. Then compute the one hot vector part. Also compute the least square regression part. After we solve this, we get the weight. So this weight and bias is for the FC layer. And finally find some training accuracies. Also then save the weight and the bias. Basically, all these 3 files try to calculate the weight for the conv layer, FC layer, and also can learn the training accuracy here. One part added by me is I modified the file to add the testing accuracy. For MNIST dataset, there are 60k images, which is quite a large dataset, can also run 10k images, depending on the computer processor. For the kernel for the Saab transform actually gets a good filter weight but may require full dataset to get reasonable results.

Thus, the training and testing accuracy for the individual FF-CNN were calculated above.

The ensemble method fuses the output decision vectors of multiple FF designed CNN to achieve the image classification problem. To increase the performance of this ensemble system, we need to enhance the diversity of the FF designed CNN. It can be done in three ways: a) different parameter settings in the convolutional layers, b) flexible feature subsets fed into the FC layers or subsets of derived features, c) multiple image embeddings of the same input score or flexible image input forms. Another idea to enhance the performance would be to cut the data set into very easy, easy, hard, difficult based on the decision confidence scores. Thus then build different ensemble systems depending on the type of data samples and increase the classification accuracy.

The network parameters of the target layer are derived in the one pass FF CNN, based on the statistics of the output data from the previous layer. There is no back propagation used in this method. The training complexity is reduced than the BP CNN. The actual performance of FF CNN is however lesser than the BP CNN in terms of accuracy. So to enhance it more, we use the ensemble system which make use of multiple FF-CNNs to enhance the classification accuracy.

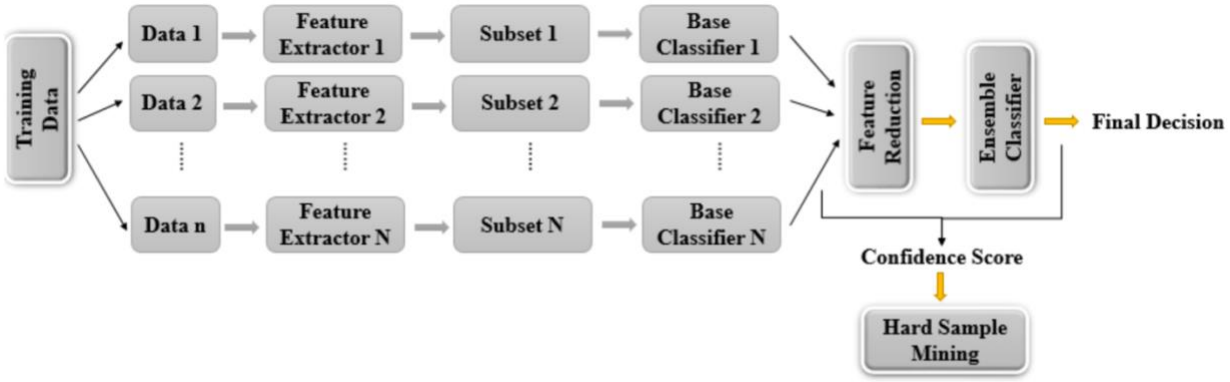


Fig. 1. Overview of the proposed FF-CNN ensemble method.

Firstly, to achieve higher performance, in the FF CNN design, channel wise PCA is applied to the spatial outputs of the convolutional layer. This is done to remove the spatial dimension redundancy, which helps to decrease the dimensions of the feature vector. Secondly, ensemble systems are developed using multiple FF CNNs as base classifiers. To better the performance of this ensemble system, 3 diversities are used which were mentioned above. Thirdly, separate the easy and hard ones in the data set depending on the confidence score which is calculated based on final decision vector of the ensemble classifier.

FF CNN consists of two parts in cascade: convolutional layers and fully connected layers. The convolutional layers are unsupervised as there are no labels involved in the construction process. It basically does the following functions – subspace approximation via spectral decomposition and maximum pooling in spatial domain. This subspace approximation is done using Saab transform followed by PCA. The Saab features are calculated through the two conv layers and feature discriminant power is enhanced because of a large receptive field.
The FC layers have a multi-stage linear LSR (least squared regressor). This is followed by clustering to generate the output of one hot vector. The index of the output space dimension defines a pseudo label. Ensemble method is implemented wherein multiple weak classifiers are integrated to make a stronger one. This can be implemented using bagging, stacked generalization, random forest. For the ensemble system to become successful, main key is diversity.

Saab transforms already reduce the redundancy in the spectral domain and max pooling. But further dimension reduction is done using the PCA to the spatial dimensions at each filter known as channel wise PCA. C-PCA is thus implemented to reduce the original high dimension to lower dimension. For the ensemble method,

use the LeNet5 type CNN, which is 2 conv layers, 2 FC layers and one output layer. Now use multiple FF CNNs as the first stage base classifiers and concatenate the output decision vectors, whose dimension is the same as the class number. Use PCA to decrease the feature dimension before going into the second stage ensemble classifier. Now to enhance and increase the diversity of the ensemble system, we use the following steps:

a) Changing the parameters in the conv layer – choose various different filter sizes. The filter spatial dimension for the two convolutional layers is the same. Use 4 different combinations of spatial dimensions i.e. (3x3,3x3), (3x3,5x5), (5x5,3x3) and (5x5,5x5). This results in different receptive field sizes. Thus, this leads to different features at the output side.
b) Subsets of derived filters – We have two feature sets got from the two conv layers – Fconv1 and Fconv2. We select a subset say Vi from these Fconv. We take the below 3 selection rules and test them. First is for each channel in Fconv1, select features randomly of the corresponding spatial dimensions of the first convolutional layer, perform C PCA to decrease the feature dimension to say K1. Then randomly select some features from K1. Secondly, apply C PCA to Fconv2 layer to get the K2 features and again randomly select some features from K2. Thirdly, implement checkerboard partitioning of Fconv1 in the spatial dimension, performing C PCA again to produce two feature subsets. Thus we get one decision vector for each feature subset.
c) Flexible input image forms – We use this to higher the diversity, like use different color models to show the color images, use different color spaces like RGB, YCrCb, Lab as different inputs to the FF CNNs. We implement the Laws filter as well so as to store the different spectral characteristics. Thus by combining these FF CNNs which have had different input forms, we produce the final decision vector.

Thus, in this manner, perform the 3 parts above and the ensemble system is able to get a better performance.

My strategy was using the a) method, where I tried different filter sizes and kernel sizes and kernel numbers and with 10 different combinations, I used this ensemble system to determine the output decision vector, giving the training and testing accuracy above.

For the ensemble classifier, use the Radial Basis function (RBF) SVM classifier. PCA is used to cascade the decision vectors got from the base classifiers of the ensemble system before the training of SVM, which leads to a reduced feature dimension.

(Source: https://en.wikipedia.org/wiki/Confusion_matrix)

Classification error – this is dependent on the number of samples incorrectly classified (false positives and false negatives) and is stated by the formula below:
Ei = f*100/n,
where f is the number of sample cases incorrectly classified, and n is the total number of sample cases.
This is done with the help of confusion matrix, also known as error matrix. Each row of the error matrix shows the instances of the predicted class and the column corresponds to the instances in an actual class. It forms a table 2x2, having rows and columns corresponding to true positive, false positive, true negative, false negative.

The percentages of errors are same for some, they are different for others. It corresponds to the error having made the prediction wrong for which digit having the highest error in predicting.
I have compared the percentage of errors for predicting the correct digit and have written a code for the calculation of the same for the BP CNN as well as FF CNN, and I have come to the following conclusion.
The percentage of errors are almost the same for the digit 9. Since the percentage of error is the same, we observe that both the models BP CNN and FF CNN work in the similar manner to predict this digit. Here, we can say that the feature extraction of the FF CNN is quite similar to the BP CNN. Hence it leads to similar error on the same digit.
The percentage of errors are different for the digit 4. Since the percentage of error is the different ie the digit is predicted correctly in BP CNN however has more or higher error in prediction for the FF CNN model, we observe that both the models BP CNN and FF CNN work in the a quite different manner to predict this digit. This happens because the features calculated for this digit in the ensemble system weren't good enough for the FF CNN model to predict the correct output decision vector, whereas the weights initialized in the BP CNN model for the features got there could predict the digit in a much better way, thus predict the output decision vector correctly and giving a small value to the classification error. Here the error is different for the two models for the same digit.

Also, we can see that the classification error we get from the confusion matrix for a particular digit, when we get a similar value of percentage error for another digit, we observe and can say that these two digits have quite similar features say, for the digit 1 and 9. These two digits have similar features like a standing line, hence in the confusion matrix we saw that the digits 1, 7 and 9 are predicted wrongly as each other for some output vectors. Also, we see similar kind of observation of such feature similarity in other digits like 0, 6, 8 and also in digits sometimes 3 and 5.

(Source: https://machinelearningmastery.com/improve-deep-learning-performance/)

Ideas to improve BP-CNNs, FF-CNNs, or both are stated below:
Improve performance with data, improve performance with algorithms, improve performance with algorithm tuning, improve performance with ensembles.
Improving performance can be increasing performance or higher testing accuracy or faster implementation.

a) Improve performance with data – i) get more data – Deep learning models and neural networks become better when trained and tested with more data. However, this goes on increasing the performance till a certain instance. ii) invent more data – this is also called data augmentation or data generation. Here, create more data which will help to train or test on the model, so the performance increases as well as the accuracy. iii) rescale the data – do this to the bounds of the activation functions. Say for example, if using sigmoid on the output to predict binary values then normalize the output values to be binary. Also should create different versions of the training dataset, like normalize to 0 to 1 or rescale to -1 to 1. Then calculate the performance. It causes it to go higher. iv) transform the data – what kind of pre-processing would help to simplify or make the accuracy higher or perform faster. v) feature selection – keeping only the necessary needed attributes and implementing good feature selection methods because features are very important to determine the distinguishing power between the output decision classes. vi) reframe the problem
b) Improve performance with algorithms – i) spot-check algorithms – Use the methods which are usually used and are the top ones, like linear methods such as logistic regression, LDA, tree methods such as decision tree and random forest, SVM, k means clustering. Using these common techniques on the dataset and finding the one with the best performance and then trying to optimize it. ii) steal from literature – Reading a lot of research papers and publications and trying to code and implement the methods depending on the type of dataset. iii) resampling methods
c) Improve performance with algorithm tuning – Various parameters as stated below can be fine tuned to get higher performance, as each of these parameters are used to build the model and have a hand in producing the output vectors or determining the input flow as well as the other decisions made in the model. These parameters are diagnostics, weight initialization, learning rate,

activation functions, network topology, batches and epochs, regularization, optimization and loss, early stopping.

d) Improve performance with ensembles – i) combine models ii) combine views iii) stacking – stacked generalization

Also, can use dividing the data into easy and hard and then designing the network according to the kind of data set it is a part of and then performing the implementation.

I personally think with experience by working on different models as well as architecture parameters and different settings, and by reading research papers and keep a track and grasping the new technologies coming up daily in the field of deep learning, we can implement and improvise the CNN model depending on the kind of data set as well as the architecture chosen and the application it is for.

Thank you so much for this EE 569 – Digital Image Processing course!!