

EE 569: Homework #6

Issue: 4/08/2019 Due: 11:59PM 04/28/2019

Name: Wenjun Li

USC ID: 8372-7611-20

Email: wenjunli@usc.edu

Submission Date: 04/28/2019

Problem: Feedforward CNN Design and Its Application to the MNIST Dataset (100%)

(a) Understanding of Feedforward-Designed Convolutional Neural Networks (FF-CNNs) (15%)

1. Motivation

Normal CNN is designed to learn features from dataset by updating parameters backwards, which costs a great computational power and a long training time. So, researchers proposed this novel feedforward designed CNN to conquer above problems.

2. Approach

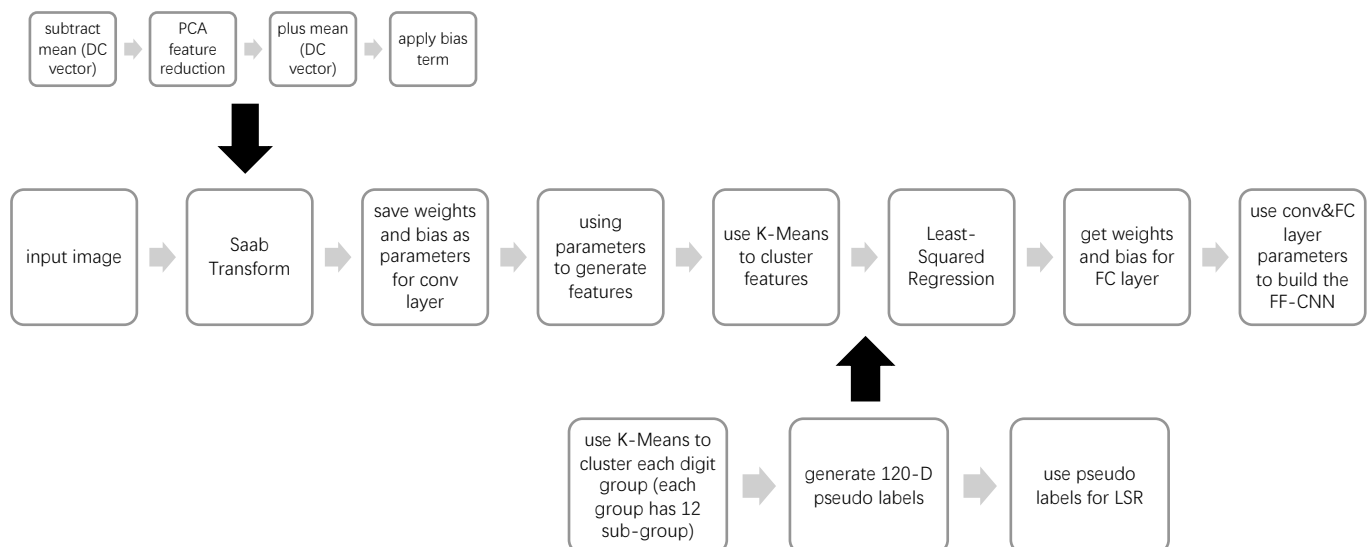
An FF-CNN has 2 modules in cascade:

- The construction of convolutional layers using Subspace approximation with adjusted bias (Saab) transform
- The construction of fully-connect (FC) layers using multi-stage linear least squared regressor (LSR)

In other words, the problems are how to compute parameters in conv layers and how to compute parameters in FC layers.

Here, I give a flow chart of FF-CNNs design and I will explain below.

Note, in this flow chart, I break the whole process into 3 parts: main process, Saab Transform and Pseudo Label Generation.



First, we use multi-stage Saab Transform to do convolutional layers construction.

(Before we start, we can separate the transform kernels (anchor vector) to 2 categories: DC anchor vector and AC anchor vector.)

- Saab transform can derive the weight parameters in the conv layer. MNIST dataset has 60,000 training images and 10,000 testing images and the images in MNIST have a size of 32×32 . We assign the kernel size to be $k \times k$ and stride to be s . So, we can get a filter image patch with size $p = (\frac{32-k}{s} + 1) * (\frac{32-k}{s} + 1)$.
- Next, we do PCA feature reduction to mean-removed training images (sample mean and patch mean are removed before we do PCA). After above operation, we now have a matrix with size $(60000, p \times k \times k)$ and we do PCA to it to obtain AC anchor vectors. Suppose we get M vectors (kernels) and each of them has a size of $k \times k$.
- The DC vector could easily be obtained by computing

$$mean = \frac{1}{k \times k} (a_0 + a_1 + \dots + a_{15})$$

- After we getting all the parameters (AC and DC vectors) from above steps, we need to select bias to avoid sign confusion. The simplest way to select suitable bias is by computing

$$bias = -\min(\sim)$$

in this way, we can make all the outputs to be non-zero, so that we can avoid sign confusion.

Above operations consist of one stage Saab transform. In order to construct FF-CNNs, we need to repeat this one-stage Saab transform several times to build a multi-layer network.

Second, we need to construct FC layers using multi-stage rectified least-squared regressor (LSR).

In order words, we need to connect the information we obtained from above operation with “pseudo-labels”.

So, what is pseudo-labels? As we know, the labels of MNIST is one digit from 0-9. Suppose we get a $5 \times 5 \times 16$ vector from an image in MNIST and we want to connect it with a label and we want them have the same dimension. Then, we need to generate “pseudo-label”, which has the same dimension as the vector we get from Saab transform. Since $5 \times 5 \times 16$ is a 400-D vector and it is too large, so, we first do PCA feature reduction and reduce it to 120-D vector. Therefore, we need to generate a 120-D “pseudo-label” using the original MNIST label.

How could we do this? Because different person has different writing styles, so, we can divide each digit group in MNIST into 12 groups. For example, there could be 12 different style of digit “0”. So, there will be a total $12 \times 10 = 120$ groups. We apply K-Means clustering (setting $k=12$) to a single digit group, let's say group “1”, and K-Means will cluster them into 12 groups. After we get 120 groups of labels, we encode them to be one-hot label. For example, the first group in group “0” should be $[1 \ 0 \ 0 \ 0 \ 0 \dots 0 \ 0 \ 0]$, the second group in group “0” can be $[0 \ 1 \ 0 \ 0 \dots 0 \ 0 \ 0]$. Note, the one-hot labels here are 120-D.

Hence, we can connect the 120-D feature vector with 120-D pseudo-label vector correspondingly. By applying the following equation we can start to train the FC layer network by solving LSR:

$$WX + b = y$$

where W is the weight vector, X is the input feature vector, b is the bias vector, and y is the pseudo-label vector.

Finally, by training the network to solving this LSR problem, we can train out network and achieve a good performance.

3. Result

No result required for this part.

4. Discussion

Question: Explain the similarities and differences between FF-CNNs and backpropagation-designed CNNs (BP-CNNs)?

Differences:

The biggest difference between FF-CNNs and BP-CNNs is that FF-CNNs does not need to update its weights (kernels) and bias through backpropagation and there is no need to train those parameters. In BP-CNNs, those parameters are randomly initialized and they need to be updated by backpropagation in order to make them powerful enough to extract features from images. However, in FF-CNNs, we use multi-stage Saab transform to get useful parameters directly, which saves a significant amount of computation power and computation time.

Another difference is that whether the operation is interpretable. BP-CNNs is like a black box and people cannot explain what's going on in the black box and there is no science to design BP-CNNs. Most of the BP-CNNs are empirical and non-interpretable. However, FF-CNNs is interpretable. For each step, we know exactly what each step is doing and we know the science behind the operations.

Similarities:

Both FF-CNNs and BP-CNNs should be multi-layer so that they can be powerful enough; both of them use kernels to extract features; both of them use FC layers after conv layers.

(b) Image Reconstructions from Saab Coefficients (35%)**1. Motivation**

Not required in this part.

2. Approach





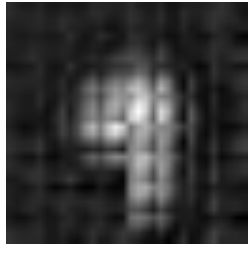
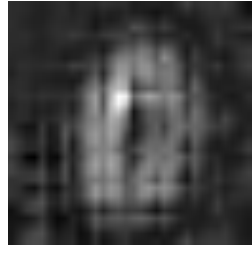
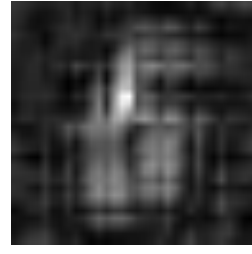
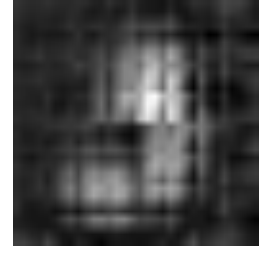
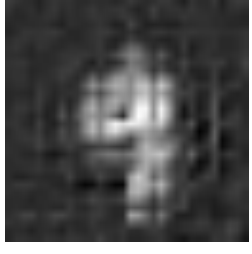
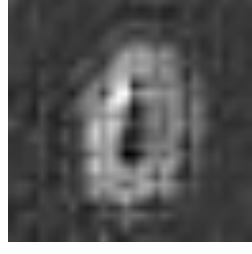
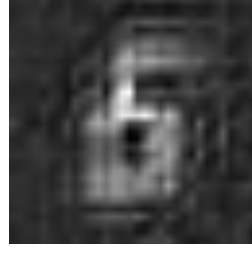
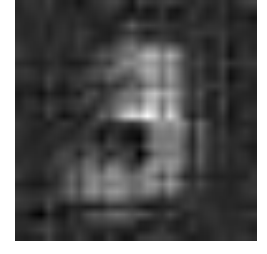
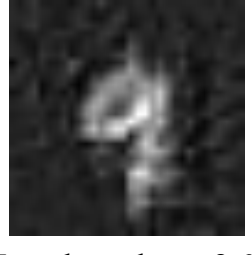



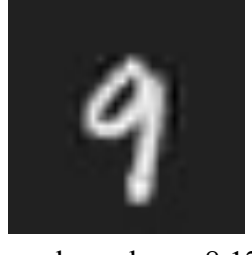
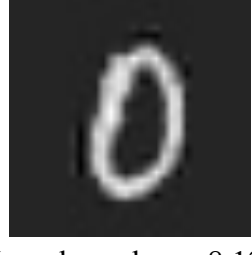


Not required in this part and is described in last part.

3. Result

The spatial size of the transform kernels is 4×4 , and the stride of each stage is 4 (non-overlapping). So, the output dimension of Saab coefficients of an image is $2 \times 2 \times N$, where N is the product of k_1 (AC kernel number in the 1st stage) and k_2 (AC kernel number in the 2nd stage). Here, I set $k_1 = 5, 7$ and $k_2 = 15, 31, 63, 127$. So, I have 4 different kernel number setting pairs, which are (5, 15), (7, 31), (7, 63), (7, 127).

The reconstruction result images are the results are shown below. Note that, in the below table, I combine AC kernel number and DC kernel number (DC is always 1), so, the kernel number pairs are (6, 16), (8, 32), (8, 64), (4, 128)

| | | | |
|-------|-------|-------|-------|
| 1.png | 2.png | 3.png | 4.png |
|-------|-------|-------|-------|

| | | | |
|---|---|--|---|
|  |  |  |  |
| Original image | Original image | Original image | Original image |
|  |  |  |  |
| Kernel number = 6,16 PSNR=14.87 | Kernel number = 6,16 PSNR=13.63 | Kernel number = 6,16 PSNR=15.10 | Kernel number = 6,16 PSNR=13.74 |
|  |  |  |  |
| Kernel number = 8,32 PSNR=15.83 | Kernel number = 8,32 PSNR=14.31 | Kernel number = 8,32 PSNR=16.13 | Kernel number = 8,32 PSNR=14.63 |
|  |  |  |  |
| Kernel number = 8,64 PSNR=17.75 | Kernel number = 8,64 PSNR=16.00 | Kernel number = 8,64 PSNR=18.49 | Kernel number = 8,64 PSNR=16.56 |
|  |  |  |  |
| Kernel number = 8,128 PSNR=28.66 | Kernel number = 8,128 PSNR=26.45 | Kernel number = 8,128 PSNR=25.26 | Kernel number = 8,128 PSNR=26.33 |

Evaluation:

As you can see from above table, I increase kernel number gradually and I compute their corresponding PSNR. For the reconstruction image with kernel number (6, 16), it has least PSNR and worst quality. When I improve the kernel number, the PSNR and image quality begin to improve. When I use (8, 128) kernel number, the reconstruction image quality is pretty good.

From this comparison, we can say that the larger kernel number we use in Saab transform, the better reconstruction quality we have. This is easy to understanding, when we use more kernels to extract the feature, the more information we can get from the original image. Hence, we can reconstruct image with more features and information we have.

4. Discussion

No discussion required for this part.

(c) Handwritten Digits Recognition using Ensembles of Feedforward-Design (50%)

1. Motivation

The kernel weights in convolutional layers of an FF-CNNs will remain the same along different training, because they use the same hyper-parameters. So, in order to improve prediction accuracy, we could use ensemble FF-CNNs. To be specific, we can use many single FF-CNNs and ensemble them together to do prediction, so that we can greatly improve testing accuracy while reduce loss.

2. Approach

Constructing a single FF-CNNs is similar as the last part, what we need to do to build a ensemble FF-CNNs is to design several single FF-CNNs and combine them together to do prediction. For example, we can design 10 single FF-CNNs and combine them together at last. Since we have 10 single FF-CNNs, we can let them vote at the end of prediction so that we can reduce loss and improve prediction accuracy. So, how do we train the ensemble FF-CNNs?

To train the 10 single FF-CNNs separately, we can split the entire MNIST dataset into 10 small dataset and use them to train single FF-CNNs separately. Since each FF-CNNs is trained based on different dataset, they will have parameters and the ensemble network will be more robust. When we combine them together at last and let them vote to predict the best choice, the performance and the robustness of the network is thus greatly improved.

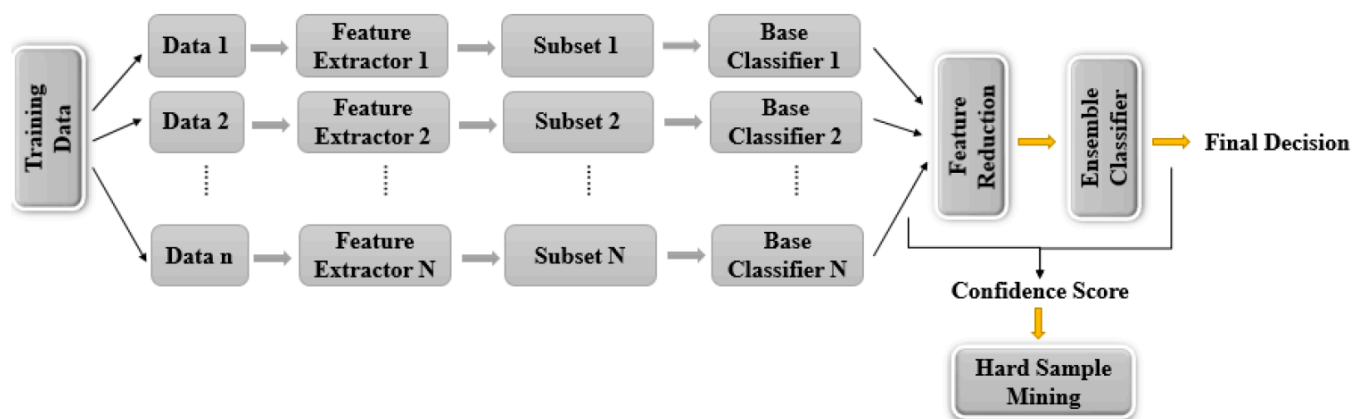


Figure 1 Ensemble FF-CNNs^[4]

There are actually several ways to train ensemble FF-CNNs. Instead of using separated training dataset to train different individual FF-CNN, we can differentiate FF-CNNs' structures so that each of them would have different properties and parameters. For example, for the 1st FF-CNN, we use (5, 5) as kernel size and (6, 16) as kernel number; for the 2nd FF-CNN, we choose (3, 3) and (8, 16) as kernel number; ... in this way, we can also train our single FF-CNN individually and combine them together at the end to

achieve a better performance.

In my *Result* part, I used the first way to build my ensemble FF-CNNs, i.e. I split the MNIST into 10 separated small dataset and use them to train FF-CNN individually. Please refer to the result below.

3. Result

Individual FF-CNN on the MNIST dataset

| Individual FF-CNN | Training | | Testing | |
|----------------------|----------|--------|----------|--------|
| | Accuracy | Loss | Accuracy | Loss |
| | 97.03% | 1.5820 | 97.10% | 1.5765 |

I run an individual FF-CNN on MNIST dataset and obtained the result shown in above table. The training accuracy on 60000 images is 97.03% and the loss is 1.5820; the testing accuracy on 10000 testing images is 97.10% and the loss is 1.5765.

To train an ensemble FF-CNN, I split MNIST training dataset into 10 non-overlapping datasets, each of which contains 6000 training images. And I train these FF-CNNs separately and finally combine them together and use SVM to let them do prediction. The results of ensemble FF-CNNs are displayed below.

Ensemble FF-CNNs with 10 individual FF-CNN

| Ensemble FF-CNNs | Training Accuracy | 97.57% |
|---------------------|-------------------|--------|
| | Testing Accuracy | 97.53% |

4. Discussion

The best result in my HW#5

| BP-CNN | Training | | Testing | |
|--------|----------|--------|----------|--------|
| | Accuracy | Loss | Accuracy | Loss |
| | 99.68% | 0.0099 | 99.21% | 0.0276 |

Question: What percentages of errors are the same? What percentages are different?

As you can see from above tables, the training accuracy and the testing accuracy of individual FF-CNN and ensemble FF-CNNs are pretty good, almost as good as BP-CNNs. However, the problem for FF-CNNs is the high loss. The loss of this individual FF-CNN is more than 100 times higher than the loss of BP-CNN. So, what's the same is the accuracy performance, and what's different is the loss performance.

Question: Propose some ideas to improve BP-CNNs, FF-CNNs or both

This FF-CNN perform well on small images, but when we are dealing with large images (for example 256*256), we have to design deeper network. But this FF-CNNs has a inherent problem: when the conv layers are too deep, PCA will possibly eliminate too much useful information. To overcome this problem, we should select relatively larger kernel size, which will retain more energy and information. This is similar to the gradient vanishing in BP-CNNs and we use non-linear activation to tackle that.

Another idea is to choose a suitable kernel number. As you may know, computing the convolution operation cost most of the time in CNN. Besides, some kernels cannot extract very useful information and they should be removed. When we designing the FF-CNN and BP-CNN, we should first try small number of kernels and gradually increase the kernel number if the network performance does not reach our expectation.

My last proposed idea is to combine BP-CNN and FF-CNN. Some ideas are shared in both BP-CNN and

FF-CNN and these ideas could be combined to design a better network. For example, we can first extract features from a simple small dataset and get parameters (kernel weights and biases), then we can plug these parameters into BP-CNN and backpropagation further refine the parameters. Saab transform could take a long time when it deals with large kernel number, large image, etc. By first applying Saab transform on a small dataset, we can save some time. Further refining the parameters using BP can significantly improve network performance. Hence, we should try to combine the advantages of BP/FF-CNN together and let them perform better.

Reference

- [1] 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*.
- [2] [MNIST] <http://yann.lecun.com/exdb/mnist/>
- [3] https://github.com/davidsonic/Interpretable_CNN
- [4] Chen, Y., Yang, Y., Wang, W., & Kuo, C. C. J. (2019). Ensembles of feedforward-designed convolutional neural networks. *arXiv preprint arXiv:1901.02154*.