Math 6535 Homework 3

Chunmei Sun, Yingxue Su

April 2019

1 Describe the data set and associated classification task

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The 10 classes are images of airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, trucks. The test set contains exactly 1000 randomly-selected images from each class. The training set contains exactly 5000 images from each class. We want to build a convolution neural network to classify images belonging to these 10 classes.
The original dataset has 5 batches of size 10000 for training. And every image is encoded as a 3072 vector. First we want reshape the vector to $32 \times 32 \times 3$ by the following code:

$$features = batch['data'].reshape((len(batch['data']), 3, 32, 32)).transpose(0, 2, 3, 1)$$

Then we combine these 5 batches together to get our whole training set.

2 CNN without hidden layer

2.1 Functions we use for building CNN

In order to build CNN, we define following several functions.
1. Weight variable function.

$$def \quad weight\_variable(shape):$$

$$initer = tf.truncated\_normal\_initializer(stddev = 0.01)$$

$$return \ \ tf.get\_variable('W', dtype = tf.float32,$$

$$shape = shape, initializer = initer)$$

2. Bias variable function.

$$def \quad bias\_variable(shape):$$

```
            $initer = tf.constant(0., shape = shape, dtype = tf.float32)$
            $return\ \ tf.get\_variable('b', dtype = tf.float32,$
                $shape = shape, initializer = initer)$
```

3. Convolution layer.

```
$def\quad conv\_layer(x, filter\_size, num\_filters, stride, name):$
    $with\quad tf.variable\_scope(name):$
        $num\_in\_channel = x.get\_shape().as\_list()[-1]$
        $shape = [filter\_size, filter\_size, num\_in\_channel, num\_filters]$
        $W = weight\_variable(shape = shape)$
        $tf.summary.histogram('weight', W)$
        $b = bias\_variable(shape = [num_filters])$
        $tf.summary.histogram('bias', b)$
        $layer = tf.nn.conv2d(x, W, strides = [1, stride, stride, 1],$
                $padding = "SAME")$
        $layer+ = b$
    $return\quad tf.nn.relu(layer), W$
```

4. Pooling layer.

```
$def\quad max\_pool(x, ksize, stride, name):$
    $return\quad tf.nn.max\_pool(x, ksize = [1, ksize, ksize, 1],$
            $strides = [1, stride, stride, 1], padding = "SAME", name = name)$
```

5. Flatten layer.

```
$def\quad flatten\_layer(layer):$
    $with\quad tf.variable\_scope('Flatten\_layer'):$
        $layer\_shape = layer.get\_shape()$
        $num\_features = layer\_shape[1:4].num\_elements()$
        $layer\_flat = tf.reshape(layer, [-1, num\_features])$
    $return\quad layer\_flat$
```

6. Fully connected layer

```
$def\quad fc\_layer(x, num\_units, name, use\_relu = True):$
    $with\quad tf.variable\_scope(name):$
    $in\_dim = x.get\_shape()[1]$
    $W = weight\_variable(shape = [in\_dim, num\_units])$
    $tf.summary.histogram('weight', W)$
    $b = bias\_variable(shape = [num\_units])$
    $tf.summary.histogram('bias', b)$
    $layer = tf.matmul(x, W)$
    $layer+ = b$
    $if \quad use\_relu:$
        $layer = tf.nn.relu(layer)$
$return \quad layer$
```

For batch learning, we define following 2 functions for batch learning.
7. Randomization function.

```
$def \quad randomize(x, y):$
    $permutation = np.random.permutation(y.shape[0])$
    $shuffled\_x = x[permutation, :, :, :]$
    $shuffled\_y = y[permutation]$
    $return \quad shuffled\_x, shuffled\_y$
```

8. Get batch function.

```
$def \quad get\_next\_batch(x, y, start, end):$
    $x\_batch = x[start : end]$
    $y\_batch = y[start : end]$
    $return \quad x\_batch, y\_batch$
```

2.2 2 convolution layer CNN

We build a CNN with the architecture as following:

layer name	window size	layer size	number of parameters
image	0	32×32 × 3	0
conv1	3×3	32×32 × 64	(3×3 × 3 + 1) × 64
pool1	2×2	16×16 × 64	0
conv2	3×3	16×16 × 32	(3×3 × 3 + 1) × 32
pool2	2×2	8×8 × 32	0
flat	0	2048×1	0
output	0	10 ×1	20490

And the code for building the CNN with the functions we defined is as following:

$$conv1, w1 = conv\_layer(x, filter\_size1, num\_filters1, stride1, name =' conv1')$$

$$pool1 = max\_pool(conv1, ksize = 2, stride = 2, name =' pool1')$$

$$conv1\_bn = tf.layers.batch\_normalization(pool1)$$

$$conv2, w2 = conv\_layer(conv1\_bn, filter\_size2, num\_filters2, stride2, name =' conv2')$$

$$pool2 = max\_pool(conv2, ksize = 2, stride = 2, name =' pool2')$$

$$conv2\_bn = tf.layers.batch\_normalization(pool2)$$

$$layer\_flat = flatten\_layer(conv2\_bn)$$

$$output\_logits = fc\_layer(layer\_flat, 10,' OUT', use\_relu = False)$$

with $filter\_size1 = 3, num\_filters1 = 64, stride1 = 1, filter\_size2 = 3, num\_filters2 = 32, stride2 = 1$.
For loss function, we use the following code:

$$loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels = y, logits = output\_logits)).$$

For accuracy, we use:

$$correct\_prediction = tf.equal(tf.argmax(output\_logits, 1), tf.argmax(y, 1), name =' correct\_pred')$$

$$accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32), name =' accuracy').$$

In this CNN, the total number of unknown parameters is

$$(3 \times 3 \times 3 + 1) \times 64 + (3 \times 3 \times 3 + 1) \times 32 + 20490 = 23178.$$

We have 50000 cases in total for training. Every case gives us 10 information since we have 10 classes in total. So for every parameter we have

$$50000 \times 10/23178 \approx 21.6$$

information, which is a pretty good situation.
Since we have a pretty big number of unknowns, we used Google Colab with GPU acceleration to implement this homework. We run 80 epochs with batch size 1000. We record accuracy and loss function on the training set and test set every 10 steps and plot these 2 figures in figure 1 and figure 2. We can see that the loss and the accuracy are still improving after 80 epochs. But the overfitting happens around 50 epochs. The highest accuracy on test set was around 70%. The lowest value of loss function was approximate 1 on the test set. When we try to calculate the accuracy and loss function on the whole training set. The memory is not enough. We had to break the training set to 5 smaller sets of size 10000. Then calculate the accuracy and loss function on smaller sets then took the average of 5 losses and 5 accuracies on the smaller sets. For the first convolution layer, we have 64 filters. So we have $3 \times 3 \times 3 \times 64$ kernels in total.

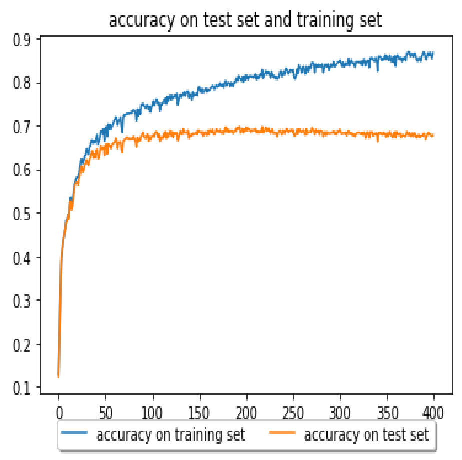


Figure 1: accuracy on training and test set

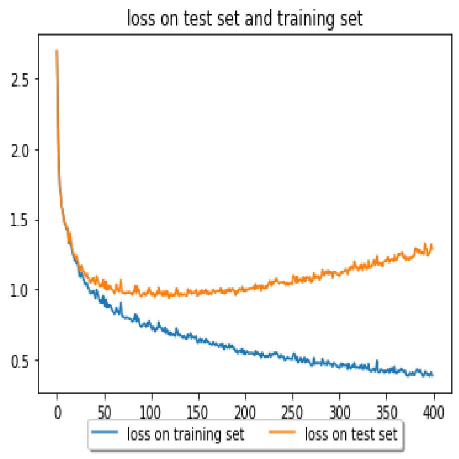


Figure 2: loss on training and test set

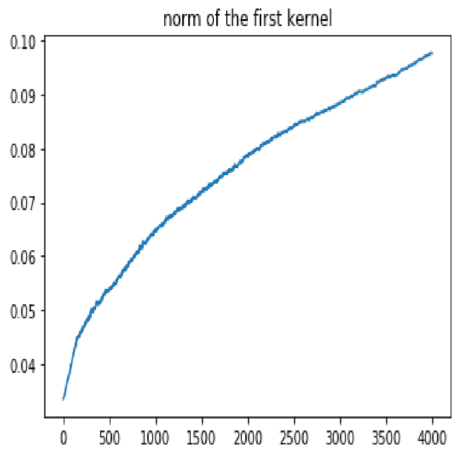


Figure 3: norm of the first kernel

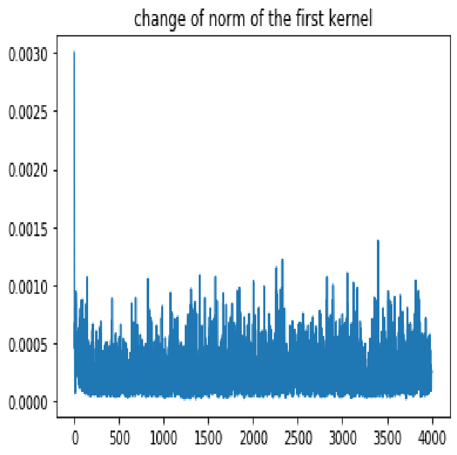


Figure 4: change of norm of first kernel

Let $K1$ be the first kernel. We plot the $||K1(t)||$ and $||K1(t+1) - K1(t)||$ in figure 3,4. We can see the $||K1(t)||$ grows fast in the beginning and the speed of growing reduces and stabilizes after a while.

The confusion matrix on the training set is:

$$CM_{training} = \begin{bmatrix} 0.8412 & 0.004 & 0.0214 & 0.003 & 0.0036 & 0.0022 & 0.0012 & 0.0008 & 0.0254 & 0.005 \\ 0.004 & 0.9174 & 0.0002 & 0.0002 & 0 & 0.0006 & 0.0004 & 0 & 0.0044 & 0.007 \\ 0.0234 & 0.0032 & 0.7414 & 0.0268 & 0.026 & 0.0246 & 0.0226 & 0.008 & 0.0074 & 0.003 \\ 0.0172 & 0.0068 & 0.0424 & 0.7722 & 0.0178 & 0.111 & 0.042 & 0.0152 & 0.0094 & 0.0062 \\ 0.0354 & 0.0052 & 0.0968 & 0.07 & 0.8974 & 0.0562 & 0.0532 & 0.0484 & 0.0013 & 0.0082 \\ 0.006 & 0.0032 & 0.0244 & 0.071 & 0.0062 & 0.754 & 0.0112 & 0.0122 & 0.0026 & 0.0042 \\ 0.0052 & 0.0032 & 0.029 & 0.0228 & 0.012 & 0.0066 & 0.858 & 0.0014 & 0.004 & 0.0006 \\ 0.0152 & 0.0058 & 0.0342 & 0.025 & 0.0288 & 0.041 & 0.0076 & 0.91 & 0.004 & 0.0054 \\ 0.026 & 0.0102 & 0.007 & 0.0042 & 0.0012 & 0.0018 & 0.002 & 0.0012 & 0.9146 & 0.0056 \\ 0.0264 & 0.041 & 0.0032 & 0.0048 & 0.001 & 0.002 & 0.0018 & 0.0028 & 0.0152 & 0.9548 \end{bmatrix}$$

The confusion matrix on the test set is:

$$CM_{test} = \begin{bmatrix} 0.682 & 0.027 & 0.055 & 0.022 & 0.014 & 0.009 & 0.005 & 0.010 & 0.068 & 0.027 \\ 0.023 & 0.739 & 0.011 & 0.003 & 0.002 & 0.001 & 0.004 & 0 & 0.031 & 0.062 \\ 0.059 & 0.009 & 0.533 & 0.059 & 0.063 & 0.055 & 0.047 & 0.025 & 0.015 & 0.007 \\ 0.025 & 0.012 & 0.078 & 0.490 & 0.055 & 0.197 & 0.071 & 0.047 & 0.025 & 0.025 \\ 0.045 & 0.013 & 0.128 & 0.110 & 0.705 & 0.062 & 0.062 & 0.112 & 0.013 & 0.014 \\ 0.007 & 0.012 & 0.064 & 0.172 & 0.033 & 0.557 & 0.043 & 0.039 & 0.015 & 0.008 \\ 0.013 & 0.007 & 0.058 & 0.059 & 0.044 & 0.021 & 0.741 & 0.011 & 0.011 & 0.006 \\ 0.025 & 0.011 & 0.051 & 0.054 & 0.071 & 0.077 & 0.018 & 0.738 & 0.005 & 0.023 \\ 0.067 & 0.044 & 0.016 & 0.013 & 0.011 & 0.007 & 0.004 & 0.006 & 0.782 & 0.033 \\ 0.054 & 0.126 & 0.006 & 0.018 & 0.002 & 0.014 & 0.005 & 0.012 & 0.035 & 0.795 \end{bmatrix}$$

We can see that the CNN has difficulty classifying class 3, class 4 and class 6. The accuracy interval of CM_{train} is:

$$ACCU_{train}[:, 1:5] =$$

$$\begin{bmatrix} [0.8360, 0.8464] & [0.0031, 0.0049] & [0.0194, 0.0234] & [0.0022, 0.0038] & [0.0028, 0.0044] \\ [0.0031, 0.0049] & [0.9135, 0.9213] & [0.0000, 0.0004] & [0.0000, 0.0004] & [0, 0] \\ [0.0213, 0.0255] & [0.0024, 0.0040] & [0.7352, 0.7476] & [0.0245, 0.0291] & [0.0237, 0.0283] \\ [0.0154, 0.0190] & [0.0056, 0.0080] & [0.0396, 0.0452] & [0.7663, 0.7781] & [0.0159, 0.0197] \\ [0.0328, 0.0380] & [0.0042, 0.0062] & [0.0926, 0.1010] & [0.0664, 0.0736] & [0.8931, 0.9017] \\ [0.0049, 0.0071] & [0.0024, 0.0040] & [0.0222, 0.0266] & [0.0674, 0.0746] & [0.0106, 0.0138] \\ [0.0042, 0.0062] & [0.0024, 0.0040] & [0.0266, 0.0314] & [0.0207, 0.0249] & [0.0105, 0.0135] \\ [0.0135, 0.0169] & [0.0047, 0.0069] & [0.0316, 0.0368] & [0.0228, 0.0272] & [0.0264, 0.0312] \\ [0.0237, 0.0283] & [0.0088, 0.0116] & [0.0058, 0.0082] & [0.0033, 0.0051] & [0.0007, 0.0017] \\ [0.0241, 0.0287] & [0.0382, 0.0438] & [0.0024, 0.0040] & [0.0038, 0.0058] & [0.0006, 0.0014] \end{bmatrix}$$

$$ACCU_{train}[:, 6:10] =$$

$$\begin{bmatrix}
[0.0015, 0.0029] & [0.0007, 0.0017] & [0.0004, 0.0012] & [0.0232, 0.0276] & [0.0040, 0.0060] \\
[0.0003, 0.0009] & [0.0001, 0.0007] & [0, 0] & [0.0035, 0.0053] & [0.0058, 0.0082] \\
[0.0224, 0.0268] & [0.0205, 0.0247] & [0.0067, 0.0093] & [0.0062, 0.0086] & [0.0022, 0.0038] \\
[0.1066, 0.1154] & [0.0392, 0.0448] & [0.0135, 0.0169] & [0.0080, 0.0108] & [0.0051, 0.0073] \\
[0.0529, 0.0595] & [0.0500, 0.0564] & [0.0454, 0.0514] & [0.0114, 0.0146] & [0.0069, 0.0095] \\
[0.7479, 0.7601] & [0.0097, 0.0127] & [0.0106, 0.0138] & [0.0019, 0.0033] & [0.0033, 0.0051] \\
[0.0055, 0.0077] & [0.8531, 0.8629] & [0.0009, 0.0019] & [0.0031, 0.0049] & [0.0003, 0.0009] \\
[0.0382, 0.0438] & [0.0064, 0.0088] & [0.9060, 0.9140] & [0.0031, 0.0049] & [0.0044, 0.0064] \\
[0.0012, 0.0024] & [0.0014, 0.0026] & [0.0007, 0.0017] & [0.9106, 0.9186] & [0.0045, 0.0067] \\
[0.0014, 0, 0026] & [0.0012, 0.0024] & [0.0021, 0.0035] & [0.0135, 0.0169] & [0.9519, 0.9577]
\end{bmatrix}$$

And the accuracy interval of CM_{test} is:

$$ACCU_{test}[:, 1:5] =$$

$$\begin{bmatrix}
[0.6673, 0.6967] & [0.0219, 0.0321] & [0.0478, 0.0622] & [0.0174, 0.0266] & [0.0103, 0.0177] \\
[0.0183, 0.0277] & [0.7251, 0.7529] & [0.0077, 0.0143] & [0.0013, 0.0047] & [0.0006, 0.0034] \\
[0.0515, 0.0665] & [0.0060, 0.0120] & [0.5172, 0.5488] & [0.0515, 0.0665] & [0.0553, 0.0707] \\
[0.0201, 0.0299] & [0.0086, 0.0154] & [0.0695, 0.0865] & [0.4742, 0.5058] & [0.0478, 0.0622] \\
[0.0384, 0.0516] & [0.0094, 0.0166] & [0.1174, 0.1386] & [0.1001, 0.1199] & [0.6906, 0.7194] \\
[0.0044, 0.0096] & [0.0086, 0.0154] & [0.0563, 0.0717] & [0.1601, 0.1839] & [0.0274, 0.0386] \\
[0.0094, 0.0166] & [0.0044, 0.0096] & [0.0506, 0.0654] & [0.0515, 0.0665] & [0.0375, 0.0505] \\
[0.0201, 0.0299] & [0.0077, 0.0143] & [0.0440, 0.0580] & [0.0469, 0.0611] & [0.0629, 0.0791] \\
[0.0591, 0.0749] & [0.0375, 0.0505] & [0.0120, 0.0200] & [0.0094, 0.0166] & [0.0077, 0.0143] \\
[0.0469, 0.0611] & [0.1155, 0.1365] & [0.0036, 0.0084] & [0.0138, 0.0222] & [0.0006, 0.0034]
\end{bmatrix}$$

$$ACCU_{test}[:, 6:10] =$$

$$\begin{bmatrix}
[0.0060, 0.0120] & [0.0028, 0.0072] & [0.0069, 0.0131] & [0.0600, 0.0760] & [0.0219, 0.0321] \\
[0.0000, 0.0020] & [0.0002, 0.0006] & [0, 0] & [0.0255, 0.0365] & [0.0544, 0.0696] \\
[0.0478, 0.0622] & [0.0403, 0.0537] & [0.0201, 0.0299] & [0.0112, 0.0188] & [0.0044, 0.0096] \\
[0.1844, 0.2096] & [0.0629, 0.0791] & [0.0403, 0.0537] & [0.0201, 0.0299] & [0.0201, 0.0299] \\
[0.0544, 0.0696] & [0.0544, 0.0696] & [0.1020, 0.1220] & [0.0094, 0.0166] & [0.0103, 0.0177] \\
[0.5413, 0.5727] & [0.0366, 0.0494] & [0.0329, 0.0451] & [0.0112, 0.0188] & [0.0052, 0.0108] \\
[0.0165, 0.0255] & [0.7271, 0.7549] & [0.0077, 0.0143] & [0.0077, 0.0143] & [0.0036, 0.0084] \\
[0.0686, 0.0854] & [0.0138, 0.0222] & [0.7241, 0.7519] & [0.0028, 0.0072] & [0.0183, 0.0277] \\
[0.0044, 0.0096] & [0.0020, 0.0060] & [0.0036, 0.0084] & [0.7689, 0.7951] & [0.0274, 0.0386] \\
[0.0103, 0, 0177] & [0.0028, 0.0072] & [0.0086, 0.0154] & [0.0292, 0.0408] & [0.7408, 0.8078]
\end{bmatrix}$$

After training, if we take the values on the flatten layer as the new input for every case. Then perform PCA analysis on the whole training new input. Plot the eigenvalues of the correlation matrix in figure 5. And plot the projection of the new input on 2 component in figure 6. First 767 eigenvalues preserve 95% of the sum of all eigenvalues. If we use the new input which has size 2046 and 767 neurons hidden layer to build a MLP to classify 10 classes images, we would have

$$2046 \times 767 + 767 + 767 \times 10 + 10 = 1577729$$

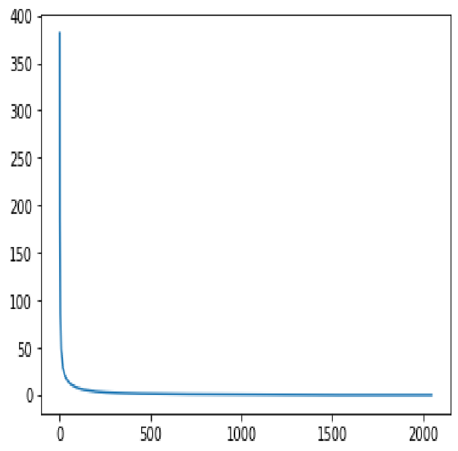


Figure 5: eigenvalues of correlation matrix

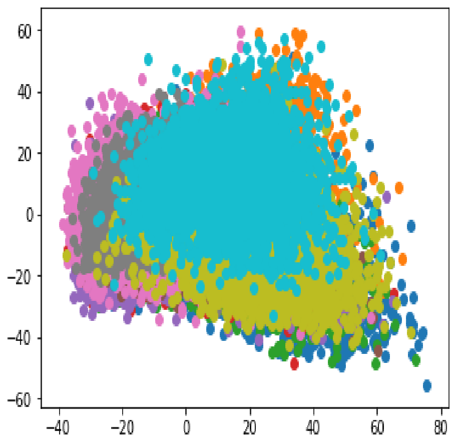


Figure 6: projection on 2 components

parameters. Then for every parameter, we have

$$50000 \times 10/1577729 = 0.3169$$

information which is a very bad situation actually. So we want to add another convolution layer and pooling layer to fully reduce the size of flatten layer and the parameters of MLP, which is stated in section 2.3.

2.3 3 convolution layer CNN

We build a CNN with the architecture as following:

layer name	window size	layer size	number of parameters
image	0	$32\times32\times3$	0
conv1	3×3	$32\times32\times64$	$(3\times3\times3+1)\times64$
pool1	2×2	$16\times16\times64$	0
conv2	3×3	$16\times16\times32$	$(3\times3\times3+1)\times32$
pool2	2×2	$8\times8\times32$	0
conv3	3×3	$8\times8\times16$	$(3\times3\times3+1)\times16$
pool3	2×2	$4\times4\times16$	0
flat	0	256×1	0
output	0	10×1	2570

In this case, we have in total

$$(3\times3\times3+1)\times(64+32+16)+2570=5706$$

unknowns. So we have

$$50000\times10/5706=87.6$$

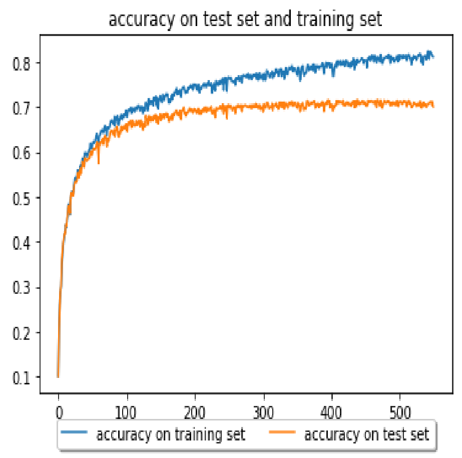


Figure 7: accuracy on training and test set

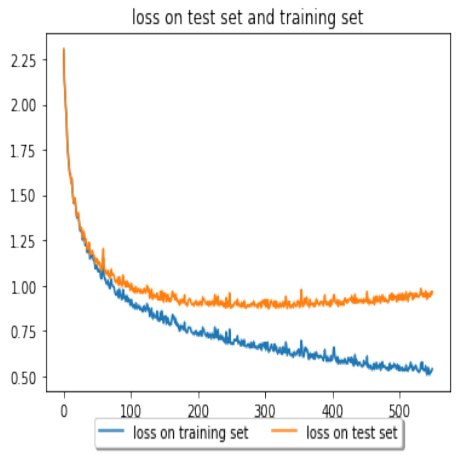


Figure 8: loss on training and test set

information for every parameter, which is very nice.

We can see that the loss and the accuracy are still improving after 110 epochs. But the accuracy and loss function stabilizes on the test set. The highest accuracy on test set was around 71%. The lowest value of loss function was approximate 0.87 on the test set. We plot the accuracy and loss function in figure 7 and 8. For the first convolution layer, we have 64 filters. So we have $3 \times 3 \times 3 \times 64$ kernels in total. Let $K1_r$,$K1_g$,$K1_b$ be the first kernel for red, green and blue. We plot the $||K1_r(t)||$,$||K1_g(t)||$,$||K1_b(t)||$ and $||K1_r(t+1) - K1_r(t)||$,$||K1_g(t+1) - K1_g(t)||$,$||K1_b(t+1) - K1_b(t)||$ in figure 9,10. We can see the all the norms of the filter grows fast in the beginning and the speed of growing reduces and stabilizes after a while. The confusion matrix of this CNN on training set is:

$$CM3_{training} =$$

$$\begin{bmatrix}
0.8162 & 0.0074 & 0.2780 & 0.0062 & 0.0068 & 0.0018 & 0.0012 & 0.0040 & 0.0336 & 0.0246 \\
0.0130 & 0.9442 & 0.0003 & 0.0004 & 0.0006 & 0.0014 & 0.0010 & 0.0016 & 0.0110 & 0.0654 \\
0.0554 & 0.0048 & 0.7758 & 0.0494 & 0.0051 & 0.0486 & 0.0232 & 0.0326 & 0.0108 & 0.0066 \\
0.0224 & 0.0042 & 0.0041 & 0.7032 & 0.0476 & 0.2 & 0.0286 & 0.0498 & 0.0198 & 0.0196 \\
0.0232 & 0.0016 & 0.0542 & 0.0508 & 0.7932 & 0.0448 & 0.0172 & 0.0560 & 0.0074 & 0.0052 \\
0.0022 & 0.0002 & 0.0182 & 0.0672 & 0.0112 & 0.6258 & 0.0040 & 0.0194 & 0.0008 & 0.0026 \\
0.0116 & 0.0074 & 0.0604 & 0.0948 & 0.0594 & 0.0458 & 0.9204 & 0.0116 & 0.0072 & 0.0104 \\
0.0064 & 0.0022 & 0.0118 & 0.0132 & 0.0254 & 0.0274 & 0.0008 & 0.8218 & 0.0016 & 0.0100 \\
0.0406 & 0.0076 & 0.0056 & 0.0074 & 0.0028 & 0.0020 & 0.0028 & 0.0012 & 0.9002 & 0.0156 \\
0.0090 & 0.0204 & 0.0022 & 0.0038 & 0.0020 & 0.0024 & 0.0008 & 0.0020 & 0.0076 & 0.8400
\end{bmatrix}$$

The confusion matrix on the test set is:

$$CM3_{test} =$$

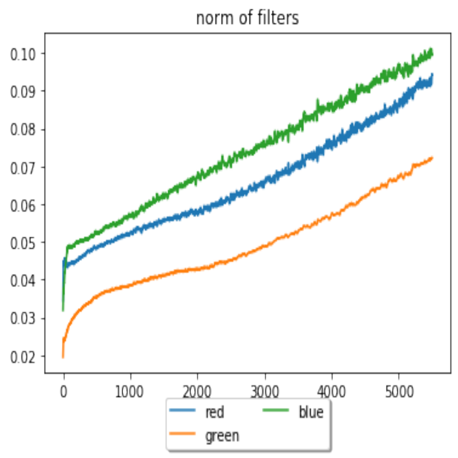


Figure 9: norm of the first kernel

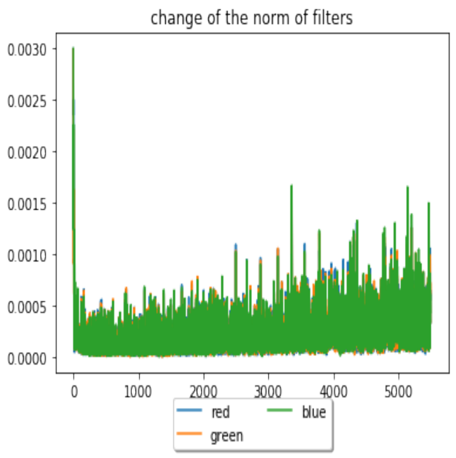


Figure 10: change of norm of first kernel

$$\begin{bmatrix}
0.728 & 0.022 & 0.050 & 0.016 & 0.018 & 0.008 & 0.003 & 0.012 & 0.077 & 0.037 \\
0.027 & 0.851 & 0.006 & 0.019 & 0.002 & 0.002 & 0.006 & 0.011 & 0.044 & 0.099 \\
0.070 & 0.003 & 0.617 & 0.092 & 0.071 & 0.055 & 0.041 & 0.047 & 0.026 & 0.010 \\
0.033 & 0.010 & 0.067 & 0.574 & 0.079 & 0.242 & 0.045 & 0.064 & 0.023 & 0.030 \\
0.025 & 0.006 & 0.081 & 0.057 & 0.655 & 0.055 & 0.035 & 0.078 & 0.014 & 0.009 \\
0.002 & 0.005 & 0.045 & 0.093 & 0.019 & 0.529 & 0.008 & 0.047 & 0.002 & 0.005 \\
0.015 & 0.015 & 0.079 & 0.102 & 0.106 & 0.047 & 0.850 & 0.015 & 0.015 & 0.007 \\
0.013 & 0.004 & 0.033 & 0.024 & 0.042 & 0.057 & 0.003 & 0.716 & 0.002 & 0.015 \\
0.066 & 0.027 & 0.012 & 0.011 & 0.007 & 0.004 & 0.007 & 0.002 & 0.776 & 0.033 \\
0.021 & 0.057 & 0.010 & 0.012 & 0.001 & 0.001 & 0.002 & 0.008 & 0.021 & 0.755
\end{bmatrix}$$

Compare $CM3_{test}$ with the accuracy interval of CM_{test}, we can see that $CM3_{test}(i,i) > upper\ bound\ of\ \ CM_{test}(i,i)$ for $i = 1, ..., 10$. So these 3 convolution layers CNN performs better than 2 layer CNN.

After training, if we take the values on the flatten layer as the new input for every case. Then perform PCA analysis on the whole training new input. Plot the eigenvalues of the correlation matrix in figure 11. First 157 eigenvalues preserve 95% of the sum of all eigenvalues. And plot the projection of the new input on 3 component in figure 12. Each color represent the image from one class. It looks like these 10 classes images have 10 clusters, which means it won't be impossible to separate them.

If we build a MLP with 157 hidden neurons, the parameters we have for this MLP is

$$256 \times 157 + 157 + 157 \times 10 + 10 = 41929.$$

Then for every parameter we have

$$50000 \times 10/41929 = 11.9$$

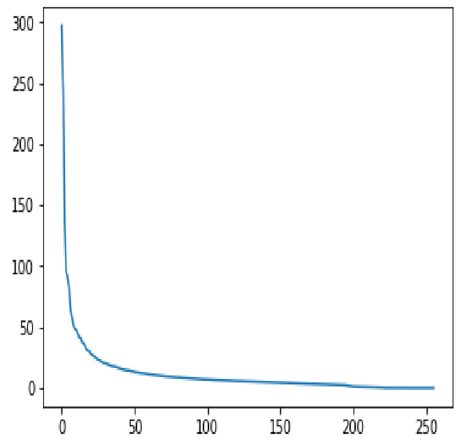


Figure 11: eigenvalues of correlation matrix

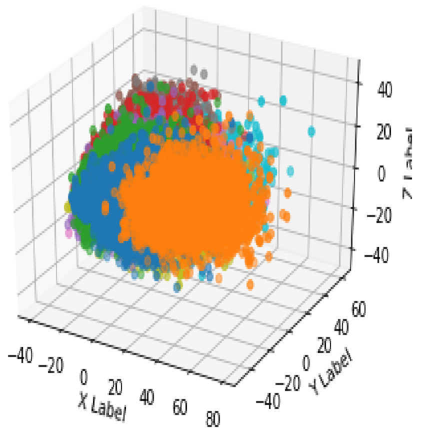


Figure 12: projection on 3 components

information, which is a relatively small number. We plot the accuracy and the loss function of this MLP in figure 13 and 14. We can see that the accuracy on the training set and test set oscillate a lot after they stabilize. The accuracy on test set is below 0.6 which is lower than the 3 convolution layer CNN we built.
The confusion matrix on the training set $CM4_{train}$ is in figure15. And the lower bound and higher bound of the confusion matrix is in figure 16 and 17.
The confusion matrix on the test set $CM4_{test}$ is in figure 18. And the lower bound and higher bound of the confusion matrix are in figure 19,20.

We can see that $CM3_{train}(i,i)$ is larger than the upper bound of $CM4_{train}(i,i)$ and $CM3_{test}(i,i)$ is larger than the upper bound of $CM4_{test}(i,i)$ for $i = 1,..,10$. So the original 3 convolution layer CNN performs better than the MLP.

2.4 3 convolution layer CNN with hidden layer

The CNN with hidden layer has the following architecture:

layer name	window size	layer size	number of parameters
image	0	$32\times32\times3$	0
conv1	3×3	$32\times32\times64$	$(3\times3\times3+1)\times64$
pool1	2×2	$16\times16\times64$	0
conv2	3×3	$16\times16\times32$	$(3\times3\times3+1)\times32$
pool2	2×2	$8\times8\times32$	0
conv3	3×3	$8\times8\times16$	$(3\times3\times3+1)\times16$
pool3	2×2	$4\times4\times16$	0
flat	0	256×1	0
hidden layer	0	157×1	$256\times157+157$
output	0	10×1	$157\times10+10$

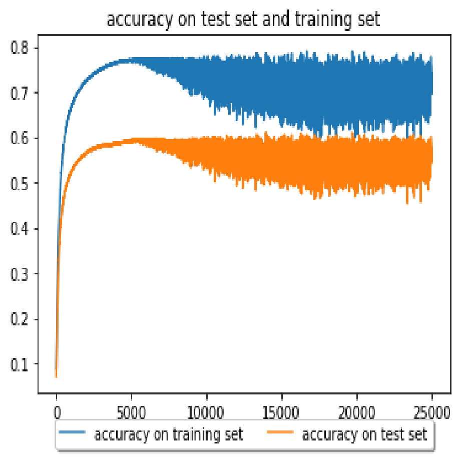


Figure 13: accuracy on training and test set

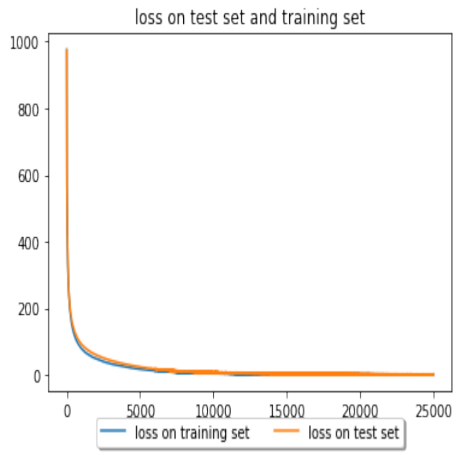


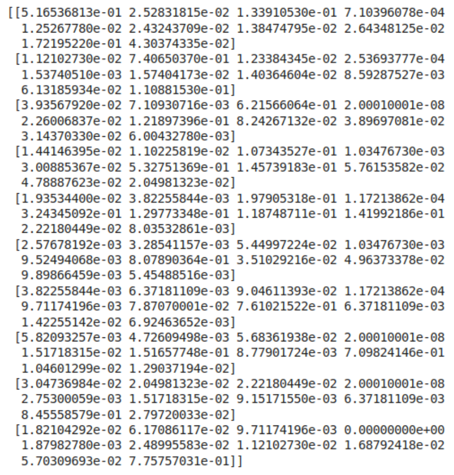
Figure 14: loss on training and test set

```
[[5.236e-01 2.760e-02 1.388e-01 1.200e-03 1.420e-02 2.660e-02 1.560e-02
  2.880e-02 1.776e-01 4.600e-02]
 [1.280e-02 7.468e-01 1.400e-02 6.000e-04 2.200e-03 1.760e-02 1.580e-02
  1.000e-02 6.480e-02 1.154e-01]
 [4.220e-02 8.400e-03 6.284e-01 2.000e-04 2.480e-02 1.266e-01 8.640e-02
  4.180e-02 3.400e-02 7.200e-03]
 [1.620e-02 1.260e-02 1.118e-01 1.600e-03 3.260e-02 5.398e-01 1.508e-01
  6.100e-02 5.100e-02 2.260e-02]
 [2.140e-02 4.800e-03 2.036e-01 4.000e-04 3.310e-01 1.346e-01 1.234e-01
  1.470e-01 2.440e-02 9.400e-03]
 [3.400e-03 4.200e-03 5.780e-02 1.600e-03 1.100e-02 8.134e-01 3.780e-02
  5.280e-02 1.140e-02 6.600e-03]
 [4.800e-03 7.600e-03 9.460e-02 4.000e-04 1.120e-02 8.260e-02 7.670e-01
  7.600e-03 1.600e-02 8.200e-03]
 [7.000e-03 5.800e-03 6.020e-02 2.000e-04 1.700e-02 1.568e-01 1.020e-02
  7.162e-01 1.200e-02 1.460e-02]
 [3.300e-02 2.260e-02 2.440e-02 2.000e-04 3.600e-03 1.700e-02 1.060e-02
  7.600e-03 8.506e-01 3.040e-02]
 [2.020e-02 6.520e-02 1.120e-02 0.000e+00 2.600e-03 2.720e-02 1.280e-02
  1.880e-02 6.040e-02 7.816e-01]]
```

Figure 15: Confusion matrix on training set

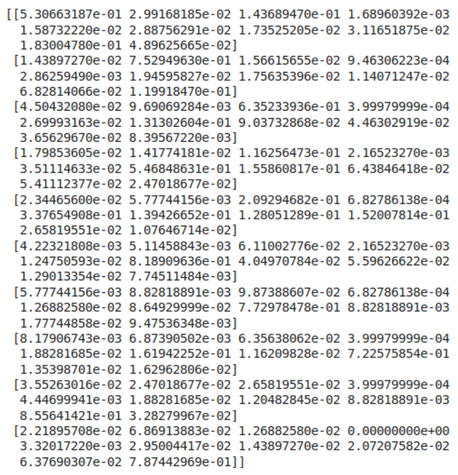
In this case, we have in total

$$(3 \times 3 \times 3 + 1) \times (64 + 32 + 16) + 256 \times 157 + 157 + 157 \times 10 + 10 = 45065$$



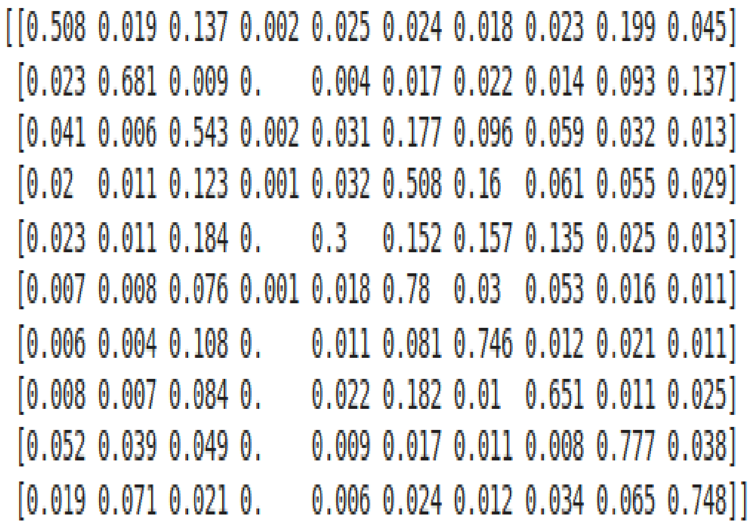
```
[[5.16536813e-01 2.52831815e-02 1.33910530e-01 7.10396078e-04
  1.25267780e-02 2.43243709e-02 1.38474795e-02 2.64348125e-02
  1.72195220e-01 4.30374335e-02]
 [1.12102730e-02 7.40650370e-01 1.23384345e-02 2.53693777e-04
  1.53740510e-03 1.57404173e-02 1.40364604e-02 8.59287527e-03
  6.13185934e-02 1.10881530e-01]
 [3.93567920e-02 7.10930716e-03 6.21566064e-01 2.00010001e-08
  2.26006837e-02 1.21897396e-01 8.24267132e-02 3.89697081e-02
  3.14370330e-02 6.00432780e-03]
 [1.44146395e-02 1.10225819e-02 1.07343527e-01 1.03476730e-03
  3.00885367e-02 5.32751369e-01 1.45739183e-01 5.76153582e-02
  4.78887623e-02 2.04981323e-02]
 [1.93534400e-02 3.82255844e-03 1.97905318e-01 1.17213862e-04
  3.24345092e-01 1.29773348e-01 1.18748711e-01 1.41992186e-01
  2.22180449e-02 8.03532861e-03]
 [2.57678192e-03 3.28541157e-03 5.44997224e-02 1.03476730e-03
  9.52494068e-03 8.07890364e-01 3.51029216e-02 4.96373378e-02
  9.89866459e-03 5.45488516e-03]
 [3.82255844e-03 6.37181109e-03 9.04611393e-02 1.17213862e-04
  9.71174196e-03 7.87070001e-02 7.61021522e-01 6.37181109e-03
  1.42255142e-02 6.92463652e-03]
 [5.82093257e-03 4.72609498e-03 5.68361938e-02 2.00010001e-08
  1.51718315e-02 1.51657748e-01 8.77901724e-03 7.09824146e-01
  1.04601299e-02 1.29037194e-02]
 [3.04736984e-02 2.04981323e-02 2.22180449e-02 2.00010001e-08
  2.75300059e-03 1.51718315e-02 9.15171550e-03 6.37181109e-03
  8.45558579e-01 2.79720033e-02]
 [1.82104292e-02 6.17086117e-02 9.71174196e-03 0.00000000e+00
  1.87982780e-03 2.48995583e-02 1.12102730e-02 1.68792418e-02
  5.70309693e-02 7.75757031e-01]]
```

Figure 16: lower bound of frequencies



```
[[5.30663187e-01 2.99168185e-02 1.43689470e-01 1.68960392e-03
  1.58732220e-02 2.88756291e-02 1.73525205e-02 3.11651875e-02
  1.83004780e-01 4.89625665e-02]
 [1.43897270e-02 7.52949630e-01 1.56615655e-02 9.46306223e-04
  2.86259490e-03 1.94595827e-02 1.75635396e-02 1.14071247e-02
  6.82814066e-02 1.19918470e-01]
 [4.50432080e-02 9.69069284e-03 6.35233936e-01 3.99979999e-04
  2.69993163e-02 1.31302604e-01 9.03732868e-02 4.46302919e-02
  3.65629670e-02 8.39567220e-03]
 [1.79853605e-02 1.41774181e-02 1.16256473e-01 2.16523270e-03
  3.51114633e-02 5.46848631e-01 1.55860817e-01 6.43846418e-02
  5.41112377e-02 2.47018677e-02]
 [2.34465600e-02 5.77744156e-03 2.09294682e-01 6.82786138e-04
  3.37654908e-01 1.39426652e-01 1.28051289e-01 1.52007814e-01
  2.65819551e-02 1.07646714e-02]
 [4.22321808e-03 5.11458843e-03 6.11002776e-02 2.16523270e-03
  1.24750593e-02 8.18909636e-01 4.04970784e-02 5.59626622e-02
  1.29013354e-02 7.74511484e-03]
 [5.77744156e-03 8.82818891e-03 9.87388607e-02 6.82786138e-04
  1.26882580e-02 8.64929999e-02 7.72978478e-01 8.82818891e-03
  1.77744858e-02 9.47536348e-03]
 [8.17906743e-03 6.87390502e-03 6.35638062e-02 3.99979999e-04
  1.88281685e-02 1.61942252e-01 1.16209828e-02 7.22575854e-01
  1.35398701e-02 1.62962806e-02]
 [3.55263016e-02 2.47018677e-02 2.65819551e-02 3.99979999e-04
  4.44699941e-03 1.88281685e-02 1.20482845e-02 8.82818891e-03
  8.55641421e-01 3.28279967e-02]
 [2.21895708e-02 6.86913883e-02 1.26882580e-02 0.00000000e+00
  3.32017220e-03 2.95004417e-02 1.43897270e-02 2.07207582e-02
  6.37690307e-02 7.87442969e-01]]
```

Figure 17: higher bound of frequencies



```
[[0.508 0.019 0.137 0.002 0.025 0.024 0.018 0.023 0.199 0.045]
 [0.023 0.681 0.009 0.    0.004 0.017 0.022 0.014 0.093 0.137]
 [0.041 0.006 0.543 0.002 0.031 0.177 0.096 0.059 0.032 0.013]
 [0.02  0.011 0.123 0.001 0.032 0.508 0.16  0.061 0.055 0.029]
 [0.023 0.011 0.184 0.    0.3   0.152 0.157 0.135 0.025 0.013]
 [0.007 0.008 0.076 0.001 0.018 0.78  0.03  0.053 0.016 0.011]
 [0.006 0.004 0.108 0.    0.011 0.081 0.746 0.012 0.021 0.011]
 [0.008 0.007 0.084 0.    0.022 0.182 0.01  0.651 0.011 0.025]
 [0.052 0.039 0.049 0.    0.009 0.017 0.011 0.008 0.777 0.038]
 [0.019 0.071 0.021 0.    0.006 0.024 0.012 0.034 0.065 0.748]]
```

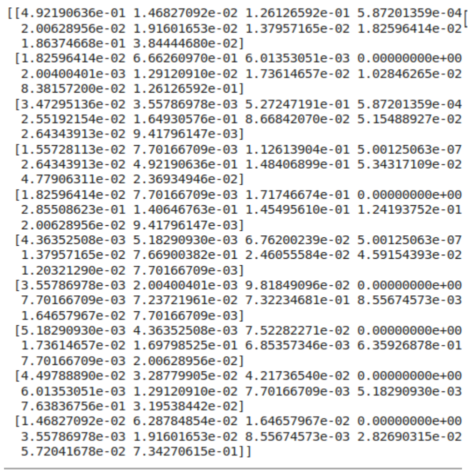
Figure 18: Confusion matrix on test set

unknowns. So we have

$$50000 \times 10/45065 = 11.1$$

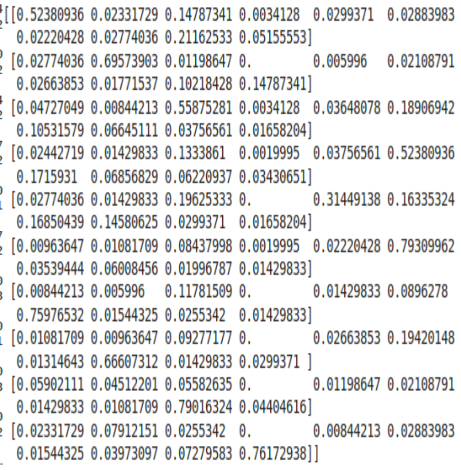
information for every parameter, which is not too bad.

After 110 epochs training, we plot the accuracy and loss function on the training set and test set in figure 19,20. The accuracy on the test set stabilizes around 0.71. For the first convolution layer, we have 64 filters. So we have $3 \times 3 \times 3 \times 64$ kernels in total. Let $K1_r$,$K1_g$,$K1_b$ be the first kernel for red,



```
[[4.92190636e-01 1.46827092e-02 1.26126592e-01 5.87201359e-04
  2.00628956e-02 1.91601653e-02 1.37957165e-02 1.82596414e-02
  1.86374668e-01 3.84444680e-02]
 [1.82596414e-02 6.66260970e-01 6.01353051e-03 0.00000000e+00
  2.00400401e-03 1.29120910e-02 1.73614657e-02 1.02846265e-02
  8.38157200e-02 1.26126592e-01]
 [3.47295136e-02 3.55786978e-03 5.27247191e-01 5.87201359e-04
  2.55192154e-02 1.64930576e-01 8.66842070e-02 5.15488927e-02
  2.64343913e-02 9.41796147e-03]
 [1.55728113e-02 7.70166709e-03 1.12613904e-01 5.00125063e-07
  2.64343913e-02 4.92190636e-01 1.48406899e-01 5.34317109e-02
  4.77906311e-02 2.36934946e-02]
 [1.82596414e-02 7.70166709e-03 1.71746674e-01 0.00000000e+00
  2.85508623e-01 1.40646763e-01 1.45495610e-01 1.24193752e-01
  2.00628956e-02 9.41796147e-03]
 [4.36352508e-03 5.18290930e-03 6.76200239e-02 5.00125063e-07
  1.37957165e-02 7.66900382e-01 2.46055584e-02 4.59154393e-02
  1.20321290e-02 7.70166709e-03]
 [3.55786978e-03 2.00400401e-03 9.81849096e-02 0.00000000e+00
  7.70166709e-03 7.23721961e-02 7.32234681e-01 8.55674573e-03
  1.64657967e-02 7.70166709e-03]
 [5.18290930e-03 4.36352508e-03 7.52282271e-02 0.00000000e+00
  1.73614657e-02 1.69798525e-01 6.85357346e-03 6.35926878e-01
  7.70166709e-03 2.00628956e-02]
 [4.49788890e-02 3.28779905e-02 4.21736540e-02 0.00000000e+00
  6.01353051e-03 1.29120910e-02 7.70166709e-03 5.18290930e-03
  7.63836756e-01 3.19538442e-02]
 [1.46827092e-02 6.28784854e-02 1.64657967e-02 0.00000000e+00
  3.55786978e-03 1.91601653e-02 8.55674573e-03 2.82690315e-02
  5.72041678e-02 7.34270615e-01]]
```

Figure 19: lower bound of frequencies



```
[[0.52380936 0.02331729 0.14787341 0.0034128  0.0299371  0.02883983
  0.02220428 0.02774036 0.21162533 0.05155553]
 [0.02774036 0.69573903 0.01198647 0.         0.005996   0.02108791
  0.02663853 0.01771537 0.10218428 0.14787341]
 [0.04727049 0.00844213 0.55875281 0.0034128  0.03648078 0.18906942
  0.10531579 0.06645111 0.03756561 0.01658204]
 [0.02442719 0.01429833 0.1333861  0.0019995  0.03756561 0.52380936
  0.1715931  0.06856829 0.06220937 0.03430651]
 [0.02774036 0.01429833 0.19625333 0.         0.31449138 0.16335324
  0.16850439 0.14580625 0.0299371  0.01658204]
 [0.00963647 0.01081709 0.08437998 0.0019995  0.02220428 0.79309962
  0.03539444 0.06008456 0.01996787 0.01429833]
 [0.00844213 0.005996   0.11781509 0.         0.01429833 0.0896278
  0.75976532 0.01544325 0.0255342  0.01429833]
 [0.01081709 0.00963647 0.09277177 0.         0.02663853 0.19420148
  0.01314643 0.66607312 0.01429833 0.0299371 ]
 [0.05902111 0.04512201 0.05582635 0.         0.01198647 0.02108791
  0.01429833 0.01081709 0.79016324 0.04404616]
 [0.02331729 0.07912151 0.0255342  0.         0.00844213 0.02883983
  0.01544325 0.03973097 0.07279583 0.76172938]]
```

Figure 20: higher bound of frequencies

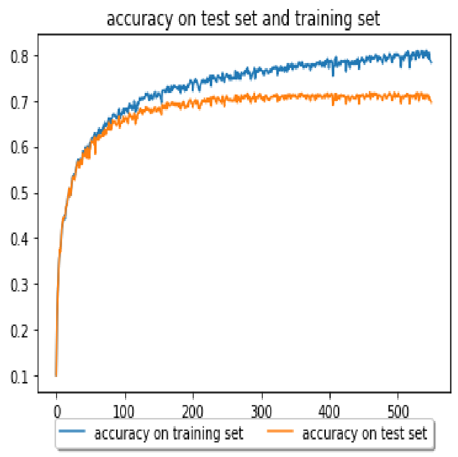


Figure 21: accuracy on training and test set

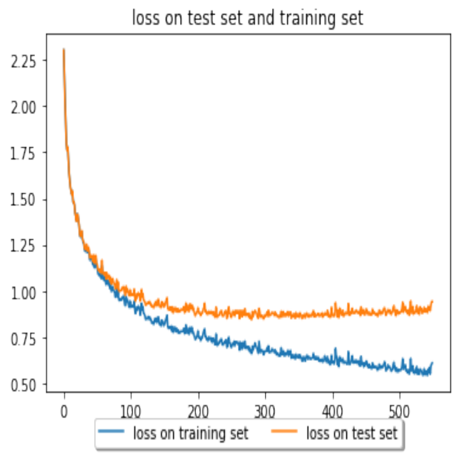


Figure 22: loss on training and test set

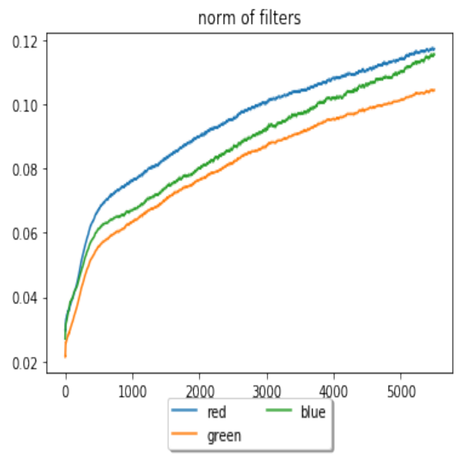


Figure 23: norm of the first kernel

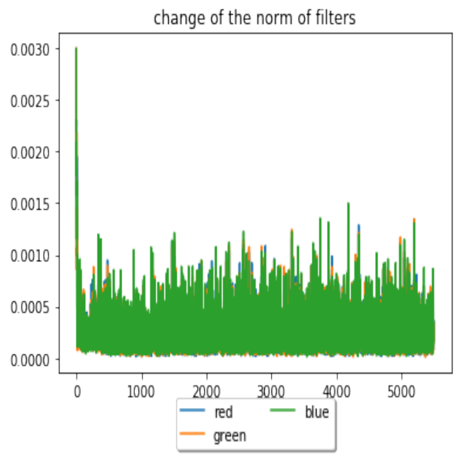
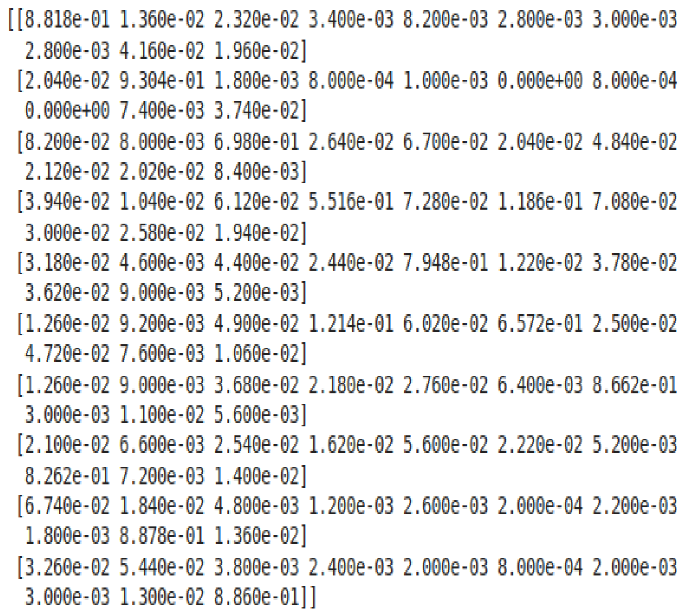


Figure 24: change of norm of first kernel

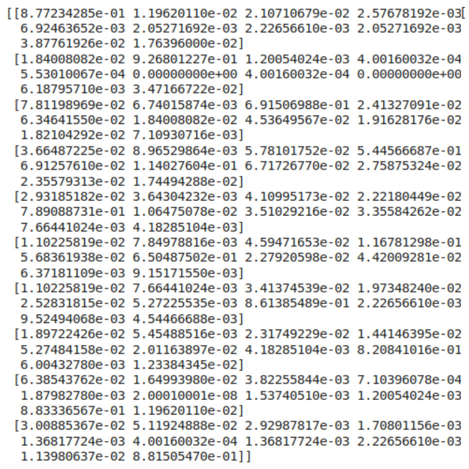
green and blue. We plot the $||K1_r(t)||$,$||K1_g(t)||$,$||K1_b(t)||$ and $||K1_r(t+1) - K1_r(t)||$,$||K1_g(t+1) - K1_g(t)||$,$||K1_b(t+1) - K1_b(t)||$ in figure 9,10. We can see the all the norms of the filter grows fast in the beginning and the speed of growing reduces and stabilizes after a while.
The confusion matrix on the training set $CM5_{train}$ is in figure25. And the lower bound and higher bound of the confusion matrix is in figure 26and 27. Compare $CM3_{train}(i,i)$ with the lower bound and higher bound of $CM5_{train}(i,i)$, we find out for $i = 2,3,4,7,8$, $CM3_{train}(i,i) > upper\ bound\ of\ \ CM5_{train}(i,i)$. For $i = 1,6,9$,$CM3_{train}(i,i) < lower\ bound\ of\ \ CM5_{train}(i,i)$

The confusion matrix on the test set $CM5_{test}$ is in figure 28. And the lower bound and higher bound of the confusion matrix is in figure 29 and 30. Compare $CM5_{test}$ with $CM3_{test}$, we can see that for $i = 1,5,6,9,10$, $CM3_{test}(i,i) < CM5_{test}(i,i)$, for $i = 3,4,7$, $CM3_{test}(i,i) > CM5_{test}(i,i)$. So it is hard to decide whether adding hidden layer improves the CNN.



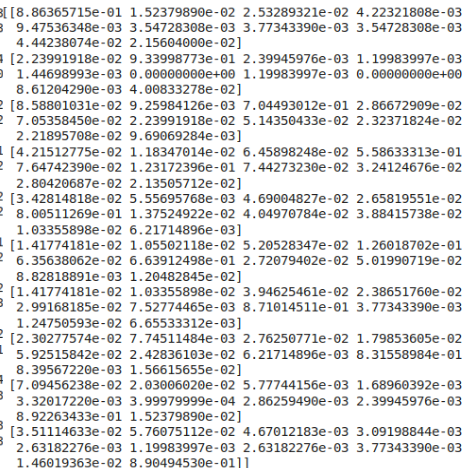
```
[[8.818e-01 1.360e-02 2.320e-02 3.400e-03 8.200e-03 2.800e-03 3.000e-03
  2.800e-03 4.160e-02 1.960e-02]
 [2.040e-02 9.304e-01 1.800e-03 8.000e-04 1.000e-03 0.000e+00 8.000e-04
  0.000e+00 7.400e-03 3.740e-02]
 [8.200e-02 8.000e-03 6.980e-01 2.640e-02 6.700e-02 2.040e-02 4.840e-02
  2.120e-02 2.020e-02 8.400e-03]
 [3.940e-02 1.040e-02 6.120e-02 5.516e-01 7.280e-02 1.186e-01 7.080e-02
  3.000e-02 2.580e-02 1.940e-02]
 [3.180e-02 4.600e-03 4.400e-02 2.440e-02 7.948e-01 1.220e-02 3.780e-02
  3.620e-02 9.000e-03 5.200e-03]
 [1.260e-02 9.200e-03 4.900e-02 1.214e-01 6.020e-02 6.572e-01 2.500e-02
  4.720e-02 7.600e-03 1.060e-02]
 [1.260e-02 9.000e-03 3.680e-02 2.180e-02 2.760e-02 6.400e-03 8.662e-01
  3.000e-03 1.100e-02 5.600e-03]
 [2.100e-02 6.600e-03 2.540e-02 1.620e-02 5.600e-02 2.220e-02 5.200e-03
  8.262e-01 7.200e-03 1.400e-02]
 [6.740e-02 1.840e-02 4.800e-03 1.200e-03 2.600e-03 2.000e-04 2.200e-03
  1.800e-03 8.878e-01 1.360e-02]
 [3.260e-02 5.440e-02 3.800e-03 2.400e-03 2.000e-03 8.000e-04 2.000e-03
  3.000e-03 1.300e-02 8.860e-01]]
```

Figure 25: Confusion matrix on training set



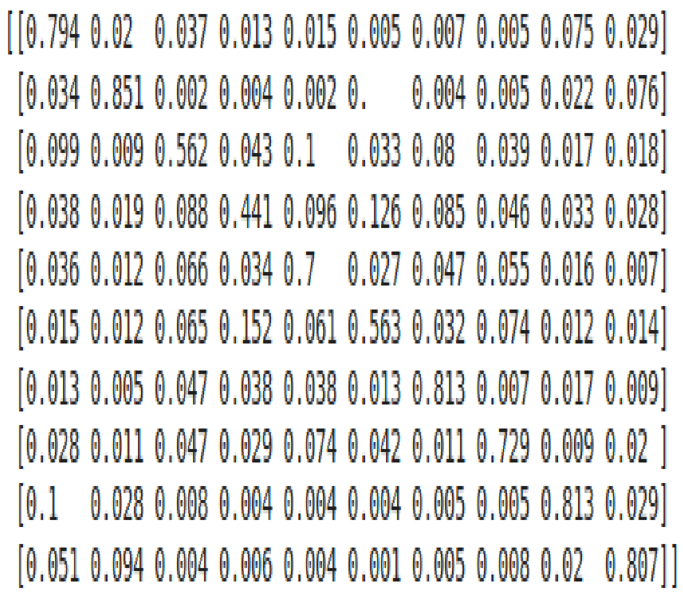
```
[[8.77234285e-01 1.19620110e-02 2.10710679e-02 2.57678192e-03
  6.92463652e-03 2.05271692e-03 2.22656610e-03 2.05271692e-03
  3.87761926e-02 1.76396000e-02]
 [1.84008082e-02 9.26801227e-01 1.20054024e-03 4.00160032e-04
  5.53010067e-04 0.00000000e+00 4.00160032e-04 0.00000000e+00
  6.18795710e-03 3.47166722e-02]
 [7.81198969e-02 6.74015874e-03 6.91506988e-01 2.41327091e-02
  6.34641550e-02 1.84008082e-02 4.53649567e-02 1.91628176e-02
  1.82104292e-02 7.10930716e-03]
 [3.66487225e-02 8.96529864e-03 5.78101752e-02 5.44566687e-01
  6.91257610e-02 1.14027604e-01 6.71726770e-02 2.75875324e-02
  2.35579313e-02 1.74494288e-02]
 [2.93185182e-02 3.64304232e-03 4.10995173e-02 2.22180449e-02
  7.89088731e-01 1.06475078e-02 3.51029216e-02 3.35584262e-02
  7.66441024e-03 4.18285104e-03]
 [1.10225819e-02 7.84978816e-03 4.59471653e-02 1.16781298e-01
  5.68361938e-02 6.50487502e-01 2.27920598e-02 4.42009281e-02
  6.37181109e-03 9.15171550e-03]
 [1.10225819e-02 7.66441024e-03 3.41374539e-02 1.97348240e-02
  2.52831815e-02 5.27225535e-03 8.61385489e-01 2.22656610e-03
  9.52494068e-03 4.54466688e-03]
 [1.89722426e-02 5.45488516e-03 2.31749229e-02 1.44146395e-02
  5.27484158e-02 2.01163897e-02 4.18285104e-03 8.20841016e-01
  6.00432780e-03 1.23384345e-02]
 [6.38543762e-02 1.64993980e-02 3.82255844e-03 7.10396078e-04
  1.87982780e-03 2.00010001e-08 1.53740510e-03 1.20054024e-03
  8.83336567e-01 1.19620110e-02]
 [3.00885367e-02 5.11924888e-02 2.92987817e-03 1.70801156e-03
  1.36817724e-03 4.00160032e-04 1.36817724e-03 2.22656610e-03
  1.13980637e-02 8.81505470e-01]]
```

Figure 26: lower bound of frequencies



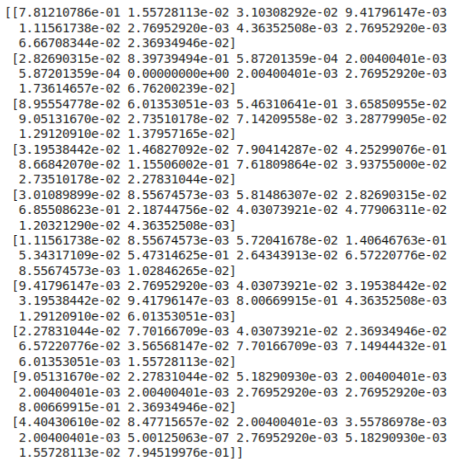
```
[[8.86365715e-01 1.52379890e-02 2.53289321e-02 4.22321808e-03
  9.47536348e-03 3.54728308e-03 3.77343390e-03 3.54728308e-03
  4.44238074e-02 2.15604000e-02]
 [2.23991918e-02 9.33998773e-01 2.39945976e-03 1.19983997e-03
  1.44698993e-03 0.00000000e+00 1.19983997e-03 0.00000000e+00
  8.61204290e-03 4.00833278e-02]
 [8.58801031e-02 9.25984126e-03 7.04493012e-01 2.86672909e-02
  7.05358450e-02 2.23991918e-02 5.14350433e-02 2.32371824e-02
  2.21895708e-02 9.69069284e-03]
 [4.21512775e-02 1.18347014e-02 6.45898248e-02 5.58633313e-01
  7.64742390e-02 1.23172396e-01 7.44273230e-02 3.24124676e-02
  2.80420687e-02 2.13505712e-02]
 [3.42814818e-02 5.55695768e-03 4.69004827e-02 2.65819551e-02
  8.00511269e-01 1.37524922e-02 4.04970784e-02 3.88415738e-02
  1.03355898e-02 6.21714896e-03]
 [1.41774181e-02 1.05502118e-02 5.20528347e-02 1.26018702e-01
  6.35638062e-02 6.63912498e-01 2.72079402e-02 5.01990719e-02
  8.82818891e-03 1.20482845e-02]
 [1.41774181e-02 1.03355898e-02 3.94625461e-02 2.38651760e-02
  2.99168185e-02 7.52774465e-03 8.71014511e-01 3.77343390e-03
  1.24750593e-02 6.65533312e-03]
 [2.30277574e-02 7.74511484e-03 2.76250771e-02 1.79853605e-02
  5.92515842e-02 2.42836103e-02 6.21714896e-03 8.31558984e-01
  8.39567220e-03 1.56615655e-02]
 [7.09456238e-02 2.03006020e-02 5.77744156e-03 1.68960392e-03
  3.32017220e-03 3.99979999e-04 2.86259490e-03 2.39945976e-03
  8.92263433e-01 1.52379890e-02]
 [3.51114633e-02 5.76075112e-02 4.67012183e-03 3.09198844e-03
  2.63182276e-03 1.19983997e-03 2.63182276e-03 3.77343390e-03
  1.46019363e-02 8.90494530e-01]]
```

Figure 27: higher bound of frequencies



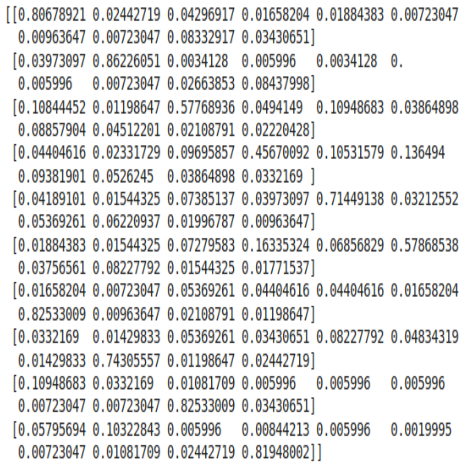
```
[[0.794 0.02  0.037 0.013 0.015 0.005 0.007 0.005 0.075 0.029]
 [0.034 0.851 0.002 0.004 0.002 0.    0.004 0.005 0.022 0.076]
 [0.099 0.009 0.562 0.043 0.1   0.033 0.08  0.039 0.017 0.018]
 [0.038 0.019 0.088 0.441 0.096 0.126 0.085 0.046 0.033 0.028]
 [0.036 0.012 0.066 0.034 0.7   0.027 0.047 0.055 0.016 0.007]
 [0.015 0.012 0.065 0.152 0.061 0.563 0.032 0.074 0.012 0.014]
 [0.013 0.005 0.047 0.038 0.038 0.013 0.813 0.007 0.017 0.009]
 [0.028 0.011 0.047 0.029 0.074 0.042 0.011 0.729 0.009 0.02 ]
 [0.1   0.028 0.008 0.004 0.004 0.004 0.005 0.005 0.813 0.029]
 [0.051 0.094 0.004 0.006 0.004 0.001 0.005 0.008 0.02  0.807]]
```

Figure 28: Confusion matrix on test set



```
[[7.81210786e-01 1.55728113e-02 3.10308292e-02 9.41796147e-03
  1.11561738e-02 2.76952920e-03 4.36352508e-03 2.76952920e-03
  6.66708344e-02 2.36934946e-02]
 [2.82690315e-02 8.39739494e-01 5.87201359e-04 2.00400401e-03
  5.87201359e-04 0.00000000e+00 2.00400401e-03 2.76952920e-03
  1.73614657e-02 6.76200239e-02]
 [8.95554778e-02 6.01353051e-03 5.46310641e-01 3.65850955e-02
  9.05131670e-02 2.73510178e-02 7.14209558e-02 3.28779905e-02
  1.29120910e-02 1.37957165e-02]
 [3.19538442e-02 1.46827092e-02 7.90414287e-02 4.25299076e-01
  8.66842070e-02 1.15506002e-01 7.61809864e-02 3.93755000e-02
  2.73510178e-02 2.27831044e-02]
 [3.01089899e-02 8.55674573e-03 5.81486307e-02 2.82690315e-02
  6.85508623e-01 2.18744756e-02 4.03073921e-02 4.77906311e-02
  1.20321290e-02 4.36352508e-03]
 [1.11561738e-02 8.55674573e-03 5.72041678e-02 1.40646763e-01
  5.34317109e-02 5.47314625e-01 2.64343913e-02 6.57220776e-02
  8.55674573e-03 1.02846265e-02]
 [9.41796147e-03 2.76952920e-03 4.03073921e-02 3.19538442e-02
  3.19538442e-02 9.41796147e-03 8.00669915e-01 4.36352508e-03
  1.29120910e-02 6.01353051e-03]
 [2.27831044e-02 7.70166709e-03 4.03073921e-02 2.36934946e-02
  6.57220776e-02 3.56568147e-02 7.70166709e-03 7.14944432e-01
  6.01353051e-03 1.55728113e-02]
 [9.05131670e-02 2.27831044e-02 5.18290930e-03 2.00400401e-03
  2.00400401e-03 2.00400401e-03 2.76952920e-03 2.76952920e-03
  8.00669915e-01 2.36934946e-02]
 [4.40430610e-02 8.47715657e-02 2.00400401e-03 3.55786978e-03
  2.00400401e-03 5.00125063e-07 2.76952920e-03 5.18290930e-03
  1.55728113e-02 7.94519976e-01]]
```

Figure 29: lower bound of frequencies



```
[[0.80678921 0.02442719 0.04296917 0.01658204 0.01884383 0.00723047
  0.00963647 0.00723047 0.08332917 0.03430651]
 [0.03973097 0.86226051 0.0034128  0.005996   0.0034128  0.
  0.005996   0.00723047 0.02663853 0.08437998]
 [0.10844452 0.01198647 0.57768936 0.0494149  0.10948683 0.03864898
  0.08857904 0.04512201 0.02108791 0.02220428]
 [0.04404616 0.02331729 0.09695857 0.45670092 0.10531579 0.136494
  0.09381901 0.0526245  0.03864898 0.0332169 ]
 [0.04189101 0.01544325 0.07385137 0.03973097 0.71449138 0.03212552
  0.05369261 0.06220937 0.01996787 0.00963647]
 [0.01884383 0.01544325 0.07279583 0.16335324 0.06856829 0.57868538
  0.03756561 0.08227792 0.01544325 0.01771537]
 [0.01658204 0.00723047 0.05369261 0.04404616 0.04404616 0.01658204
  0.82533009 0.00963647 0.02108791 0.01198647]
 [0.0332169  0.01429833 0.05369261 0.03430651 0.08227792 0.04834319
  0.01429833 0.74305557 0.01198647 0.02442719]
 [0.10948683 0.0332169  0.01081709 0.005996   0.005996   0.005996
  0.00723047 0.00723047 0.82533009 0.03430651]
 [0.05795694 0.10322843 0.005996   0.00844213 0.005996   0.0019995
  0.00723047 0.01081709 0.02442719 0.81948002]]
```

Figure 30: higher bound of frequencies