MACHINE VISION HW4

Soheil Shirvani

810195416

Question 1

- Part 1:

  The CIE XYZ color space encompasses all color sensations that are visible to a person with average eyesight. That is why CIE XYZ (Tristimulus values) is a device-invariant representation of color. It serves as a standard reference against which many other color spaces are defined.

  The CIE model capitalizes on this fact by setting Y as luminance. Z is quasi-equal to blue, or the S cone response, and X is a mix of response curves chosen to be nonnegative. The XYZ tristimulus values are thus analogous to, but different from, the LMS cone responses of the human eye. Setting Y as luminance has the useful result that for any given Y value, the XZ plane will contain all possible chromaticities at that luminance.

  The unit of the tristimulus values X, Y, and Z is often arbitrarily chosen so that Y = 1 or Y = 100 is the brightest white that a color display supports. In this case, the Y value is known as the relative luminance. The corresponding whitepoint values for X and Z can then be inferred using the standard illuminants.

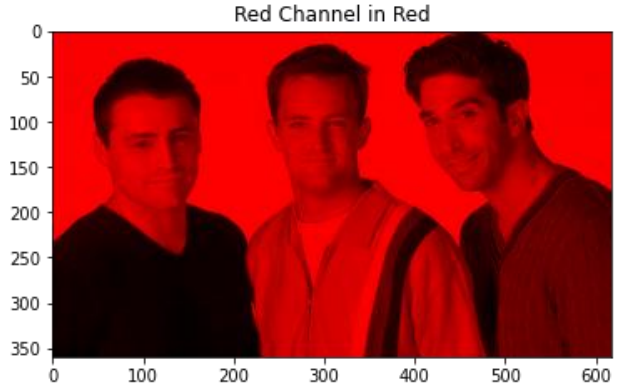
Because the "red", "green" and "blue" which your monitor uses are pale, probably not noticeable but still pale. It is surprising that your monitor used distinguishably pale colours and was said to have small colour space.

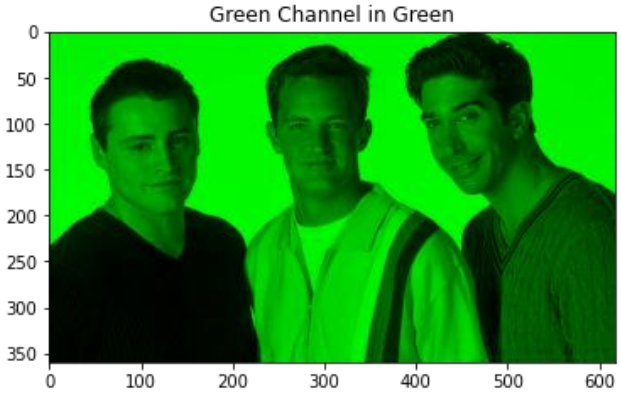
No matter how pale the "red", "green" and "blue" (and ANY other set of three different colours) are, it is always possible to reproduce a colour with them if you may have negative amount of each. But, this is not possible physically.

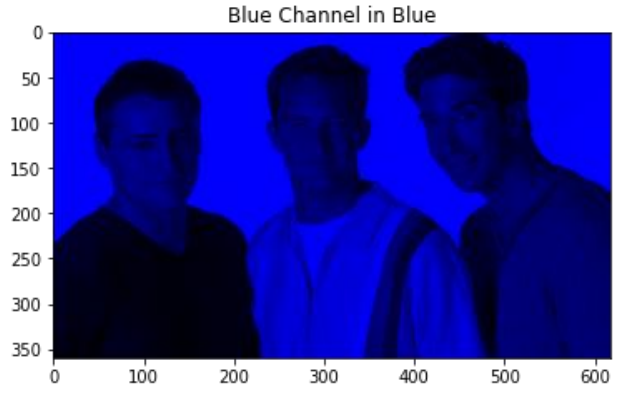
No matter how saturated the "X", "Y" and "Z" are you cannot practically reproduce arbitrary visible colour with them, even if they are monochromatic (fully saturated), see reasoning below.

- Part 2:

Here we plot all the channels of an input image:





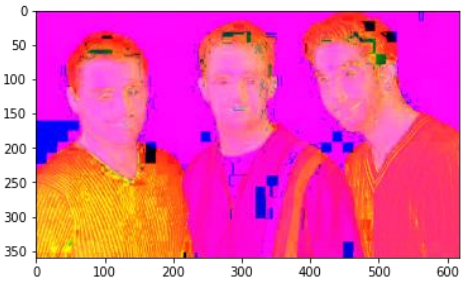


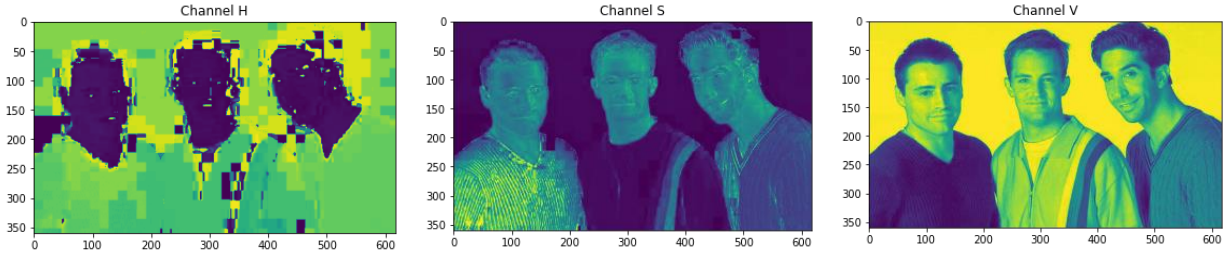
Part2 Continue: RGB vs sRGB:

sRGB is an RGB (red, green, blue) color space that HP and Microsoft created cooperatively in 1996 to use on monitors, printers, and the Web. It is often the "default" color space for images that contain no color space information, especially if the images' pixels are stored in 8-bit integers per color channel. This specification allowed sRGB to be directly displayed on typical CRT monitors of the time, which greatly aided its acceptance. sRGB defines the chromaticities of the red, green, and blue primaries, the colors where one of the three channels is nonzero and the other two are zero. The gamut of chromaticities that can be represented in sRGB is the color triangle defined by these primaries. As with any RGB color space, for non-negative values of R, G, and B it is not possible to represent colors outside this triangle, which is well inside the range of colors visible to a human with normal trichromatic vision.

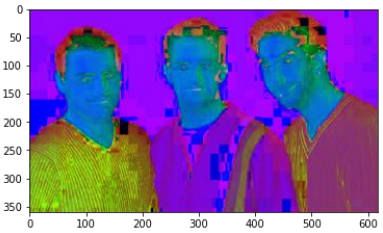
- Part 3)

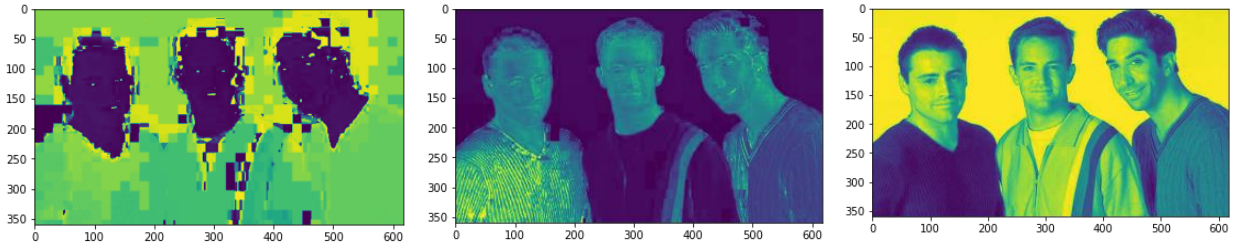
  Here we are going to convert the image to HSV and plot each channel. After Plotting we have:





And if you use OpenCV Library we have:





Which are almost the same.

Our function is like Below:

```
def rgb2hsv(r, g, b):
    r, g, b = r/255.0, g/255.0, b/255.0
    mx = max(r, g, b)
    mn = min(r, g, b)
    df = mx-mn
    if mx == mn:
        h = 0
    elif mx == r:
        h = (60 * ((g-b)/df) + 360) % 360
    elif mx == g:
        h = (60 * ((b-r)/df) + 120) % 360
    elif mx == b:
        h = (60 * ((r-g)/df) + 240) % 360
    if mx == 0:
        s = 0
    else:
        s = df/mx
    v = mx
    return h, s, v
```

Part 3 Continue) HSV (hue, saturation, value, also known as HSB or hue, saturation, brightness)

HUE

Hue is the color portion of the model, expressed as a number from 0 to 360 degrees:

- **Red** falls between 0 and 60 degrees.
- **Yellow** falls between 61 and 120 degrees.
- **Green** falls between 121 and 180 degrees.
- **Cyan** falls between 181 and 240 degrees.
- **Blue** falls between 241 and 300 degrees.
- **Magenta** falls between 301 and 360 degrees.

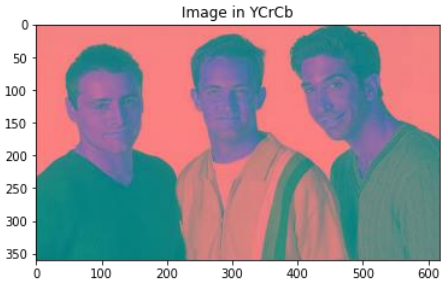
SATURATION

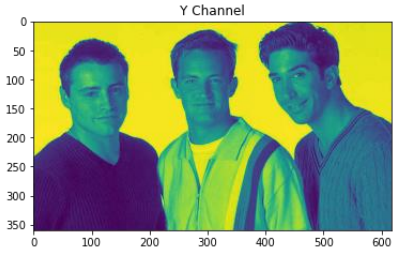
Saturation describes the amount of gray in a particular color, from 0 to 100 percent. Reducing this component toward zero introduces grayer and produces a faded effect. Sometimes, saturation appears as a range from 0 to 1, where 0 is gray, and 1 is a primary color.

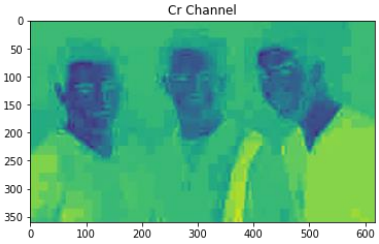
VALUE (OR BRIGHTNESS)

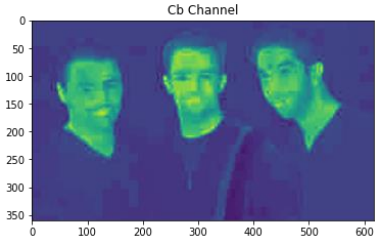
Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0 to 100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color.

- Part 4) Here we are going to convert our image to Y'CbCr, here we have:

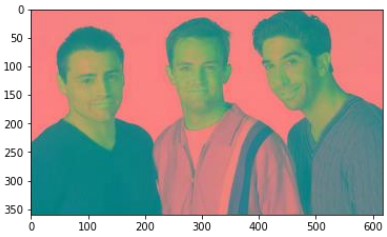


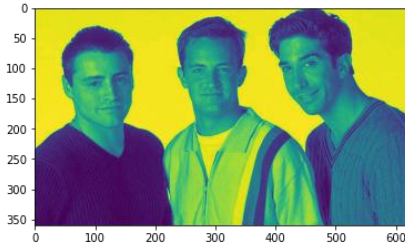


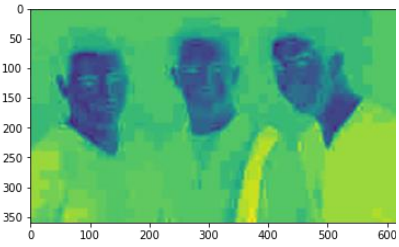


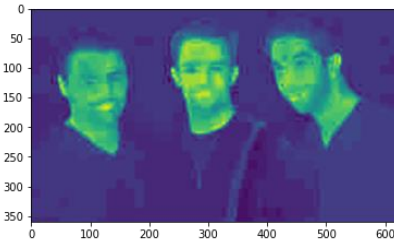


And if we use Library we have:









And the function which did this is:

```python
def bgr2ycbcr(bgr_image):
    rows = bgr_image.shape[0]
    cols = bgr_image.shape[1]

    YCrCv_image = np.zeros(shape = bgr_image.shape)

    print(rows,cols)

    for i in range(0, rows):
        for j in range(0, cols):
            R = bgr_image[i, j][2]
            G = bgr_image[i, j][1]
            B = bgr_image[i, j][0]

            Y =   0  +    0.299*R +    0.587*G  + 0.114*B
            Cb = 128 - 0.168736*R - 0.331264*G  + 0.5*B
            Cr = 128 + 0.5*R      -  0.418688*G -  0.081312*B

            YCrCv_image[i, j] = Y, Cr, Cb

    return YCrCv_image
```

Part 4 Continue)

The difference between YCbCr and RGB is that RGB represents colors as combinations of red, green and blue signals, while YCbCr represents colors as combinations of a brightness signal and two chroma signals. In YCbCr, Y is luma (brightness), Cb is blue minus luma (B-Y) and Cr is red minus luma (R-Y). The luma channel, typically denoted Y (more accurately Y', indicating that the channel is gamma encoded), approximates the monochrome picture content. The two chroma channels, Cb and Cr, are color difference channels. During creation of YCbCr signals from RGB (the encoding process), higher frequency signal content is removed from the Cb and Cr channels to compress the signal. YCbCr video is considered a form of lossless compression, as the elements of the original RGB signal that are removed in creating YCbCr are picture elements that humans aren't able to see at normal viewing distances.

RF tuners, cable and satellite set top boxes, and DVD/Blu-ray disk players all natively transmit or store YCbCr format digital video signals, since each of these limited bandwidth transmission and storage mediums benefit from signal compression. The native video signals produced by computers and game consoles are typically uncompressed RGB signal format, however, since their transmission mediums do not benefit from signal compression.

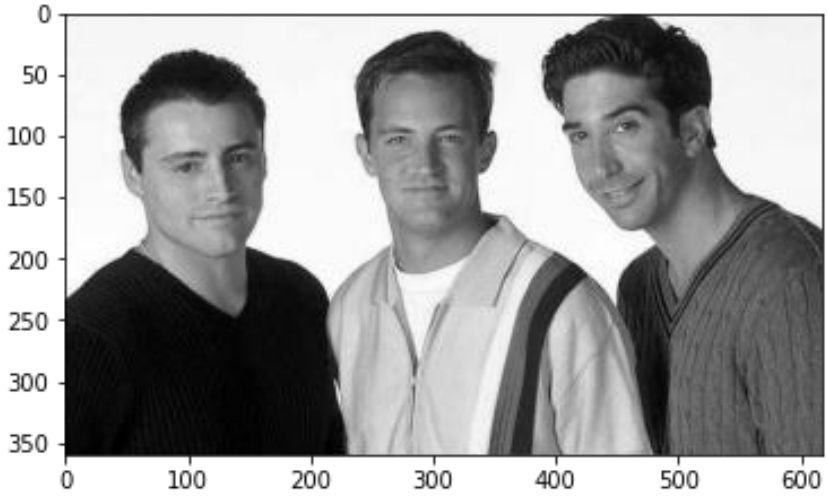
Human retinal receptors respond to light energy in terms of its red, green, and blue light components, so RGB light control works best for imaging display devices. Hence, the need for a color decoder at the end of a display's signal processing path to transform color signals from the YCbCr color space back to the RGB color space for presentation on an imaging display device.

Digital Y′CbCr (8 bits per sample) is derived from analog R'G'B' as follows:

$$\begin{aligned} Y' &= 16+ (65.481 \cdot R' + 128.553 \cdot G' + 24.966 \cdot B') \\ C_B &= 128+ (-37.797 \cdot R' - 74.203 \cdot G' + 112.0 \cdot B') \\ C_R &= 128+ (112.0 \cdot R' - 93.786 \cdot G' - 18.214 \cdot B') \end{aligned}$$

- Part 5)

In this part we are going to convert our image to Gray Scale
This is our Reslut:



We did this transformation by multiplying [0.299, 0.587, 0.144] to our RGB image and put the equal channel for all 3 channels. Like below:

```
def bgr2gray(bgr_image):
    rows = bgr_image.shape[0]
    cols = bgr_image.shape[1]

    gray_image = np.zeros(shape = bgr_image.shape)

    print(rows,cols)

    for i in range(0, rows):
        for j in range(0, cols):
            R = bgr_image[i, j][2]
            G = bgr_image[i, j][1]
            B = bgr_image[i, j][0]

            L = R * 299/1000 + G * 587/1000 + B * 114/1000

            gray_image[i, j] = L

    return gray_image[:,:,0]
```

```
just_gray = np.zeros((gray.shape[0],gray.shape[1],3), dtype=int)
just_gray[:,:,0] = gray
just_gray[:,:,1] = gray
just_gray[:,:,2] = gray
```

- Part 6)

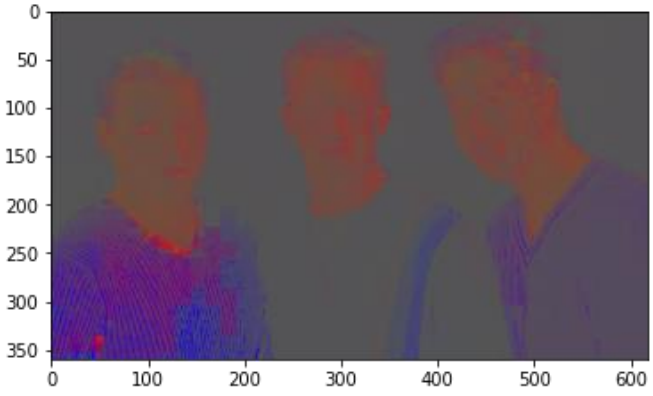
  Here we are going to generate a mask so we can extract the faces of the actors from other parts of the image.

  First, we know our input image is:

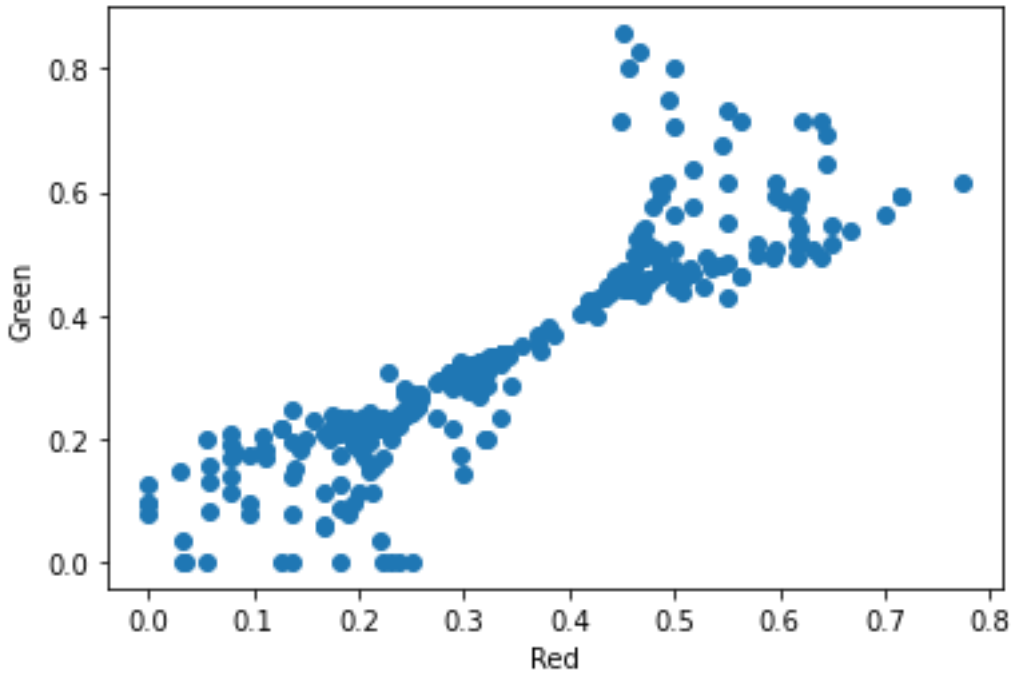
  Then we are going to normalize the image using the code blue:

```
normal_image = np.zeros(shape=img.shape)

for i in range(0, img.shape[0]):
    for j in range(0, img.shape[1]):
        R = float(img[i, j][2])
        G = float(img[i, j][1])
        B = float(img[i, j][0])

        if(B == G == R == 0):
            B = G = R = 1

        RGB_sum = (B + G + R)
        r = R/RGB_sum
        g = G/RGB_sum
        b = B/RGB_sum

#           print(B,G,R)
#           print(RGB_sum)
#           print(b,g,r)
#           print()

        normal_image[i, j] = b, g, r
```

  Which Results in Same Image:



Now if plot the Red and Green value Channels of our image we can see:



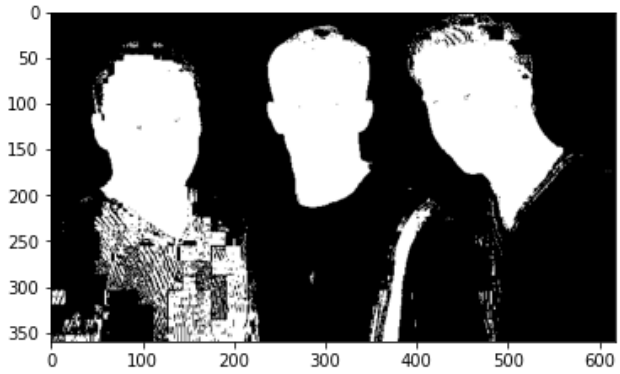

```
1 normal_image[300,100]
```

```
array([0.38235294, 0.14705882, 0.47058824])
```

From the normal image we can see that the faces have more red values and red is more important in that areas. So by cutting off the red value we can see that the images are becoming darker and their faces are becoming white which is what we want. Our cut off is like below:

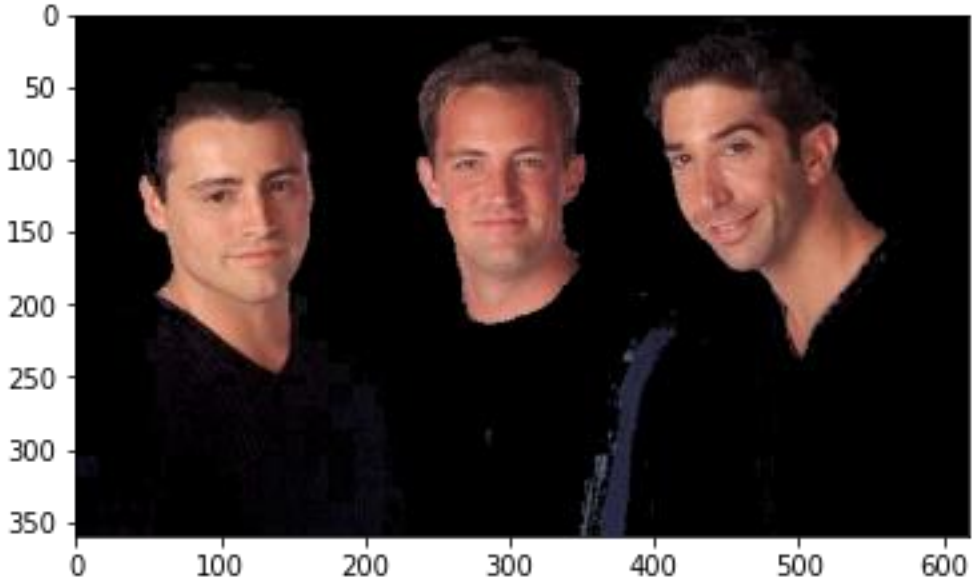
```
if (0.3 >= normal_image[i,j,0] or normal_image[i,j,0] >= 0.37) and (0.1 <= normal_image[i,j,1] <= 0.4):
    mask[i,j] = 1,1,1
```

And the result image is:



Which is almost what we wanted since the faces are extracted from the whole image.

Now if we apply our mask to our first image, we can see the image below:



Which is the image of faces. Although there are some disturbance in the last image (it is because some areas of image are alike the faces) but faces are well extracted.

We found the cutoff for green and red channels by experiments so they may not be optimal nor unique but here was a good result of that.

Question 2

Here we are going to implement a 2 layer fully connected neural network for classification of Cifer10 dataset.

Our constraints are to have Relu as activation function of first hidden layer, SoftMax as activation function of output layer. Our optimizer is SGD with 128 batch size so Mini Batch SGD, we have 0.9 momentum and using learning rate decay for this implementation. Number of neurons in hidden layer is 150. Also, we assign almost 0 to all the weight vectors initially. Due to maybe bad implementation, it was impossible to set Learning rate to 0.001, so instead in base model it was set to 0.0001 and we change it to 0.001 and 0.00001 later.

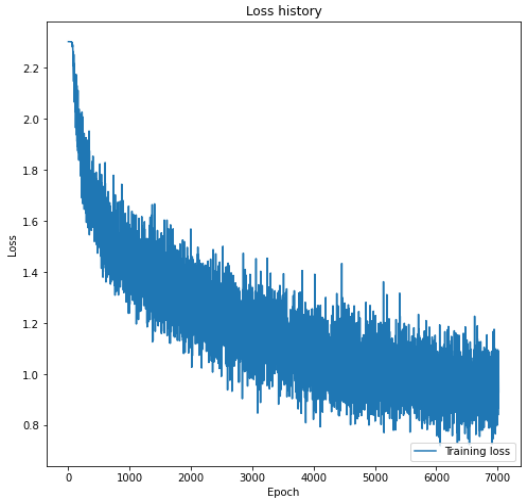
We have a forward path in which our input vector and first layer weight vector will be multiplied and pass through Relu. Then the answer will be multiplied to second weight vector and passes through SoftMax to detect. Negative Log Likelihood will be calculated then in the backward pass the gradients will be measured and will be used to update the parameters.

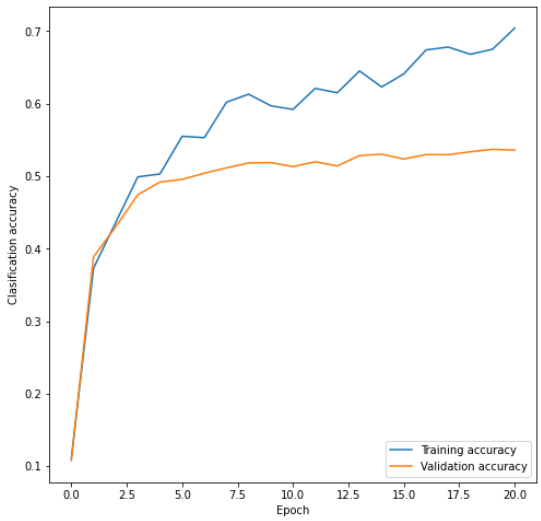
This is our base model for this question.

Because of resource limitation models are trained in google colab.

1. Base model answer:

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.111000, validation accuracy - 0.102400, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 1.685500, train accuracy - 0.406000, validation accuracy - 0.391000, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 1.590621, train accuracy - 0.464000, validation accuracy - 0.448000, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 1.440223, train accuracy - 0.506000, validation accuracy - 0.472000, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 1.294530, train accuracy - 0.536000, validation accuracy - 0.492800, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 1.220404, train accuracy - 0.520000, validation accuracy - 0.495400, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 1.203078, train accuracy - 0.559000, validation accuracy - 0.503000, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 1.247928, train accuracy - 0.575000, validation accuracy - 0.517800, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 1.162617, train accuracy - 0.568000, validation accuracy - 0.517000, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 1.199291, train accuracy - 0.602000, validation accuracy - 0.524000, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 1.159579, train accuracy - 0.624000, validation accuracy - 0.519800, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 1.236229, train accuracy - 0.644000, validation accuracy - 0.525200, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 1.056653, train accuracy - 0.622000, validation accuracy - 0.524000, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 1.158512, train accuracy - 0.646000, validation accuracy - 0.530000, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 1.128528, train accuracy - 0.651000, validation accuracy - 0.531200, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 0.922983, train accuracy - 0.655000, validation accuracy - 0.530600, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 0.976936, train accuracy - 0.664000, validation accuracy - 0.531200, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 1.052893, train accuracy - 0.662000, validation accuracy - 0.531200, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 1.162200, train accuracy - 0.687000, validation accuracy - 0.530000, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 1.054632, train accuracy - 0.678000, validation accuracy - 0.538600, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 0.931533, train accuracy - 0.681000, validation accuracy - 0.532600, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.538600
```





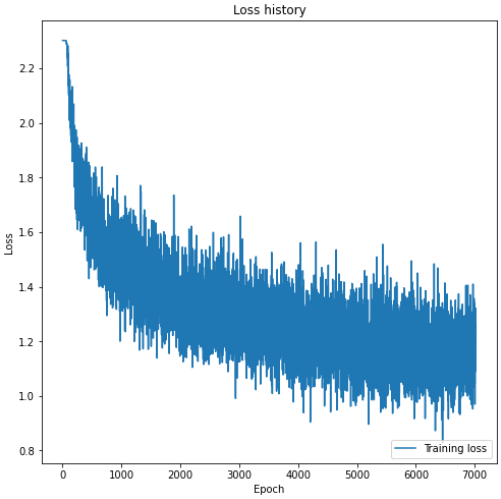

```
Test accuracy:  0.5183
```

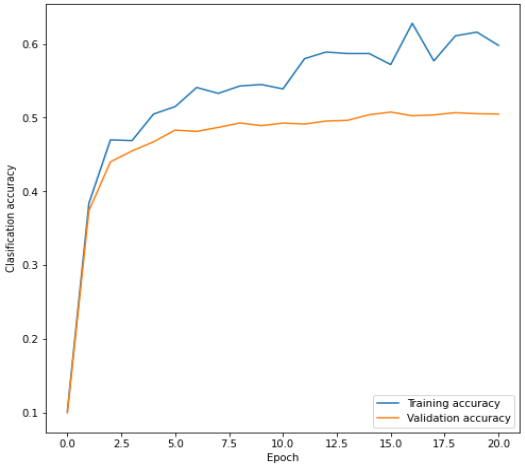
After running the base model, a test accuracy of 51% achieved. As the plots shows, after epoch 8-10 model started to overfitted to the training data since the validation accuracy did not rise as the training accuracy did. It means our model only did well on the training dataset and may not be a good generalized model for the unseen data (Test data). There could be a couple of ways to stop the overfitting in which some of them are tested in the following parts. Other ways like early stopping or weight decaying could be solutions too.

Since it was our base model, we change the hyper parameters to see the impact of them on our accuracy and overfitting processes in the following sections.

2. Now we change the Neurons of our hidden layer to **50**

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.102000, validation accuracy - 0.099800, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 1.781497, train accuracy - 0.384000, validation accuracy - 0.374000, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 1.591052, train accuracy - 0.470000, validation accuracy - 0.440200, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 1.534569, train accuracy - 0.469000, validation accuracy - 0.455000, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 1.470776, train accuracy - 0.505000, validation accuracy - 0.467400, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 1.226751, train accuracy - 0.515000, validation accuracy - 0.483200, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 1.517072, train accuracy - 0.541000, validation accuracy - 0.481400, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 1.305330, train accuracy - 0.533000, validation accuracy - 0.486800, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 1.187239, train accuracy - 0.543000, validation accuracy - 0.492800, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 1.278775, train accuracy - 0.545000, validation accuracy - 0.489200, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 1.335673, train accuracy - 0.539000, validation accuracy - 0.492600, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 1.305334, train accuracy - 0.580000, validation accuracy - 0.491400, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 1.089951, train accuracy - 0.589000, validation accuracy - 0.495400, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 1.336762, train accuracy - 0.587000, validation accuracy - 0.496400, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 1.242899, train accuracy - 0.587000, validation accuracy - 0.504000, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 1.177686, train accuracy - 0.572000, validation accuracy - 0.507600, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 1.183359, train accuracy - 0.628000, validation accuracy - 0.502600, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 1.141293, train accuracy - 0.577000, validation accuracy - 0.503800, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 1.253339, train accuracy - 0.611000, validation accuracy - 0.506800, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 1.122087, train accuracy - 0.616000, validation accuracy - 0.505400, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 1.263462, train accuracy - 0.598000, validation accuracy - 0.505000, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.507600
```

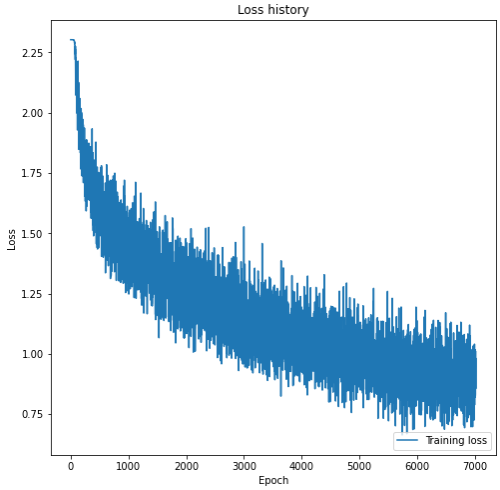


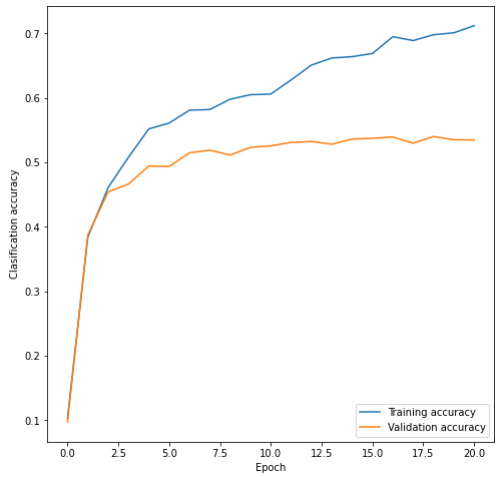



```
Test accuracy:  0.4968
```

Now we change the Neurons of our hidden layer to **200**

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.086000, validation accuracy - 0.095200, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 1.587746, train accuracy - 0.403000, validation accuracy - 0.377400, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 1.380195, train accuracy - 0.438000, validation accuracy - 0.448800, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 1.462744, train accuracy - 0.523000, validation accuracy - 0.485200, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 1.308063, train accuracy - 0.494000, validation accuracy - 0.492600, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 1.161809, train accuracy - 0.529000, validation accuracy - 0.509600, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 1.238418, train accuracy - 0.553000, validation accuracy - 0.503400, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 1.078622, train accuracy - 0.572000, validation accuracy - 0.520600, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 1.194606, train accuracy - 0.608000, validation accuracy - 0.520600, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 1.219792, train accuracy - 0.614000, validation accuracy - 0.519600, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 0.996143, train accuracy - 0.640000, validation accuracy - 0.522000, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 1.160946, train accuracy - 0.626000, validation accuracy - 0.526200, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 1.036290, train accuracy - 0.643000, validation accuracy - 0.528800, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 1.011243, train accuracy - 0.662000, validation accuracy - 0.522800, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 1.228372, train accuracy - 0.656000, validation accuracy - 0.530000, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 0.967270, train accuracy - 0.671000, validation accuracy - 0.529000, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 0.838765, train accuracy - 0.699000, validation accuracy - 0.534800, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 1.003663, train accuracy - 0.692000, validation accuracy - 0.532600, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 0.892827, train accuracy - 0.705000, validation accuracy - 0.534600, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 0.921280, train accuracy - 0.715000, validation accuracy - 0.533000, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 1.008986, train accuracy - 0.672000, validation accuracy - 0.533600, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.534800
```





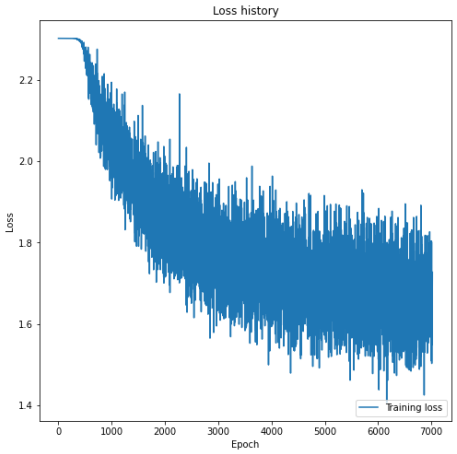

```
Test accuracy:  0.521
```

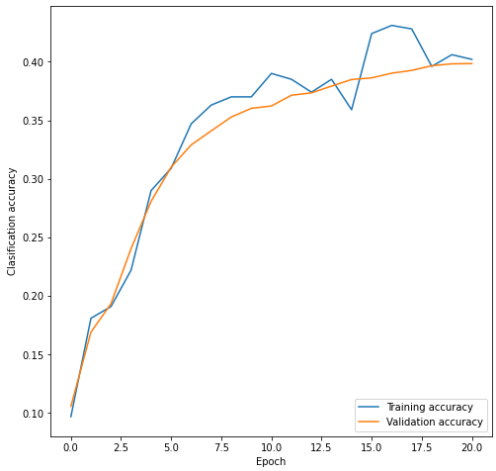
In this part we change the number of neurons in our hidden layer. As the plots shows when we had 50 neurons on hidden layer a test accuracy of 49% achieved and when it was 200 our test accuracy was 52%. Plots shows that despite the 49% when 50 neurons are on hidden layer, we can see less overfitting happened. In 200 neurons, the training and validating accuracies are diverging a lot and more epochs means more overfitting. So, we can conclude that even with 50 neurons we reached a lower accuracy it may be better to use for unseen data since it can be generalized better. If the epochs were larger, we could see more overfitting on 200 neurons since validating accuracy would not rise as the training accuracy but this may happen less in 50 neurons.

3. Now we change the learning rate to **0.00001**

```
classifier = TwoLayerNeuralNet(32*32*3, 150, 10)  # initialize the neural net
best_model, loss_history, train_history, val_history = \
    classifier.train(x_train, y_train, x_val, y_val, reg=0.00001, lr=1e-5, momentum=0.9, lr_decay=0.9,
                     method='momentum', mini_batch_SGD=True, num_epoch=20, batch_size=128)
```

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.097000, validation accuracy - 0.105800, learning rate - 1.000000e-05
Finished epoch 1 / 20: loss - 2.300609, train accuracy - 0.181000, validation accuracy - 0.169200, learning rate - 9.000000e-06
Finished epoch 2 / 20: loss - 2.135010, train accuracy - 0.191000, validation accuracy - 0.193400, learning rate - 8.100000e-06
Finished epoch 3 / 20: loss - 2.015020, train accuracy - 0.222000, validation accuracy - 0.240400, learning rate - 7.290000e-06
Finished epoch 4 / 20: loss - 1.967445, train accuracy - 0.290000, validation accuracy - 0.280800, learning rate - 6.561000e-06
Finished epoch 5 / 20: loss - 1.763819, train accuracy - 0.309000, validation accuracy - 0.310000, learning rate - 5.904900e-06
Finished epoch 6 / 20: loss - 1.759313, train accuracy - 0.347000, validation accuracy - 0.329000, learning rate - 5.314410e-06
Finished epoch 7 / 20: loss - 1.925868, train accuracy - 0.363000, validation accuracy - 0.341000, learning rate - 4.782969e-06
Finished epoch 8 / 20: loss - 1.623649, train accuracy - 0.370000, validation accuracy - 0.352800, learning rate - 4.304672e-06
Finished epoch 9 / 20: loss - 1.771302, train accuracy - 0.370000, validation accuracy - 0.360200, learning rate - 3.874205e-06
Finished epoch 10 / 20: loss - 1.599304, train accuracy - 0.390000, validation accuracy - 0.362200, learning rate - 3.486784e-06
Finished epoch 11 / 20: loss - 1.713607, train accuracy - 0.385000, validation accuracy - 0.371400, learning rate - 3.138106e-06
Finished epoch 12 / 20: loss - 1.746380, train accuracy - 0.374000, validation accuracy - 0.373400, learning rate - 2.824295e-06
Finished epoch 13 / 20: loss - 1.799539, train accuracy - 0.385000, validation accuracy - 0.379200, learning rate - 2.541866e-06
Finished epoch 14 / 20: loss - 1.725795, train accuracy - 0.359000, validation accuracy - 0.384800, learning rate - 2.287679e-06
Finished epoch 15 / 20: loss - 1.602119, train accuracy - 0.424000, validation accuracy - 0.386200, learning rate - 2.058911e-06
Finished epoch 16 / 20: loss - 1.706854, train accuracy - 0.431000, validation accuracy - 0.390200, learning rate - 1.853020e-06
Finished epoch 17 / 20: loss - 1.599278, train accuracy - 0.428000, validation accuracy - 0.392600, learning rate - 1.667718e-06
Finished epoch 18 / 20: loss - 1.588828, train accuracy - 0.396000, validation accuracy - 0.396600, learning rate - 1.500946e-06
Finished epoch 19 / 20: loss - 1.648386, train accuracy - 0.406000, validation accuracy - 0.398200, learning rate - 1.350852e-06
Finished epoch 20 / 20: loss - 1.726955, train accuracy - 0.402000, validation accuracy - 0.398400, learning rate - 1.215767e-06
finished optimization. best validation accuracy: 0.398400
```



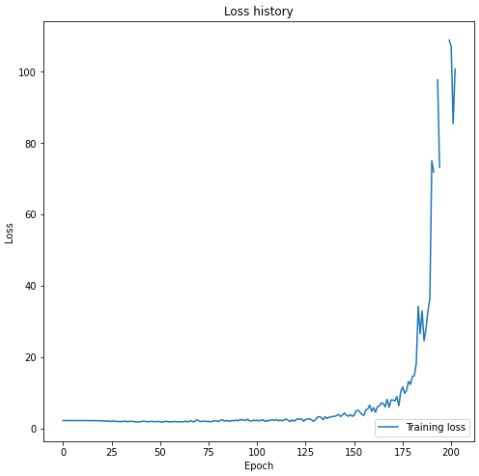


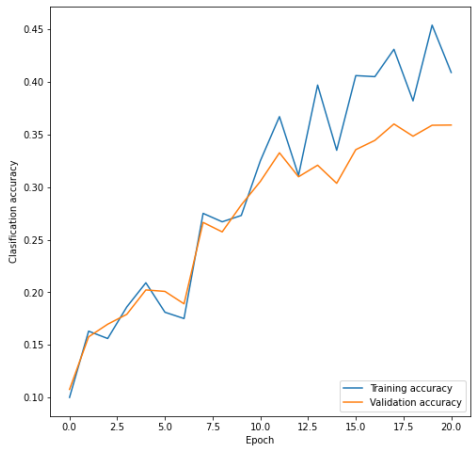

```
Test accuracy:  0.4068
```

Now we change it to 0.001

```
classifier = TwoLayerNeuralNet(32*32*3, 150, 10)  # initialize the neural net
best_model, loss_history, train_history, val_history = \
    classifier.train(x_train, y_train, x_val, y_val, reg=0.0001, lr=1e-3, momentum=0.9, lr_decay=0.9,
                     method='momentum', mini_batch_SGD=True, num_epoch=20, batch_size=128)
```

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.100000, validation accuracy - 0.107600, learning rate - 1.000000e-03
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:99: RuntimeWarning: divide by zero encountered in log
Finished epoch 1 / 20: loss - inf, train accuracy - 0.163000, validation accuracy - 0.157600, learning rate - 9.000000e-04
Finished epoch 2 / 20: loss - inf, train accuracy - 0.156000, validation accuracy - 0.169600, learning rate - 8.100000e-04
Finished epoch 3 / 20: loss - inf, train accuracy - 0.186000, validation accuracy - 0.179000, learning rate - 7.290000e-04
Finished epoch 4 / 20: loss - inf, train accuracy - 0.209000, validation accuracy - 0.202200, learning rate - 6.561000e-04
Finished epoch 5 / 20: loss - inf, train accuracy - 0.181000, validation accuracy - 0.200800, learning rate - 5.904900e-04
Finished epoch 6 / 20: loss - inf, train accuracy - 0.175000, validation accuracy - 0.189000, learning rate - 5.314410e-04
Finished epoch 7 / 20: loss - inf, train accuracy - 0.275000, validation accuracy - 0.266400, learning rate - 4.782969e-04
Finished epoch 8 / 20: loss - inf, train accuracy - 0.267000, validation accuracy - 0.257400, learning rate - 4.304672e-04
Finished epoch 9 / 20: loss - inf, train accuracy - 0.273000, validation accuracy - 0.283000, learning rate - 3.874205e-04
Finished epoch 10 / 20: loss - inf, train accuracy - 0.325000, validation accuracy - 0.305400, learning rate - 3.486784e-04
Finished epoch 11 / 20: loss - inf, train accuracy - 0.367000, validation accuracy - 0.332600, learning rate - 3.138106e-04
Finished epoch 12 / 20: loss - inf, train accuracy - 0.311000, validation accuracy - 0.309800, learning rate - 2.824295e-04
Finished epoch 13 / 20: loss - inf, train accuracy - 0.397000, validation accuracy - 0.320800, learning rate - 2.541866e-04
Finished epoch 14 / 20: loss - inf, train accuracy - 0.335000, validation accuracy - 0.303600, learning rate - 2.287679e-04
Finished epoch 15 / 20: loss - inf, train accuracy - 0.406000, validation accuracy - 0.335600, learning rate - 2.058911e-04
Finished epoch 16 / 20: loss - inf, train accuracy - 0.405000, validation accuracy - 0.344400, learning rate - 1.853020e-04
Finished epoch 17 / 20: loss - inf, train accuracy - 0.431000, validation accuracy - 0.360000, learning rate - 1.667718e-04
Finished epoch 18 / 20: loss - inf, train accuracy - 0.382000, validation accuracy - 0.348400, learning rate - 1.500946e-04
Finished epoch 19 / 20: loss - inf, train accuracy - 0.454000, validation accuracy - 0.358800, learning rate - 1.350852e-04
Finished epoch 20 / 20: loss - inf, train accuracy - 0.409000, validation accuracy - 0.359000, learning rate - 1.215767e-04
finished optimization. best validation accuracy: 0.360000
```






```
Test accuracy:  0.3624
```

In this part we changed the learning rate to 0.00001 and 0.001 to compare the results.

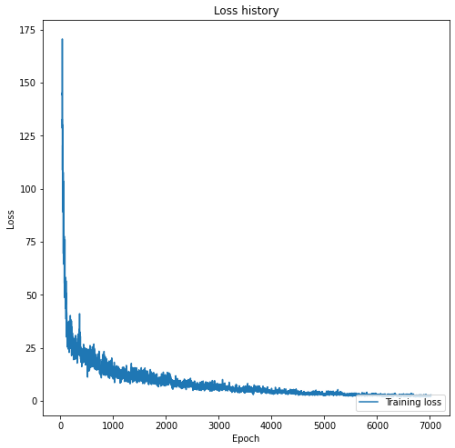
When the learning rate was 0.00001 which is one tenth of our base model, we can see the model did not reached a converge, so we can say our model is underfitted. Model reached the accuracy of 40% but if the epochs were bigger it could have reached a better accuracy because of low learning rate our update steps are lower and model needs more time to converge to a minimum.

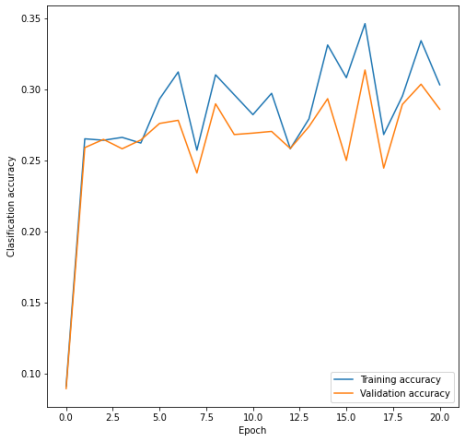
As we set the learning rate to 0.001 which was ten times bigger than our base model, we can see our model accuracy is noisy. This is because our learning rate is big so updating steps are abruptly and model could not find an optimal minimum. As we can see its accuracy is even lower than 0.00001 learning rate since it may never reach a convergence.

4. Now we change the initial values of weight to normal distribution with mean 0 and std of 0.1

```
self.model = {}
# self.model['W1'] = 0.00001 * np.random.randn(input_size, hidden_size)   # small random values
self.model['W1'] = numpy.random.normal(loc=0.0, scale=0.1, size=(input_size, hidden_size))   # small random values
self.model['b1'] = np.zeros(hidden_size)                                                     # zeros

# self.model['W2'] = 0.00001 * np.random.randn(hidden_size, output_size)   # small random values
self.model['W2'] = numpy.random.normal(loc=0.0, scale=0.1, size=(hidden_size, output_size))   # small random values
self.model['b2'] = np.zeros(output_size)                                                     # zeros
```

```
Training data shape:  (45000, 3072)       Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)       Train labels shape:  (5000,)
Test data shape:       (10000, 3072)       Test labels shape:  (10000,)

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:99: RuntimeWarning: divide by zero encountered in log
Finished epoch 0 / 20: loss - inf, train accuracy - 0.091000, validation accuracy - 0.089400, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 32.817018, train accuracy - 0.265000, validation accuracy - 0.258800, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 15.140894, train accuracy - 0.264000, validation accuracy - 0.264600, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 14.447478, train accuracy - 0.266000, validation accuracy - 0.258000, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 11.859141, train accuracy - 0.262000, validation accuracy - 0.264200, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 11.924516, train accuracy - 0.293000, validation accuracy - 0.275800, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 7.841896, train accuracy - 0.312000, validation accuracy - 0.278000, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 8.671788, train accuracy - 0.257000, validation accuracy - 0.241000, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 5.306695, train accuracy - 0.310000, validation accuracy - 0.289600, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 6.913266, train accuracy - 0.296000, validation accuracy - 0.268000, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 7.027223, train accuracy - 0.282000, validation accuracy - 0.269000, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 4.246932, train accuracy - 0.297000, validation accuracy - 0.270200, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 4.586328, train accuracy - 0.258000, validation accuracy - 0.258200, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 3.554937, train accuracy - 0.279000, validation accuracy - 0.273600, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 3.953671, train accuracy - 0.331000, validation accuracy - 0.293200, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 3.304609, train accuracy - 0.308000, validation accuracy - 0.249800, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 3.315563, train accuracy - 0.346000, validation accuracy - 0.313400, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 2.711556, train accuracy - 0.268000, validation accuracy - 0.244400, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 2.773810, train accuracy - 0.295000, validation accuracy - 0.289200, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 2.452017, train accuracy - 0.334000, validation accuracy - 0.303400, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 2.755745, train accuracy - 0.303000, validation accuracy - 0.285800, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.313400
```



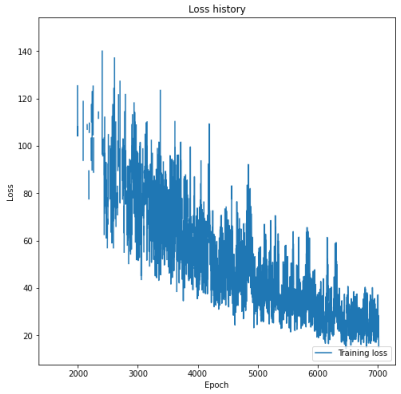


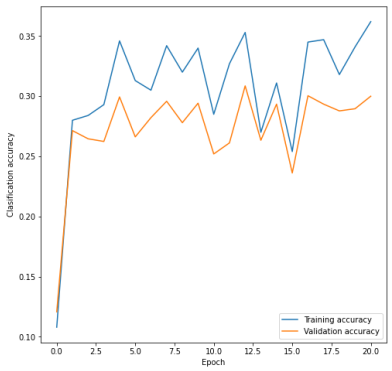

```
Test accuracy:  0.2884
```

Now we set the initial weights to a uniform distribution between 0 and 1

```
self.model = {}
# self.model['W1'] = 0.00001 * np.random.randn(input_size, hidden_size)   # small random values
self.model['W1'] = numpy.random.uniform(low=0.0, high=1.0, size=(input_size, hidden_size))   # small random values
self.model['b1'] = np.zeros(hidden_size)
                                # zeros
# self.model['W2'] = 0.00001 * np.random.randn(hidden_size, output_size)  # small random values
self.model['W2'] = numpy.random.uniform(low=0.0, high=1.0, size=(hidden_size, output_size))   # small random values
self.model['b2'] = np.zeros(output_size)                                  # zeros
```

```
Training data shape:  (45000, 3072)      Train labels shape:  (45000,)
Validation data shape:  (5000, 3072)      Train labels shape:  (5000,)
Test data shape:      (10000, 3072)      Test labels shape:  (10000,)

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:99: RuntimeWarning: divide by zero encountered in log
Finished epoch 0 / 20: loss - inf, train accuracy - 0.108000, validation accuracy - 0.120800, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - inf, train accuracy - 0.280000, validation accuracy - 0.271400, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - inf, train accuracy - 0.284000, validation accuracy - 0.264600, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - inf, train accuracy - 0.293000, validation accuracy - 0.262400, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - inf, train accuracy - 0.346000, validation accuracy - 0.299400, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - inf, train accuracy - 0.313000, validation accuracy - 0.266200, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - inf, train accuracy - 0.305000, validation accuracy - 0.282200, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 74.732631, train accuracy - 0.342000, validation accuracy - 0.295800, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 82.539765, train accuracy - 0.320000, validation accuracy - 0.278000, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - inf, train accuracy - 0.340000, validation accuracy - 0.294200, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 86.193150, train accuracy - 0.285000, validation accuracy - 0.252000, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 67.142814, train accuracy - 0.327000, validation accuracy - 0.261200, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 60.704099, train accuracy - 0.353000, validation accuracy - 0.308600, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - inf, train accuracy - 0.270000, validation accuracy - 0.263400, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 54.176239, train accuracy - 0.311000, validation accuracy - 0.293400, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 33.078339, train accuracy - 0.254000, validation accuracy - 0.236200, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 32.247010, train accuracy - 0.345000, validation accuracy - 0.300400, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 38.997964, train accuracy - 0.347000, validation accuracy - 0.293400, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 33.264214, train accuracy - 0.318000, validation accuracy - 0.287800, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 31.616191, train accuracy - 0.341000, validation accuracy - 0.289600, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 15.065149, train accuracy - 0.362000, validation accuracy - 0.300000, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.308600
```






```
Test accuracy:  0.3014
```

Here we change the initial value of weights in our model as we can compare the results.

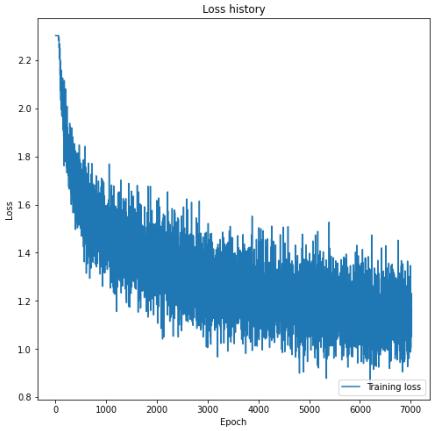
As it can be seen in both cases accuracies are noisy and model could not find an optimal answer for the final weights. This is because initial weights might impact the updating steps since all the weights are almost the same model will find it difficult to find a value for each weight individually and steps could be noisy. Each weight needs to be updated in respect to other weights in their next layer but as we see they are all the same so it is impossible for model to find out which weight had more impact on the answer it finds and as our model update the weights more and more answers will appeared. So, it was impossible for the model to reach a value for weights and did not converge at all. Even with bigger epochs it is not necessarily reach a convergence although after a big number of epochs model may find out the correct way but it is kind of randomly.

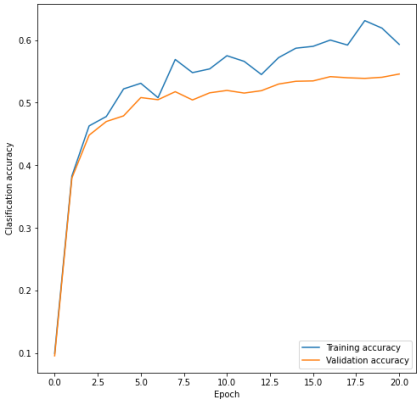
Question 2 Extra

1. Here we are going to decompose our input vector from 32*32*3 to 128 vectors then run the base model for it:

```
from sklearn.decomposition import PCA
# PCA deComposition
pca = PCA(n_components=128)
x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)
```

```
Training data shape:  (45000, 128)      Train labels shape:  (45000,)
Validation data shape:  (5000, 128)      Train labels shape:  (5000,)
Test data shape:      (10000, 128)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302586, train accuracy - 0.100000, validation accuracy - 0.096000, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 1.676397, train accuracy - 0.383000, validation accuracy - 0.379800, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 1.392072, train accuracy - 0.463000, validation accuracy - 0.448200, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 1.454464, train accuracy - 0.478000, validation accuracy - 0.470000, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 1.416620, train accuracy - 0.522000, validation accuracy - 0.479000, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 1.343343, train accuracy - 0.531000, validation accuracy - 0.508200, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 1.377145, train accuracy - 0.508000, validation accuracy - 0.504800, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 1.264922, train accuracy - 0.569000, validation accuracy - 0.517600, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 1.205311, train accuracy - 0.548000, validation accuracy - 0.504400, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 1.290314, train accuracy - 0.554000, validation accuracy - 0.515800, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 1.121025, train accuracy - 0.575000, validation accuracy - 0.519600, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 1.297327, train accuracy - 0.566000, validation accuracy - 0.515400, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 1.056863, train accuracy - 0.545000, validation accuracy - 0.519200, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 1.219253, train accuracy - 0.572000, validation accuracy - 0.529800, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 1.225816, train accuracy - 0.587000, validation accuracy - 0.534200, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 1.051106, train accuracy - 0.590000, validation accuracy - 0.534800, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 1.165837, train accuracy - 0.600000, validation accuracy - 0.541600, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 1.242460, train accuracy - 0.592000, validation accuracy - 0.539800, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 1.029183, train accuracy - 0.631000, validation accuracy - 0.538800, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 1.063203, train accuracy - 0.619000, validation accuracy - 0.540600, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 1.167677, train accuracy - 0.593000, validation accuracy - 0.545800, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.545800
```



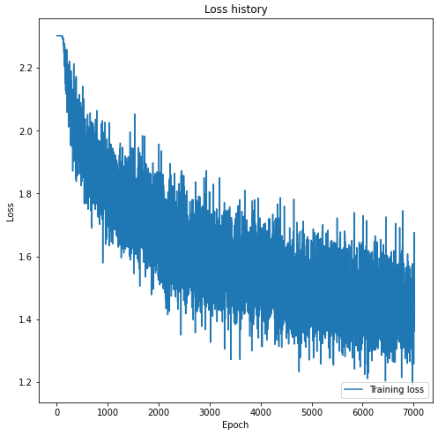


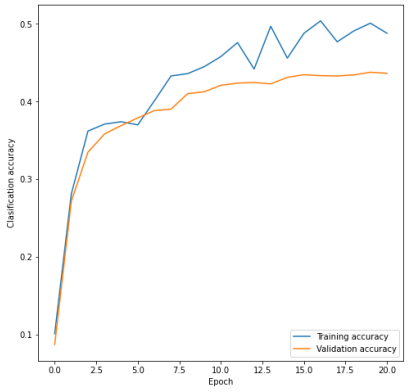

```
Test accuracy:  0.5198
```

2. Here we convert the images to gray scale and then train the model on them.

```python
#  Convert to Gray Scale
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

# print(x_train)
x_train = rgb2gray(x_train)
x_test = rgb2gray(x_test)
```

```
Training data shape:  (45000, 1024)      Train labels shape:  (45000,)
Validation data shape:  (5000, 1024)      Train labels shape:  (5000,)
Test data shape:      (10000, 1024)      Test labels shape:  (10000,)

Finished epoch 0 / 20: loss - 2.302585, train accuracy - 0.101000, validation accuracy - 0.087000, learning rate - 1.000000e-04
Finished epoch 1 / 20: loss - 2.119606, train accuracy - 0.281000, validation accuracy - 0.271600, learning rate - 9.000000e-05
Finished epoch 2 / 20: loss - 1.876034, train accuracy - 0.362000, validation accuracy - 0.334800, learning rate - 8.100000e-05
Finished epoch 3 / 20: loss - 1.783630, train accuracy - 0.371000, validation accuracy - 0.358400, learning rate - 7.290000e-05
Finished epoch 4 / 20: loss - 1.812134, train accuracy - 0.374000, validation accuracy - 0.369000, learning rate - 6.561000e-05
Finished epoch 5 / 20: loss - 1.769240, train accuracy - 0.370000, validation accuracy - 0.379000, learning rate - 5.904900e-05
Finished epoch 6 / 20: loss - 1.646556, train accuracy - 0.401000, validation accuracy - 0.388400, learning rate - 5.314410e-05
Finished epoch 7 / 20: loss - 1.646091, train accuracy - 0.433000, validation accuracy - 0.390200, learning rate - 4.782969e-05
Finished epoch 8 / 20: loss - 1.611570, train accuracy - 0.436000, validation accuracy - 0.410200, learning rate - 4.304672e-05
Finished epoch 9 / 20: loss - 1.668770, train accuracy - 0.445000, validation accuracy - 0.412600, learning rate - 3.874205e-05
Finished epoch 10 / 20: loss - 1.506161, train accuracy - 0.458000, validation accuracy - 0.421000, learning rate - 3.486784e-05
Finished epoch 11 / 20: loss - 1.484169, train accuracy - 0.476000, validation accuracy - 0.423800, learning rate - 3.138106e-05
Finished epoch 12 / 20: loss - 1.544057, train accuracy - 0.442000, validation accuracy - 0.424600, learning rate - 2.824295e-05
Finished epoch 13 / 20: loss - 1.444879, train accuracy - 0.497000, validation accuracy - 0.422800, learning rate - 2.541866e-05
Finished epoch 14 / 20: loss - 1.585065, train accuracy - 0.456000, validation accuracy - 0.431200, learning rate - 2.287679e-05
Finished epoch 15 / 20: loss - 1.374619, train accuracy - 0.488000, validation accuracy - 0.434600, learning rate - 2.058911e-05
Finished epoch 16 / 20: loss - 1.383290, train accuracy - 0.504000, validation accuracy - 0.433400, learning rate - 1.853020e-05
Finished epoch 17 / 20: loss - 1.582738, train accuracy - 0.477000, validation accuracy - 0.433000, learning rate - 1.667718e-05
Finished epoch 18 / 20: loss - 1.463670, train accuracy - 0.491000, validation accuracy - 0.434400, learning rate - 1.500946e-05
Finished epoch 19 / 20: loss - 1.553094, train accuracy - 0.501000, validation accuracy - 0.437800, learning rate - 1.350852e-05
Finished epoch 20 / 20: loss - 1.414408, train accuracy - 0.488000, validation accuracy - 0.436400, learning rate - 1.215767e-05
finished optimization. best validation accuracy: 0.437800
```






```
Test accuracy:  0.4187
```

Here in this part, first we use PCA to decrease our feature vector from 32*32*3 to 128. As the plots shows it reached 52% accuracy which is a great accuracy among all the parts and besides our model trained very fast since our input was very small and it could calculate all the weights and updating them fast. And because 128 components of PCA could almost include about 90% variance of the whole image so it is a great number and lessen the overfitting in our model since our input vector is small and it is hard for model to memorize the input. So, by using PCA we had better accuracy, faster training and less overfitting which was great results.

In the second part, we converted our input images to gray scale so instead of having 32*32*3 images we had 32*32 images which is smaller than before. It reached an accuracy of 42% which is not great. Our accuracy was low because when converting from RGB to gray a lot of vital information's will disappear and unlike the PCA part we can't extract all the information for classification from it. Despite the lower accuracy, model trained faster than our base model since input vector was smaller and resulted in less overfitting but still it is not a good idea to convert them to gray scale because of loss of information.

Question 3

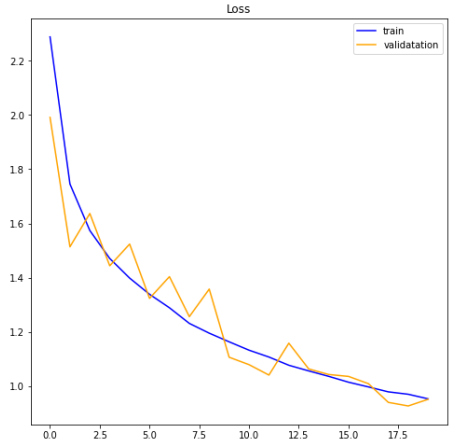
Here we implement a convolutional neural network to classify the cifar10 dataset. Here we used TensorFlow package to do so.

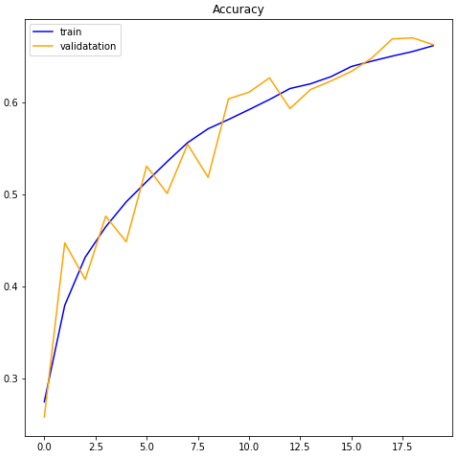
Our define network is shown below:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_14 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_15 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_16 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_17 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_18 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_19 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               262272
_________________________________________________________________
batch_normalization_20 (Batc (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 552,874
Trainable params: 551,722
Non-trainable params: 1,152
```

After training we have:

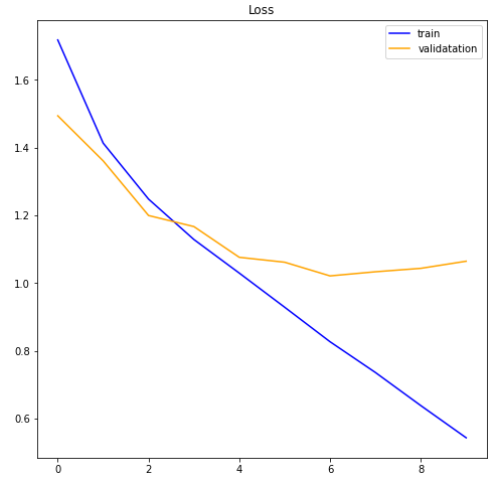
```
352/352 [==============================] - 390s 1s/step - loss: 1.2008 - accuracy: 0.5667 - val_loss: 1.1725 - val_accuracy: 0.5820
Epoch 10/20
352/352 [==============================] - 387s 1s/step - loss: 1.1589 - accuracy: 0.5808 - val_loss: 1.0532 - val_accuracy: 0.6266
Epoch 11/20
352/352 [==============================] - 383s 1s/step - loss: 1.1418 - accuracy: 0.5921 - val_loss: 1.0815 - val_accuracy: 0.6166
Epoch 12/20
352/352 [==============================] - 382s 1s/step - loss: 1.1196 - accuracy: 0.5965 - val_loss: 1.2567 - val_accuracy: 0.5638
Epoch 13/20
352/352 [==============================] - 379s 1s/step - loss: 1.1046 - accuracy: 0.6047 - val_loss: 1.0558 - val_accuracy: 0.6270
Epoch 14/20
352/352 [==============================] - 379s 1s/step - loss: 1.0742 - accuracy: 0.6168 - val_loss: 1.0235 - val_accuracy: 0.6410
Epoch 15/20
352/352 [==============================] - 380s 1s/step - loss: 1.0549 - accuracy: 0.6207 - val_loss: 0.9875 - val_accuracy: 0.6522
Epoch 16/20
352/352 [==============================] - 378s 1s/step - loss: 1.0464 - accuracy: 0.6253 - val_loss: 0.9447 - val_accuracy: 0.6648
Epoch 17/20
352/352 [==============================] - 382s 1s/step - loss: 1.0132 - accuracy: 0.6378 - val_loss: 0.9373 - val_accuracy: 0.6640
Epoch 18/20
352/352 [==============================] - 380s 1s/step - loss: 1.0103 - accuracy: 0.6393 - val_loss: 1.1132 - val_accuracy: 0.6036
Epoch 19/20
352/352 [==============================] - 382s 1s/step - loss: 0.9895 - accuracy: 0.6475 - val_loss: 0.8944 - val_accuracy: 0.6806
Epoch 20/20
352/352 [==============================] - 383s 1s/step - loss: 0.9766 - accuracy: 0.6523 - val_loss: 0.9421 - val_accuracy: 0.6532
313/313 [==============================] - 20s 64ms/step - loss: 0.9757 - accuracy: 0.6501
```

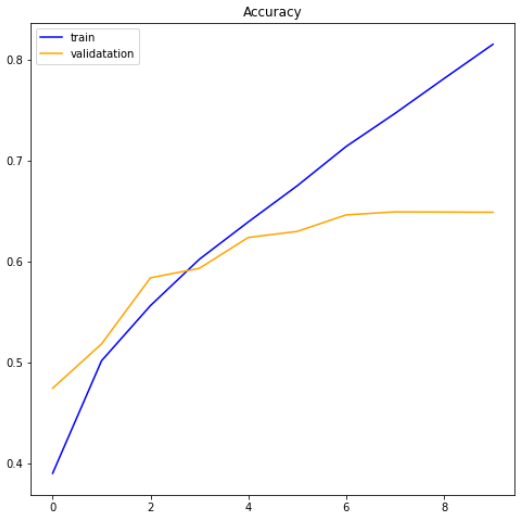





```
Test Accuracy is :  64.910
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_30 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 32, 32, 64)        36928
_________________________________________________________________
flatten_5 (Flatten)          (None, 65536)             0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               8388736
_________________________________________________________________
dense_11 (Dense)             (None, 10)                1290
=================================================================
Total params: 8,455,594
Trainable params: 8,455,594
Non-trainable params: 0
```






```
Epoch 9/10
352/352 [==============================] - 673s 2s/step - loss: 0.6273 - accuracy: 0.7871 - val_loss: 1.0436 - val_accuracy: 0.6492
Epoch 10/10
352/352 [==============================] - 659s 2s/step - loss: 0.5222 - accuracy: 0.8261 - val_loss: 1.0647 - val_accuracy: 0.6490
313/313 [==============================] - 32s 102ms/step - loss: 1.1299 - accuracy: 0.6294
Test Accuracy is :  62.940
```

Because learning our conv layer without any pooling and normalization took very long to train, I only trained it for 10 epochs. Still, we can see that without max pooling, network got overfitted to the training data since training accuracy is diverge from the validation accuracy. Despite the low difference between two networks, we can see the one with Pooling layers did not overfitted and can be generalized well but the second one, overfitted a lot and it is not a good model for generalization.

Another effect of max pooling is that learning without pooling take much more time since a lot of more calculation is needed. Max Pooling help to decrease the input feature vector for the next layer, it only takes the most valuable information from the previous image, this helps to decrease the mathematics complexity.

Here we can see the output of first and second convolutional layers for the first image.

As it can be seen in the output images, we can see only some of the features from the input vector are calculated and by multiplying it to the weight vector, a feature map image will be resulted. After each conv layer more and more features are extracted until at the end we get a flatten feature map including only some vital information to detect the image.