

Homework 3 for Deep Learning and Data Mining

Xiaoqian Chen; Yewen Huang

June 3, 2019

1 Data Set

Clothes shopping is a taxing experience. My eyes get bombarded with too much information. Sales, coupons, colors, toddlers, flashing lights, and crowded aisles are just a few examples of all the signals forwarded to my visual cortex, whether or not I actively try to pay attention. The visual system absorbs an abundance of information. Should I go for that HM khaki pants? Is that a Nike tank top? What color are those Adidas sneakers?

Can a computer automatically detect pictures of shirts, pants, dresses, and sneakers? It turns out that accurately classifying images of fashion items is surprisingly straight-forward to do, given quality training data to start from. In this homework, we'll walk through building a machine learning model for recognizing images of fashion objects using the Fashion dataset. It contains images of various articles of clothing and accessories, such as shirts, bags, shoes, and other fashion items.

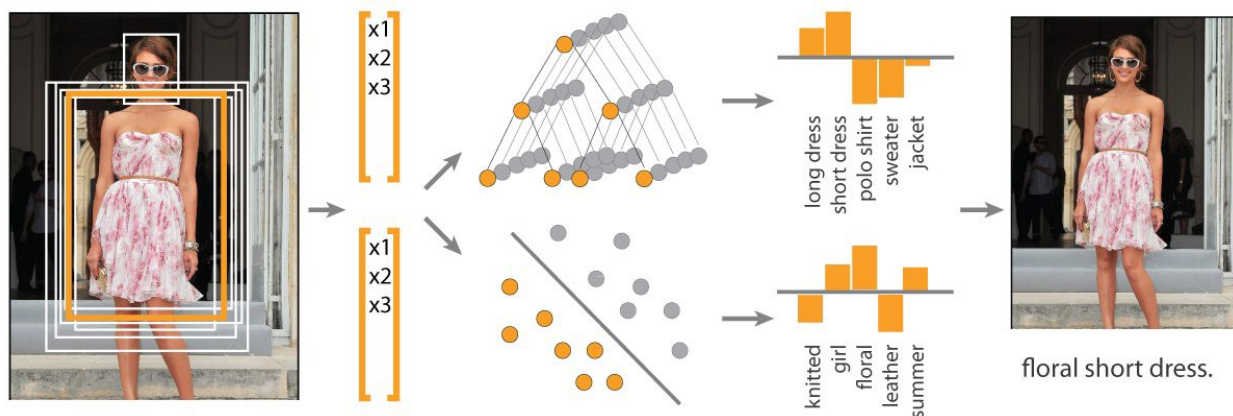


Figure 1: Task

The Fashion training set contains 60,000 examples, and the test set contains 10,000 examples. Each example is a 28x28 gray scale image, associated with a label from 10 classes (t-shirts,

trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots). There are 6,000 examples for each class in the training set.

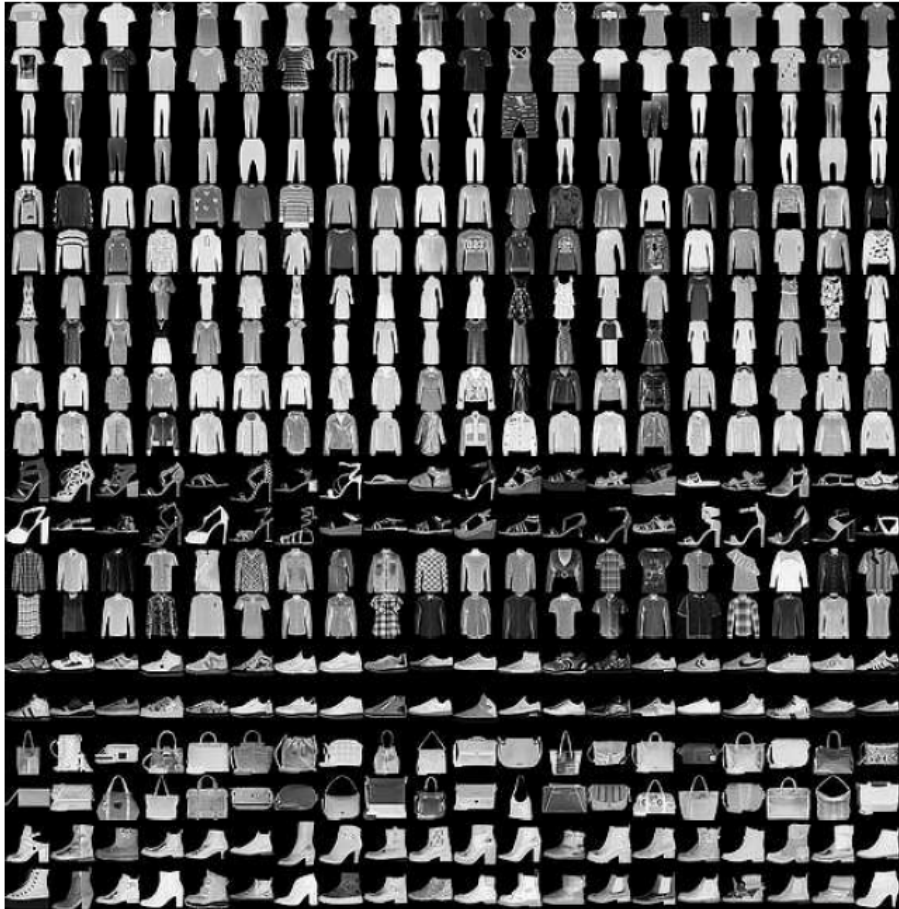
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 2: data examples

The images have been size-normalized and centered in a fixed-size image (28×28 pixels) with values range from 0 to 256, where 0 is black and 255 is white. For simplicity, each image has been flattened and converted to a 1×784 vector. For output layer, there are 10 distinct classes, presented by 10 independent standard basis vector $\{e_1, \dots, e_{10}\}$, where $e_1 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]'$ stand for the first class "T-Shirt/Top" and so on.

Data set description		
	Training set	Test set
Set Size	60,000	10,000
Value Range of Input Descriptors	$\{0, \dots, 255\}$	$\{0, \dots, 255\}$
Size of each Input Descriptors	784	784
Size of each class	6,000	1,000

Table 1: Data set description

A straightforward technique for reducing dimensions is Principal Component Analysis (PCA). The Embedding Projector computes the top 3 principal components. Here we reduce it to 3-dim, which is could be visualization.

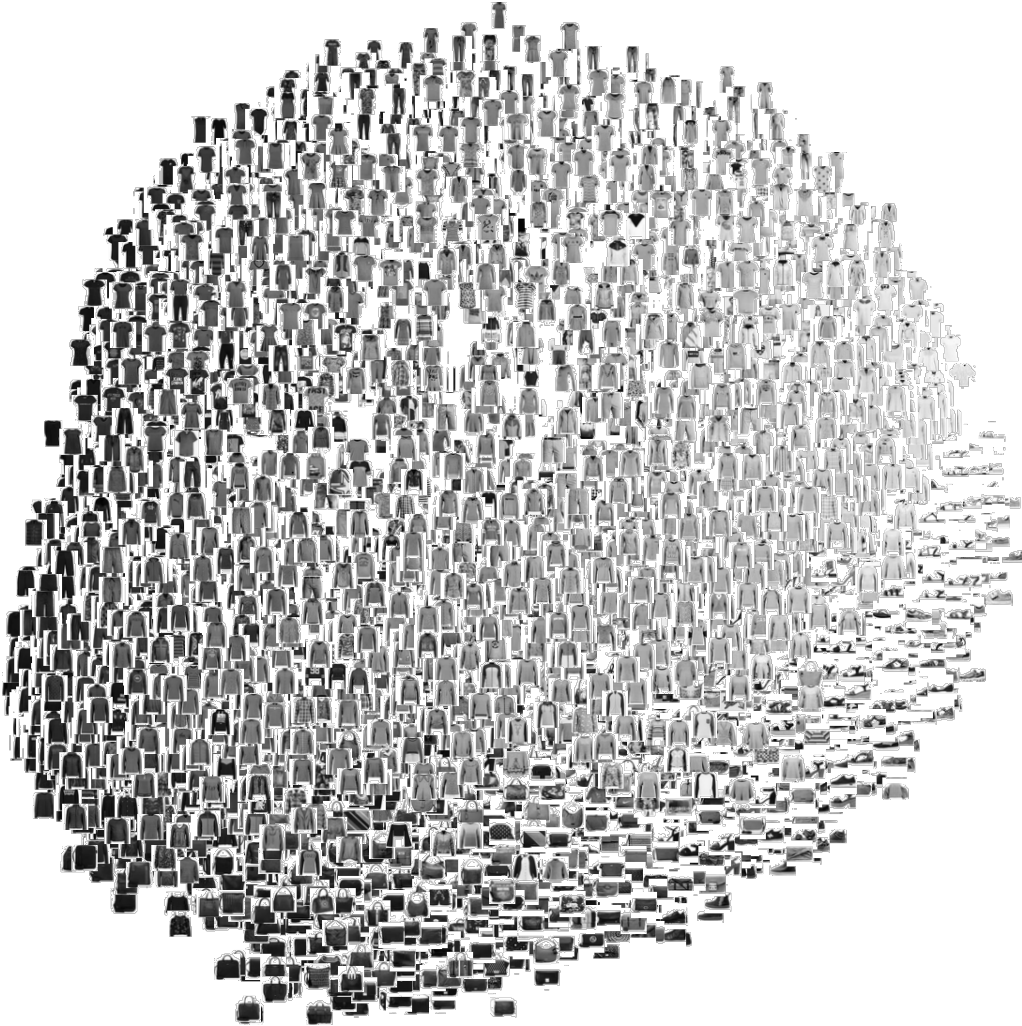


Figure 3: PCA with the top 3 principal components

2 Architecture

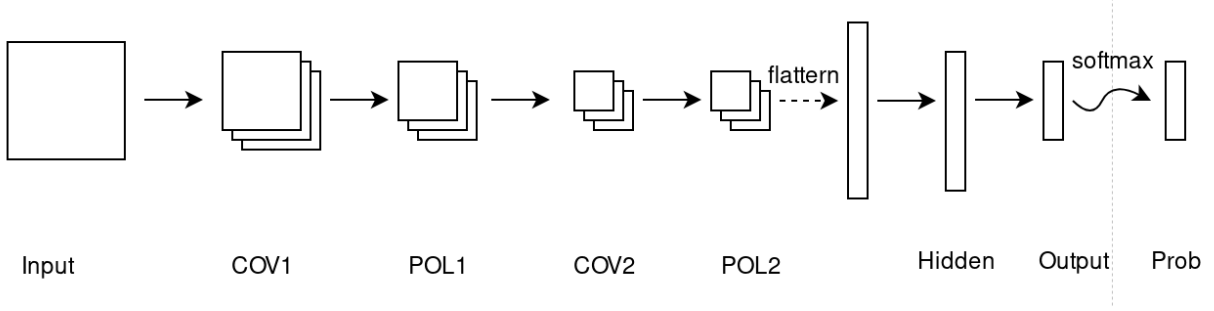


Figure 4: Networks architecture

The architecture for the convolution neural network is presented above. The whole network expresses the mapping between raw image pixel and their class identity. The input data is defined as first layer. The next one is the 1st convolution layer, which consists of the amount of 16 of size 5×5 filter. In general, the convolution layer is used to slide through the width and height of each input image, and compute the product of the inputs region and the weight learning parameters. This will generate another image that consists of responses of the filter at given region. Consequently, we have the 1st pooling layer reduces size to 14×14 of input images as per results of a convolution layer. The 2nd convolution layer applies the number of 32 of 5×5 filter. After applying the same method between the 1st convolution layer and 1st pooling layer, we obtain the 2nd pooling layer in size 7×7 . Then we flatten the output, this is used without parameter. Next, we use flatten layer as our input to do MLP. Utilizing PCA analysis and response function, we get the hidden layer. After hidden layer, we get computed real value output in size 10. In order to get binary vector, we introduce softmax function to get the last layer softmax activation that classifies the 10 categories of data in Fashion dataset.

3 Select sizes for layers

For the first convolutional layer, we set the convolution filters are 5×5 pixels. There are 16 of these filters. Use of zero-padding. We used zero padding of $2 = (5 - 1)/2$, to fit across the spatial dimension of input. There is 1 stride of the sliding window, means we move the filters one pixel at a time. In the first convolution layer, feature maps is $28 \times 28 \times 16$. The first max pooling filter is 2×2 square window. In the first pool layer, feature maps is $14 \times 14 \times 16$.

- Size of filter $F1=5$
- Number of filters $K1=16$
- stride $S1=1$
- the amount of zero padding $P1=2$

- Size of window in pool W1=2

Similarly, the second convolution filters are 5×5 pixels. There are 32 of these filters. And There is 1 stride of the sliding window. In the second convolution layer, feature maps is $14 \times 14 \times 32$. The second max pooling filter is 2×2 window. In the second pool layer, feature maps is $7 \times 7 \times 32$.

- Size of filter F2=5
- Number of filters K2=32
- stride S2=1
- the amount of zero padding P2=2
- Size of window in pool W2=2

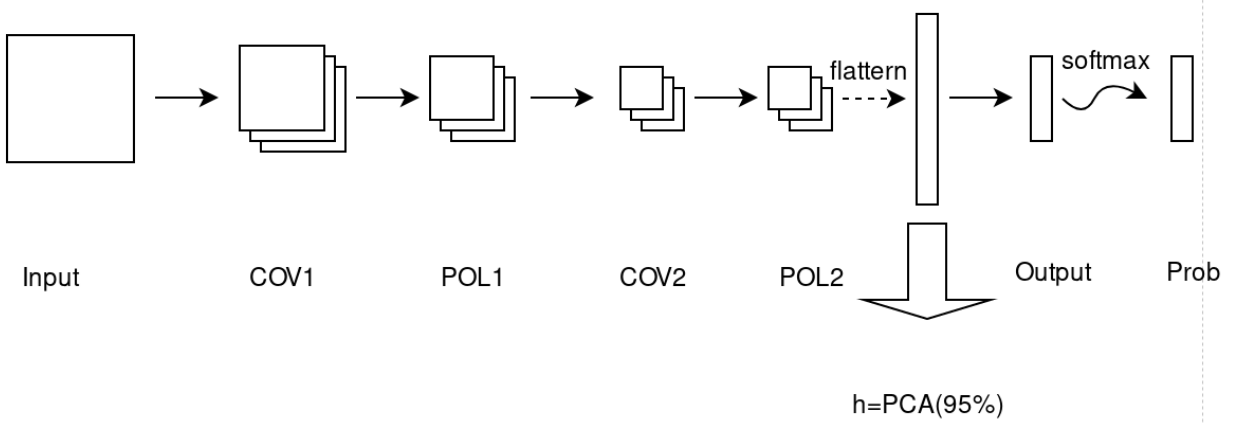


Figure 5: alternative size of hidden layer

After the second pool layer, we flatten the output, it will be in the size of 1×1568 . Apply PCA analysis to the entire training data set, h95 is the smallest number of eigenvalues, that preserves 95% of the total sum of eigenvalues. We choose h95 as the size of the hidden layer.

$$95\% = \frac{\sum_{i=1}^{h95} s_i}{\sum_{i=1}^{1568} s_i} \quad (1)$$

Figure 6 shows the decreasing sequence of eigenvalues of the correlation matrix of the data set. h95 = 151, which means the first 151 biggest eigenvalues preserves 95% of the total sum of the eigenvalues.

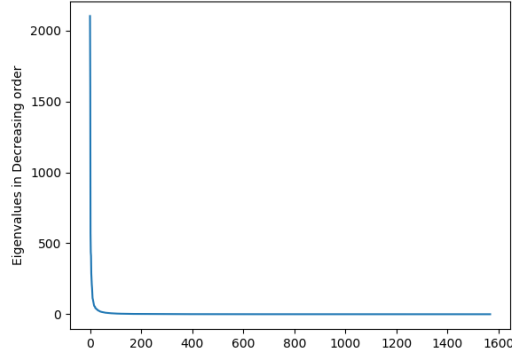


Figure 6: Eigenvalues in decreasing order

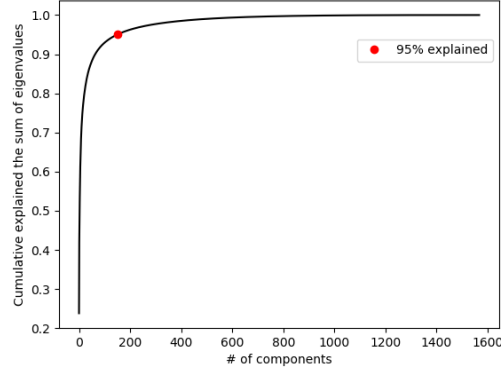


Figure 7: Cumulative Explained the sum of eigenvalues

It intend to the output layer, which is in the size of classes (10 here). The softmax function applied to the output from the output layer, which lead to out final output. Assume we have $O_k = (y_1, y_2, \dots, y_{10})$ from the output layer. There is a probability vector as output of softmax function, such that $OUT_k = (p_1, p_2, \dots, p_{10})$, and $p_1 + p_2 + \dots + p_{10} = 1$.

$$OUT_k = \begin{bmatrix} p_1 \\ p_2 \\ \dots \\ p_{10} \end{bmatrix} = \begin{bmatrix} \frac{e^{y_1}}{\sum_{i=1}^{10} e^{y_i}} \\ \frac{e^{y_2}}{\sum_{i=1}^{10} e^{y_i}} \\ \dots \\ \frac{e^{y_{10}}}{\sum_{i=1}^{10} e^{y_i}} \end{bmatrix} \quad (2)$$

The final classification of X_k is computed from OUT_k by identifying the class have the biggest probabilities.

$$classification of X_k = argmax(OUT_k) \quad (3)$$

Until now, we have $(5 \times 5 + 1) * 16 + (5 \times 5 + 1) * 32 + 1568 \times 151 + 151 + 151 \times 10 + 10 = 239,687$ unknown parameters need to estimate in the network, and $60,000 * (28 * 28) = 47,040,000$

training data, there are approximately $\frac{47,040,000}{239,687} = 196$ data used to estimate each parameter, that is fine.

4 implement automatic training

In this project, we train our neural network by Python3.5 with Tensorflow, which is an open-source machine learning library for research and production.

For initialization of the weights, we use the function

```
init=tf.random_normal().
```

The learning algorithm is use gradient descent to minimize Average CROSS ENTROPY error between computed and true values to get a adapted network. For the Average CROSS ENTROPY error (ACRE), assume that the total number of input examples is N , for k -th example ($1 \leq k \leq N$), we have the final probability output OUT_k after learning and the true class vector $Target_k$ corresponding to the input.

$$ACRE = -\frac{1}{N} \left(\sum_{k=1}^N Target_k \cdot \log(OUT_k) \right) \quad (4)$$

The RELU function is applied in the hidden layer.

$$RELU(x) = \max\{0, x\} \quad (5)$$

We need terminologies like epochs, batch size, iterations only when the data is too big which happens all the time in machine learning and we cant pass all the data to the computer at once. So, to overcome this problem we need to divide the data into smaller sizes and give it to our computer one by one and update the weights of the neural networks at the end of every step to fit it to the data given. There are some optimal options for MLP, including batch learning, learning rate and early stopping strategy.

- epoch: one forward pass and one backward pass of all the training examples.
- weights initialization: initialization of the all weights and bias
- batch size: the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- learning rate: gradient descent step size, we set $lr = \frac{lr_0}{epoch}$
- early stopping: If the early shopping rule is applied, we mark minimal validation loss. Once validation loss go upward, we stop training.
- filter size: the length of the square window/filters in the convolution layer.

- pool filter size: the length of the square window/filters in the max pool layer.

And the following figure 8 shows how the learning rate decreases during learning.

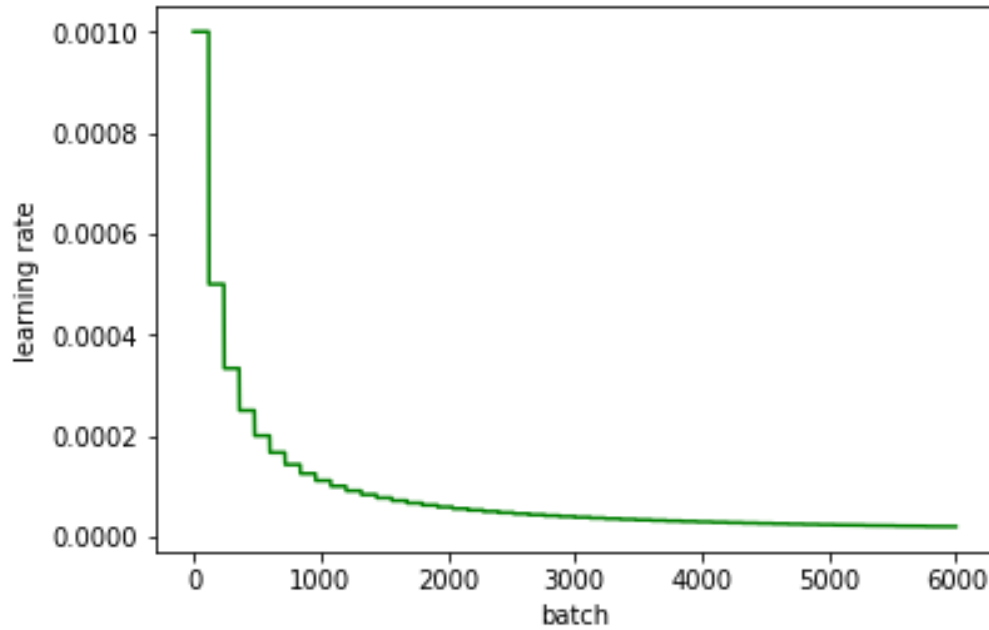


Figure 8: learning rate

```
epochs = 50  # Total number of training epochs
batch_size = 500  # Training batch size
lr_0 = 0.001  # Initial learning rate
do_early_stopping = True  # if apply early stopping learning

# 1st Convolutional Layer
filter_size1 = 5  # Convolution filters are 5*5 pixels.
num_filters1 = 16  # There are 16 of these filters.
stride1 = 1  # The stride of the sliding window

# 2nd Convolutional Layer
filter_size2 = 5  # Convolution filters are 5 x 5 pixels.
num_filters2 = 32  # There are 32 of these filters.
stride2 = 1  # The stride of the sliding window

h1 = 151  # Number of neurons in hidden layer.
```

Learning quality:

For monitoring learning quality, define batch average cross-entropy error (BACRE) and the

gradient norm.

$$BACRE = \frac{1}{B} \sum_{k=1}^B Target_k \cdot \log(OUT_k) \quad (6)$$

$$||G|| = \sqrt{\sum_{i=1}^N \frac{|W(i+1) - W(i)|^2}{N}} \quad (7)$$

After each Epoch(m) , m= 1,2, ... compute the performance indicator percentage of correct classifications, which is defined as "accuracy" in the code, on the whole training set and the whole test set.

$$Accuracy = \frac{N_{correct}}{N_{correct} + N_{incorrect}} \quad (8)$$

```
correct_prediction = tf.equal(tf.argmax(output, 1),
                              tf.argmax(y, 1), name='correct_pred')
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                   tf.float32), name='accuracy')
```

Also, select the best epoch Epoch(m*=best_so_far_epoch) after which the learning should be stopped to avoid overfit.

```
if do_early_stopping == True :
    if acc_test > best_so_far_acc:
        best_so_far_acc = acc_test
        best_so_far_epoch = epoch
```

After the last epoch Epoch(m*) compute and interpret the 10x10 confusion matrix of the best networks we saved in the early stopping strategy(m*).The matrix columns represent the prediction labels and the rows represent the real labels.

	Actual class 1	Actual class 2	...	Actual class 10
Predicted class 1				
Predicted class 2				
...				
Predicted class 10				

Table 2: confusion matrix

For comparing the time of the learning with different options, we use function "datetime" to record the time length of training. start_time is put on the beginning of the session.

```
time_elapsed = datetime.now() - start_time
print( 'Time_elapsed_(hh:mm:ss.ms)_{'}.format( time_elapsed ))
```

5 Performance analysis

Early stopping: test accuracy is highest after 15 epoch. We choose the model that we had then. Plot the the accuracy for training set and test set in Figure 11, that is the percentage of correct classifications. Early stopping learning rule give us the best epoch to stop, which is the epoch have the highest accuracy for the test set, $m^*=15$. At this epoch, the accuracy of test set is 91.92%. Compute the 10x10 confusion matrix. The diagonal elements are close to 1, which means they have good prediction.

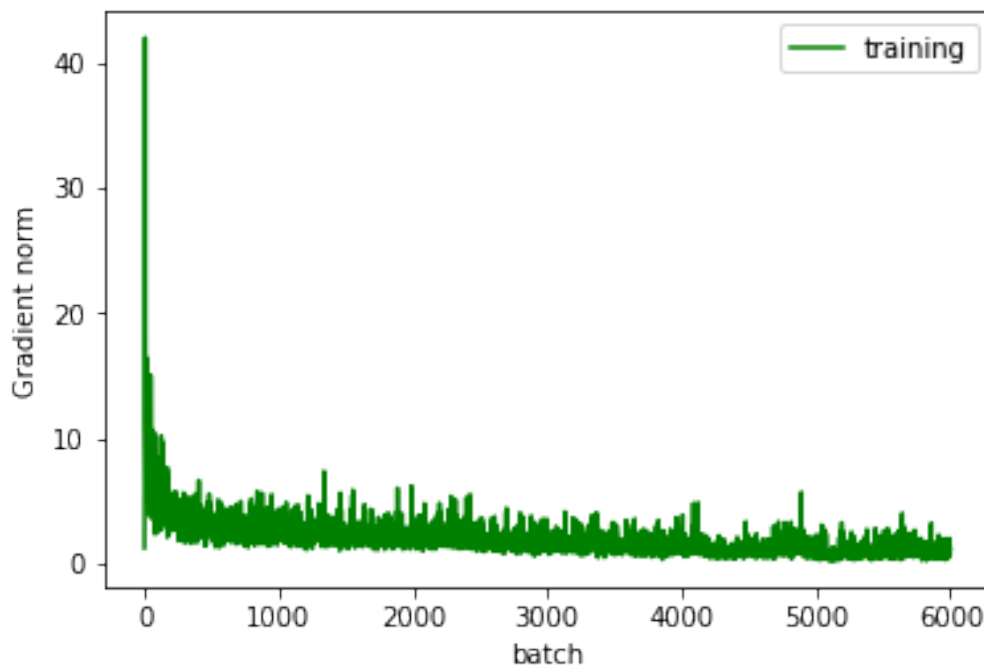


Figure 9: gradient norm

By using gradient decent to minimize average cross entropy error between computed and desired value to get a adapted model, we require the gradient of the loss function to be stable small. As we can see in figure 8, the norm of the gradient decrease quickly and converges to 0 after batch 3000.

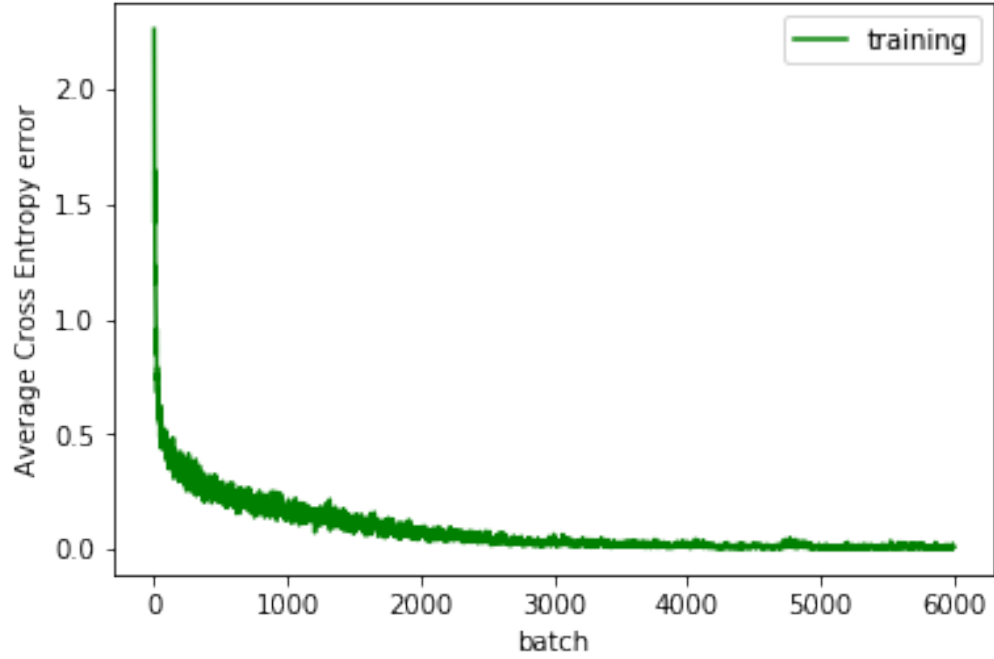


Figure 10: Batch average cross entropy error

We choose a subset of training set to do batch learning, and we get batch average cross entropy per batch in figure 10. After learning, the average cross entropy error goes to be around 0 really fast. And after batch 3000, it almost does not decrease anymore, which means it may happen overfitting after that.

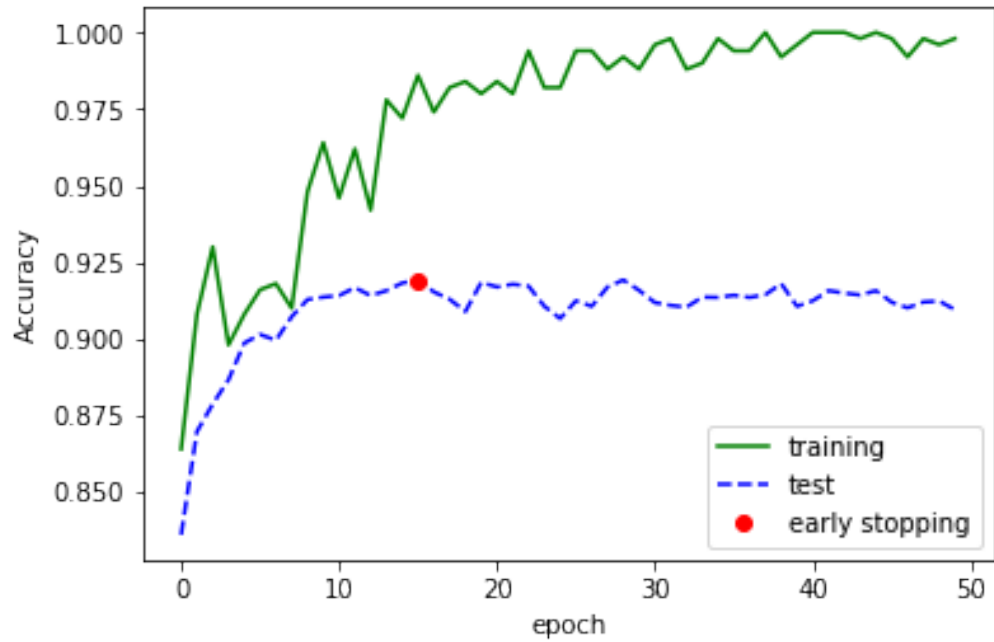


Figure 11: accuracy

Figure 11 is the prediction accuracy for training and test set. According to early stopping learning rule, we choose epoch 15 as its best epoch because test accuracy achieves it highest percentage of correct classification, 91.92%, at this epoch. Then we have $m^*=15$. As you can see, test accuracy is going to be stable and even going down after this epoch, and hence we choose 15 as our best epoch value to avoid overfitting.

	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9	CL10
CL1	5729	1	35	26	3	0	204	0	2	0
CL2	0	5993	0	6	0	0	1	0	0	0
CL3	33	1	5684	11	171	0	100	0	0	0
CL4	27	5	7	5864	65	0	31	0	1	0
CL5	2	0	160	47	5689	0	102	0	0	0
CL6	0	0	0	0	0	5988	0	8	0	4
CL7	245	0	151	61	125	0	5418	0	0	0
CL8	0	0	0	0	0	7	0	5938	0	55
CL9	0	0	0	2	0	0	0	0	5998	0
CL10	0	0	0	0	0	2	0	76	0	5922

Table 3: confusion matrix of training set

	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9	CL10
CL1	892	2	19	12	1	1	62	1	10	0
CL2	0	990	0	8	1	0	0	0	1	0
CL3	19	1	881	8	34	0	55	0	2	0
CL4	13	0	9	938	19	0	17	0	4	0
CL5	1	0	56	33	858	0	49	0	3	0
CL6	0	0	0	0	0	966	0	28	0	6
CL7	114	2	59	27	59	0	735	0	4	0
CL8	0	0	0	0	0	2	0	972	0	26
CL9	2	1	1	2	0	2	0	3	989	0
CL10	0	0	0	0	0	5	1	23	0	971

Table 4: confusion matrix of test set

These two tables show the 10X10 confusion matrix with respect to training and test set. Each column represents the correct prediction percentage for different 10 classes. If the ratio of diagonal value over the sum of its column is close to 1, then the best accuracy attains for this class belonging to this column. The best prediction is obtained for class 2, class 4, class 6, class 8, class 9, class 10. Only few classes are not accurately classified. Therefore, the trained network works pretty well generally.

The most difficult classification is between the Class1 (T-Shirt/Top) and Class7 (Shirt). In the test set, if we have class1(T-shirt/Top), it has $\frac{114}{1000} = 11.4\%$ probability to predict it as class7(shirt). And if we have class7(Shirt), it has $\frac{62}{1000} = 6.2\%$ probability to predict it as

class1(T-shirt/Top). Base on the current convolution classifier, it have about $\frac{892+735}{892+62+735+114} = 90.2\%$ accuracy rate to distinguish them. Let's try to build MLP only base on a pair of two classes. It may help to improve the performance.

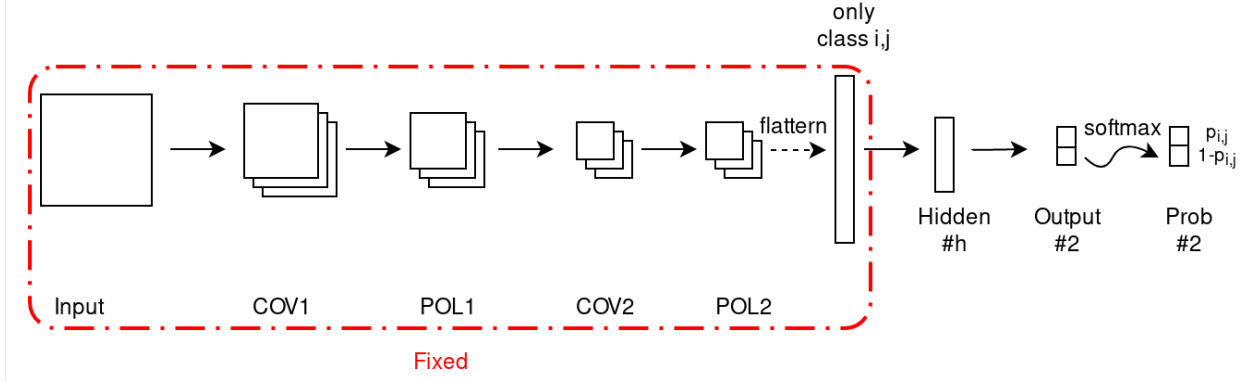


Figure 12: The architecture for calculate score in the test set

After we run $\frac{10 \times 9}{2} = 45$ MLP with 1 hidden layer, it gives the $P_{i,j}$, which means the ture class is i, but the prediction is j. Calculate the Score(i,j), Network i vote for class j:

$$S_{i,j} = -\log(1 - p_{i,j}) \quad (9)$$

	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9	CL10
CL1	*	0	0.02	0.01	0	0	0.07	0	0	0
CL2	*	*	0	0	0	0	0	0	0	0
CL3	*	*	*	0	0.02	0	0.04	0	0	0
CL4	*	*	*	*	0.01	0	0.01	0	0	0
CL5	*	*	*	*	*	0	0.04	0	0	0
CL6	*	*	*	*	*	*	0	0.02	0	0
CL7	*	*	*	*	*	*	*	0	0	0
CL8	*	*	*	*	*	*	*	*	0	0.02
CL9	*	*	*	*	*	*	*	*	*	0

Table 5: Score table for the test set

In table 5, every value of Score(i,j) in each column is relatively low. Look at the case between class1 and class7, $p_{1,7} = 93.4\%$, that is improved (90.2%). But the T-shirt/Top and Shirt are so similar that even I cannot distinguish them well when I check some mis-classified examples. Generally speaking, the convolution neural network works well for the test set.

6 Impact of various learning options

- filter size:F1=3,F2=3

- filter size: $F1=7, F2=7$

Early stopping: test accuracy is highest after 19 epoch. We choose the model that we had then.
The best accuracy: 0.9183

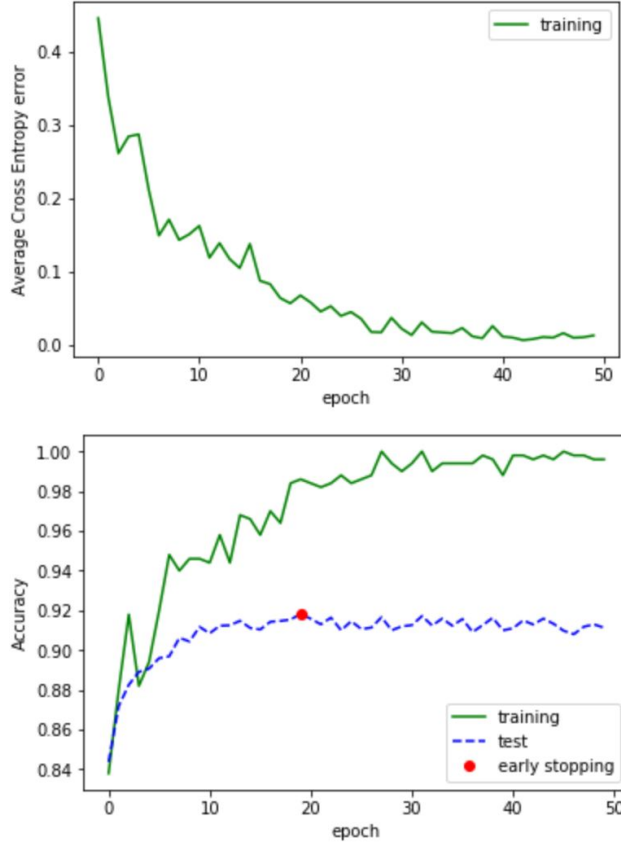


Figure 13: small filter, $F1=F2=3$

We set the the filter size for both the 1st and the 2nd convolution layer to be 3, and plot average cross entropy and accuracy in figure 12 which is the percentage of correct classification. After reevaluate the test prediction accuracy with test set, the test accuracy attains its maximum at 19 epoch. Then we set $m^*=19$. At this epoch, the accuracy of test set is about 91.83%.

Early stopping: test accuracy is highest after 49 epoch. We choose the model that we had then.
The best accuracy: 0.9142

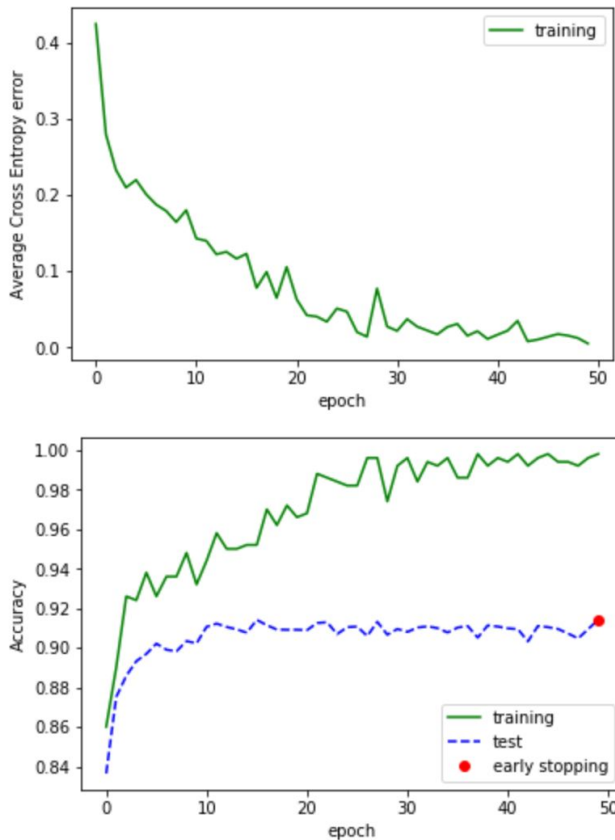


Figure 14: small filter, $F1=F2=7$

We let the filter size for the 1st and the 2nd convolution layer be both 7. And we obtain the graph for the average cross entropy and accuracy in figure 13. Based on early stopping learning rule, we choose epoch 49 as our best epoch since it achieves its maximum value be average test accuracy of 91.42%.

In the lecture, we mention that the experience of the filter size should not be too larger, properly 3 7. In our case, the test accuracy is barely change but the filter size = 5, gives the best accuracy rate in the test set.

- number of filters: $K1=4, K2=8$

Early stopping: test accuracy is highest after 37 epoch. We choose the model that we had then.
The best accuracy: 0.9054

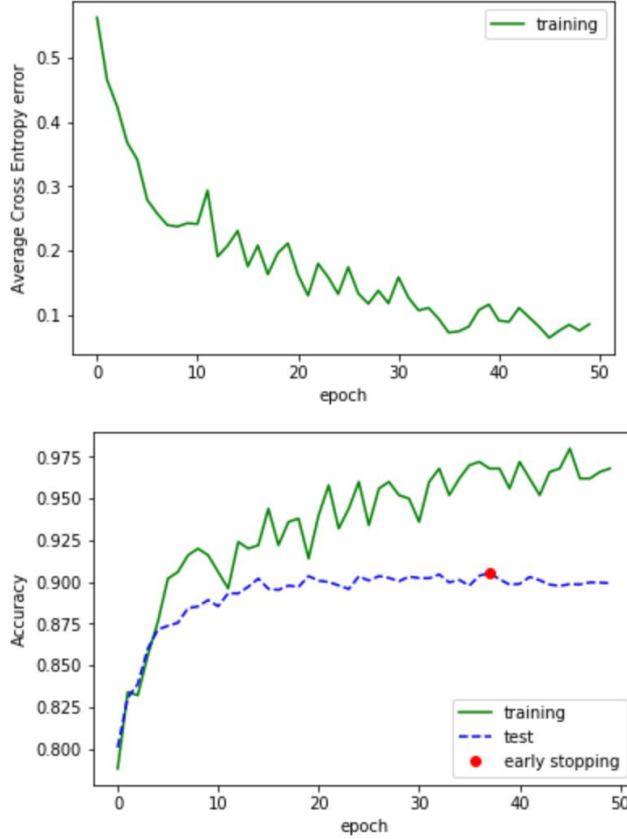


Figure 15: small filter numbers, $K1=4, K2=8$

In this part, we set filter size be 4 for the 1st convolution layer, and filter size be 8 for the second convolution layer. We get the result in figure 14. With the purpose to avoid overfitting, we select epoch 37 as the best epoch value. Then we set $m^*=37$, at which test accuracy has its extreme value 90.54%.

Less number of filters, less unknown weights need to estimate. It save time in the computation, but it also perform worse. From this implement, we could observe that the loss function go downward in the slower rate and (90.54% ; 91.92%) worse performance. Hence, more filter number tends to be more computationally expensive and better accuracy.

- remove the second convolution layer(COL2) and pooling layer(POL2)

Early stopping:test accuracy is highest after 27 epoch. We choose the model that we had then.
The best accuracy: 0.9149

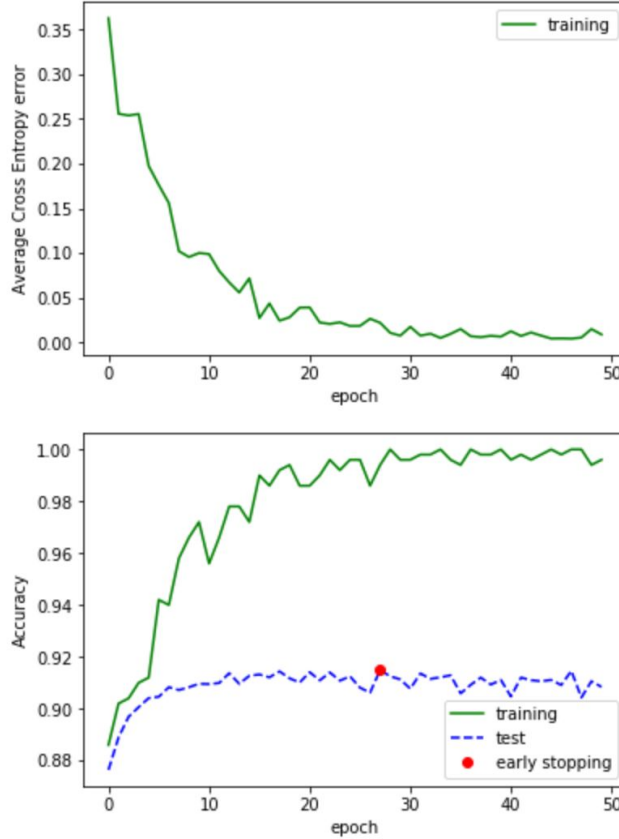


Figure 16: remove COL2 and POL2

By removing the 2nd convolution layer and 2nd pooling layer, we plot average cross entropy and accuracy in figure 14. Applying the early stopping learning rule, we obtain the best epoch value. It is 27 epoch, and the best accuracy is 91.49% at this epoch.

Similarly to the hidden layer number in MLP, more repeated convolution layer and pooling layers should have better performance. On the other size, if we put to many layer, it may bring out too many parameters and expensive computation. We need a balance in decide the number of layers.

In our case, the architecture with repeated two time convolution layers and pooling layer performs better a little (91.92%;91.49%). But if we repeat it three times, there are approximately less than $\frac{47,040,000}{239,687} = 196$ data used to estimate each parameter, that will effect estimation deficiency. Consider the level of improved and the computation efficiency, we do not choose to repeat three times .

- Weight Initialization:

In the Weight Initialization, we do not know what the final value of every weight should be in the trained network, but with proper data normalization it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative. A reasonable-sounding idea then might be to set all the initial weights to zero, which we expect to be the best guess in expectation. This turns out to be a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during back propagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same. Therefore, we still want the weights to be very close to zero, but as we have argued above, not identically zero.

- Batch Size:

Larger batch size means more "accurate" gradients, in the sense of them being closer to the true gradient one would get from whole-batch (processing the entire training set at every step) training. This leads to lower variance in the gradients. But it would be slow due to the number of iteration need per example. On the other side, a small batch size is often more efficient computationally. There would be huge noise in the update since the gradient update direction is only reliant on small part of data point. It currently seems that with deep networks it's preferable to take many small steps than to take fewer larger ones. And it bring the stochastic factor into training, which may lead to lower loss. Some online experience is about (50 1000).

- learning rate

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minimal, it could also mean that well be taking a long time to converge, especially if we get stuck on a plateau region. Conversely, if you specify a learning rate that is too large, the next point will perpetually bounce haphazardly across the bottom of the well like a quantum mechanics experiment gone horribly wrong.