Department of Mechanical and Mechatronics Engineering

ECE 457B – Assignment #2

A Report Prepared For:

Prof. Fakhri Karray

ECE 457B

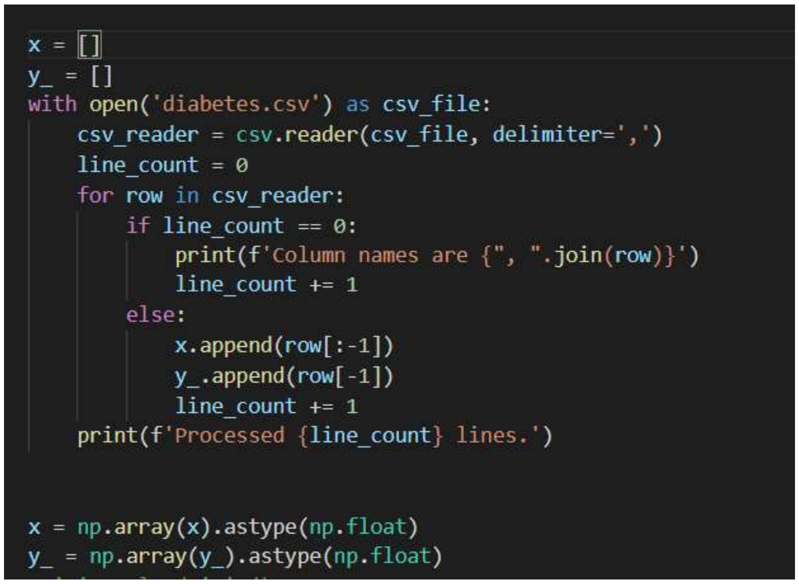
Fundamentals of Computational Intelligence

A Report Prepared By:

Kapilan Satkunanathan (20694418)

Table of Contents

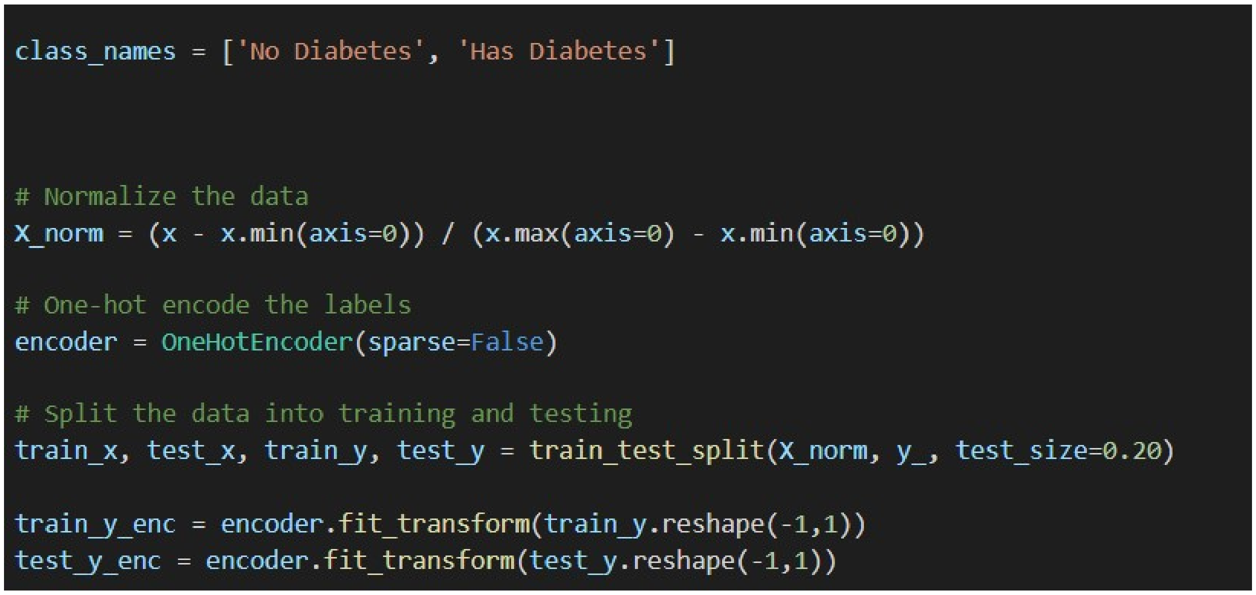
Question 1



```
x = []
y_ = []
with open('diabetes.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            x.append(row[:-1])
            y_.append(row[-1])
            line_count += 1
    print(f'Processed {line_count} lines.')


x = np.array(x).astype(np.float)
y_ = np.array(y_).astype(np.float)
```

Figure 1 loading data

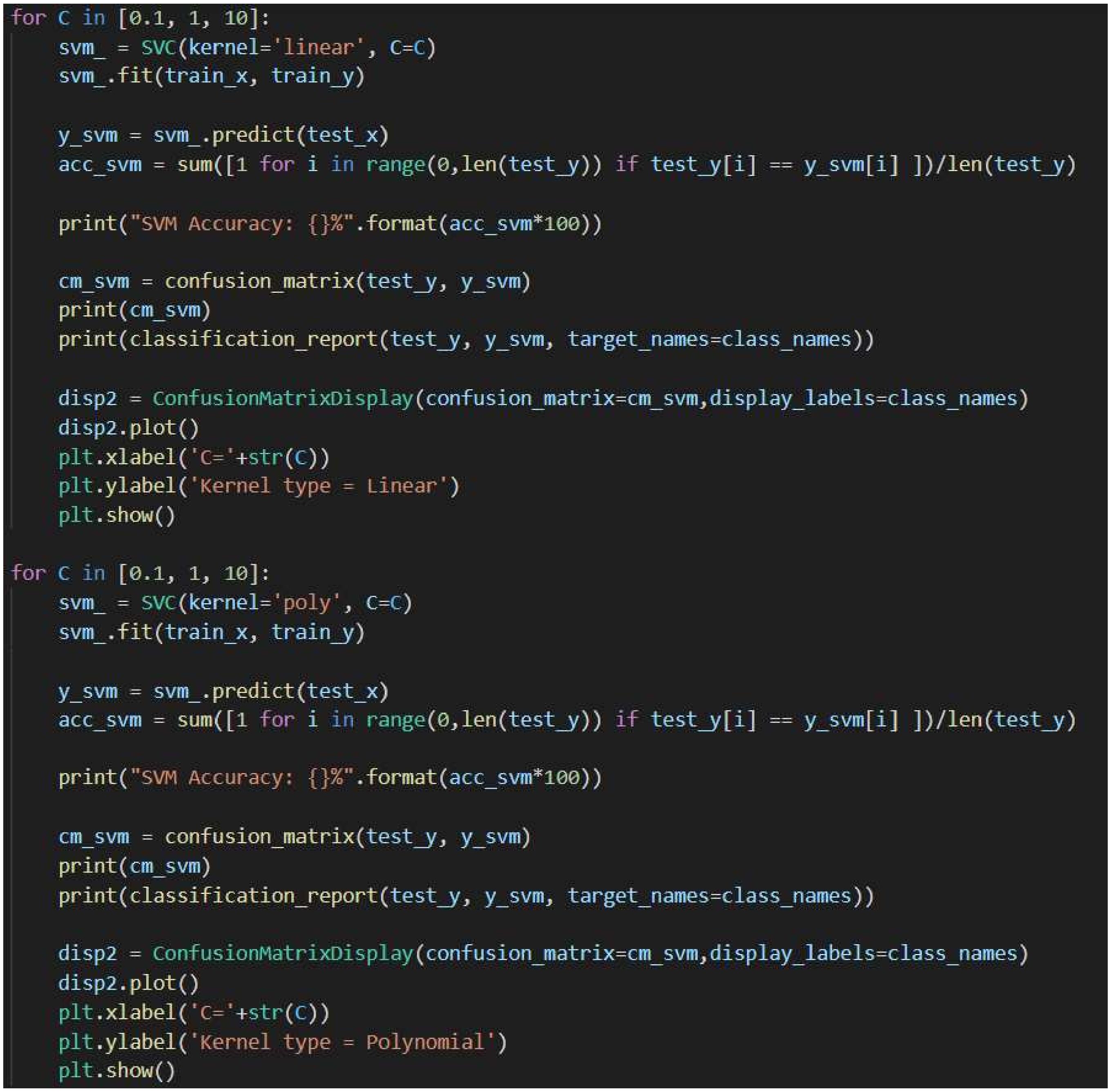
Figure 1 demonstrates the loading of the data onto the SVM, it essentially utilizes a csv reader to parse the data. The result of whether the patient has diabetes or not is stored into the y_ array. The rest of the parameters of that patient case is stored in the x array.



```
class_names = ['No Diabetes', 'Has Diabetes']



# Normalize the data
X_norm = (x - x.min(axis=0)) / (x.max(axis=0) - x.min(axis=0))

# One-hot encode the labels
encoder = OneHotEncoder(sparse=False)

# Split the data into training and testing
train_x, test_x, train_y, test_y = train_test_split(X_norm, y_, test_size=0.20)

train_y_enc = encoder.fit_transform(train_y.reshape(-1,1))
test_y_enc = encoder.fit_transform(test_y.reshape(-1,1))
```

Figure 2 Train/Test split and Class labels

That data is then normalized as shown in Figure 2, also 20% of the original data set is set aside to be used as test data to validate our model and observe performance specs such as accuracy, precision, recall and F1 score. Note, the class labels are designated as ‘No Diabetes’ (patient is negative) and ‘Has Diabetes’ (patient is positive).



```
for C in [0.1, 1, 10]:
    svm_ = SVC(kernel='linear', C=C)
    svm_.fit(train_x, train_y)

    y_svm = svm_.predict(test_x)
    acc_svm = sum([1 for i in range(0,len(test_y)) if test_y[i] == y_svm[i] ])/len(test_y)

    print("SVM Accuracy: {}%".format(acc_svm*100))

    cm_svm = confusion_matrix(test_y, y_svm)
    print(cm_svm)
    print(classification_report(test_y, y_svm, target_names=class_names))

    disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_svm,display_labels=class_names)
    disp2.plot()
    plt.xlabel('C='+str(C))
    plt.ylabel('Kernel type = Linear')
    plt.show()

for C in [0.1, 1, 10]:
    svm_ = SVC(kernel='poly', C=C)
    svm_.fit(train_x, train_y)

    y_svm = svm_.predict(test_x)
    acc_svm = sum([1 for i in range(0,len(test_y)) if test_y[i] == y_svm[i] ])/len(test_y)

    print("SVM Accuracy: {}%".format(acc_svm*100))

    cm_svm = confusion_matrix(test_y, y_svm)
    print(cm_svm)
    print(classification_report(test_y, y_svm, target_names=class_names))

    disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_svm,display_labels=class_names)
    disp2.plot()
    plt.xlabel('C='+str(C))
    plt.ylabel('Kernel type = Polynomial')
    plt.show()
```

Figure 3 Create and Run SVM

Thanks to the library module of sklearn, both the SVM's with a linear and polynomial kernel can be created easily as shown in Figure 3. It's also easily modifiable the C value for both these SVM models. C represents the error cost parameter that addresses misclassification issue, where an increase in C leads to less tolerance the SVM has for misclassification. This increase in C also leads to a less generalizing SVM model.

Results

The following figures are the results yielded from the code as seen in Appendix I. Note, this is for the test data set where it is 20% of the entire data set which has a size of 768 where 20% of that is 154 cases analyzed for each confusion matrix. **Has diabetes is treated as positive and No diabetes is treated as negative**.

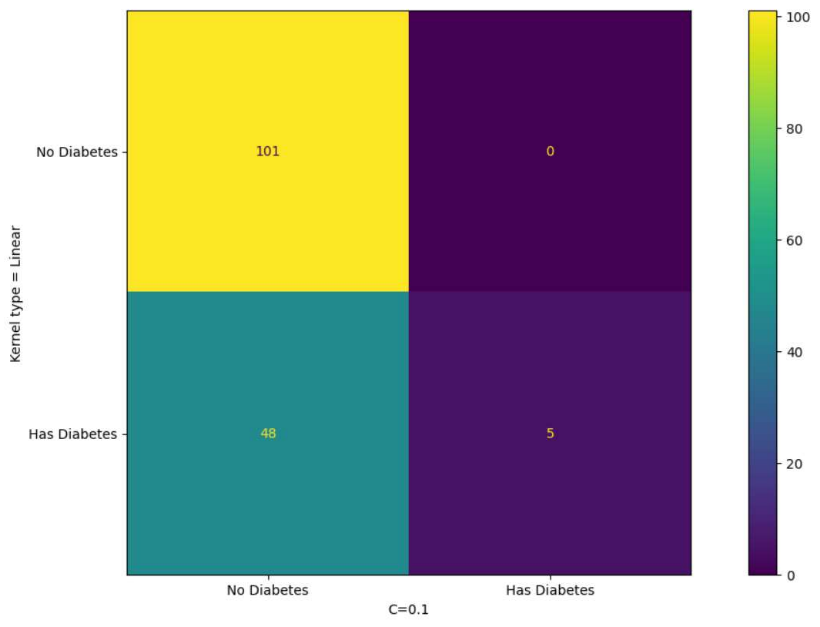


Figure 4 Confusion Matrix Kernel Linear C = 0.1

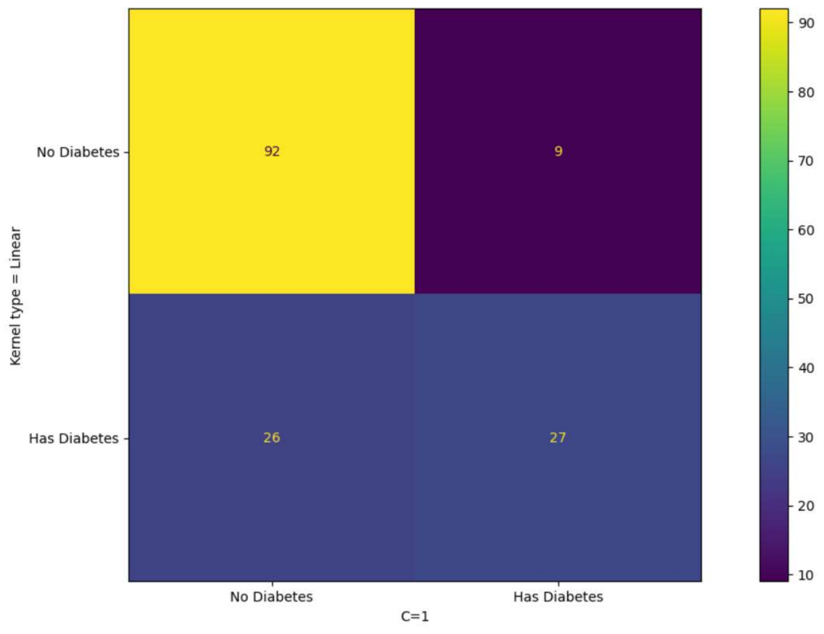


Figure 5 Confusion Matrix Kernel Linear C=1

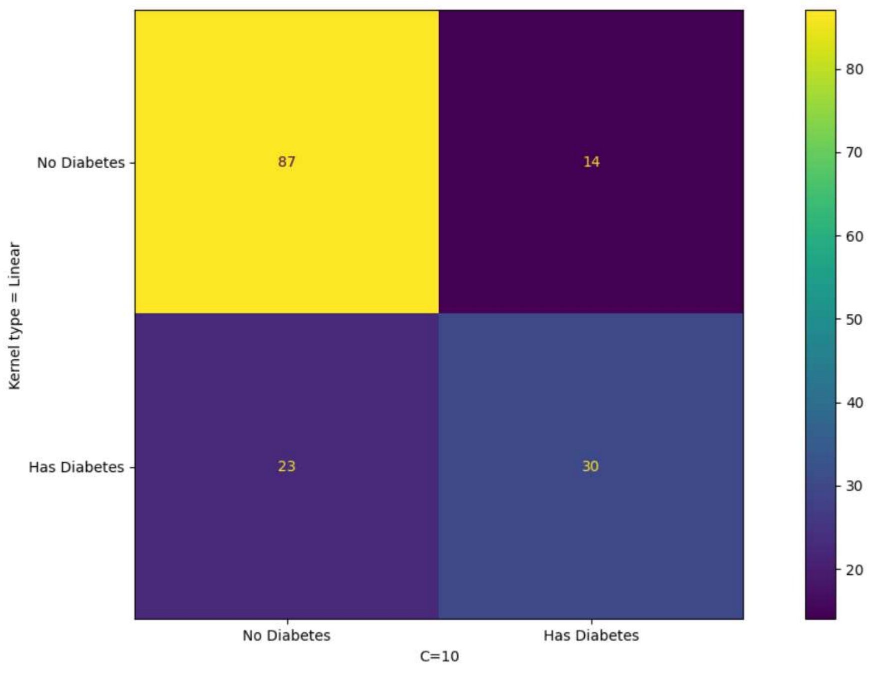


Figure 6 Confusion Matrix Kernel Linear C=10

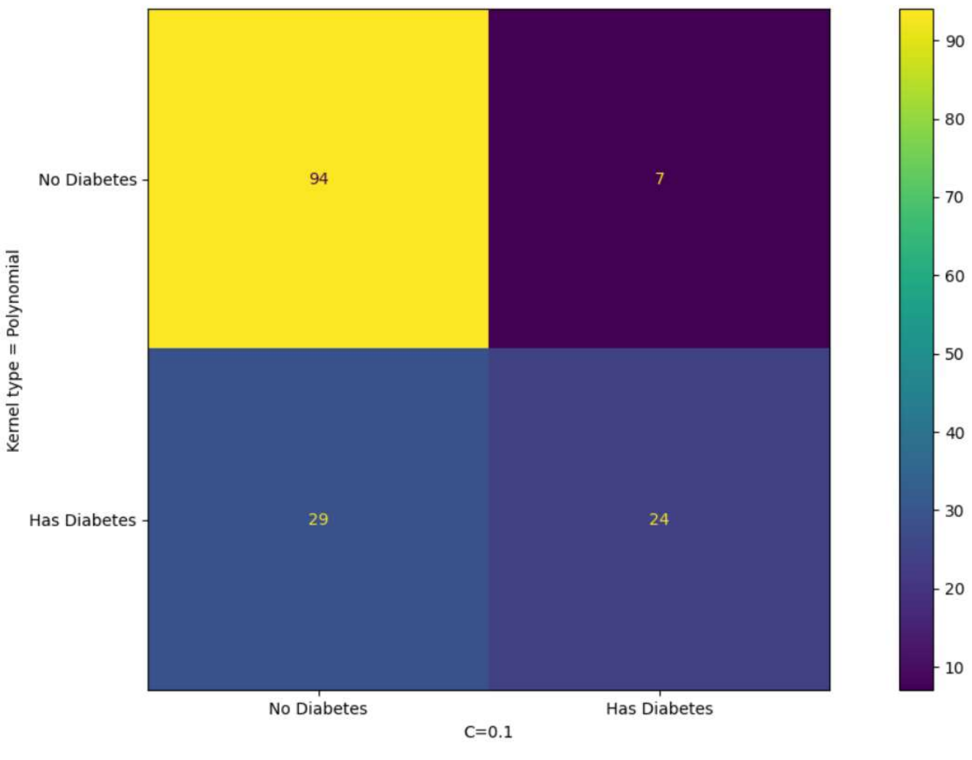


Figure 7 Confusion Matrix Kernel Polynomial C=0.1

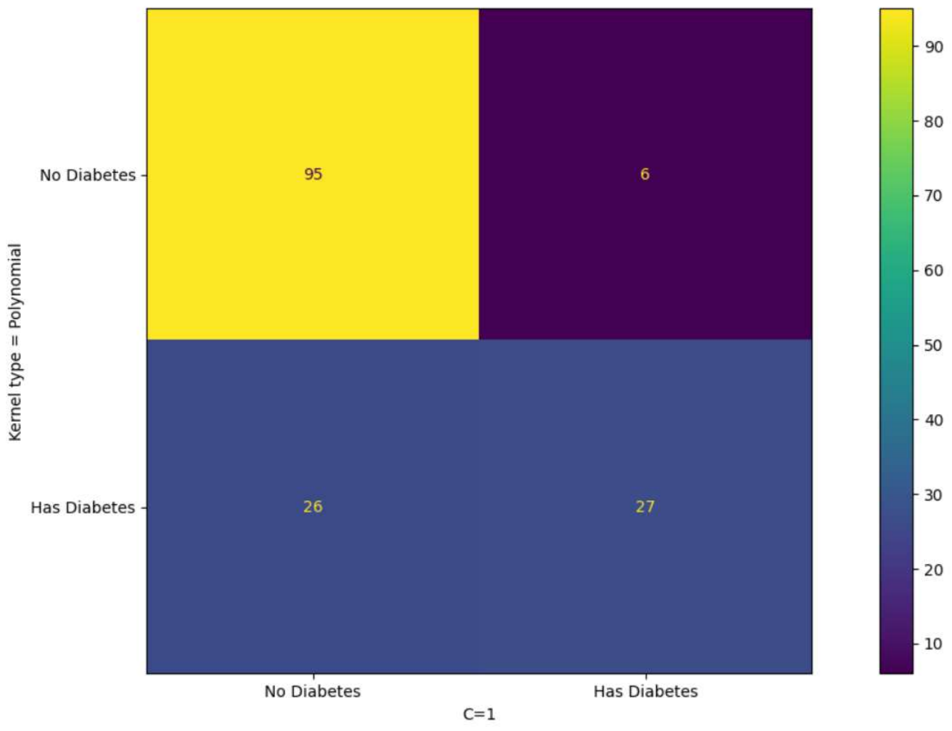


Figure 8 Confusion Matrix Kernel Polynomial C=1

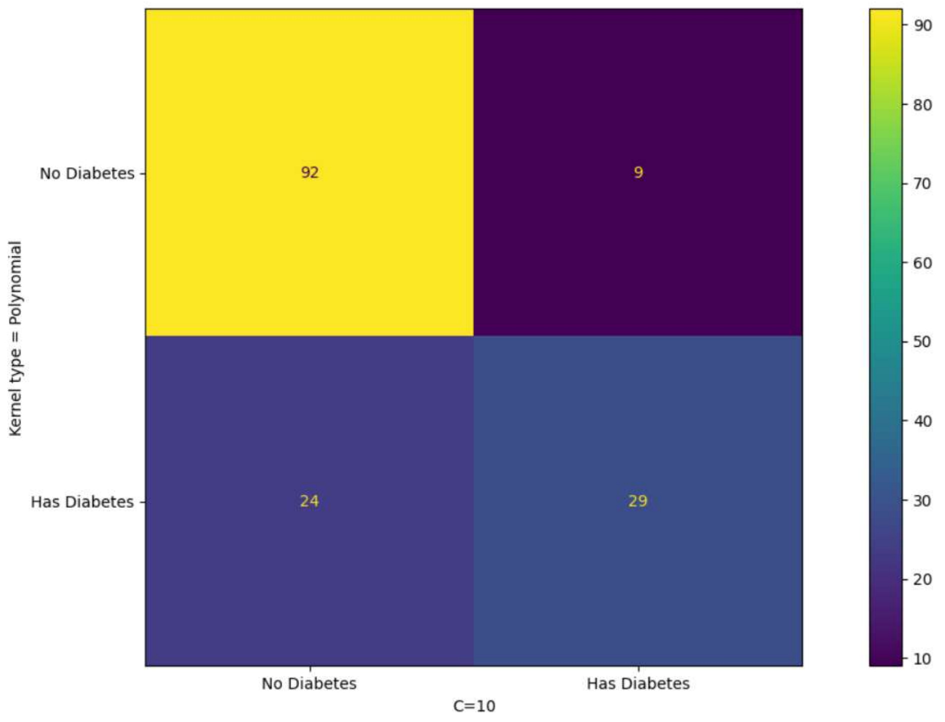


Figure 9 Confusion Matrix Polynomial Kernel C=10

Table 1 performance stats from confusion matrices

Kernel Type	C Value	Accuracy (%)	Precision (%)	Recall (%)	F1 Score
Linear	.1	68.83	9.43	100	0.17
Linear	1	77.27	51	75	0.61
Linear	10	75.97	56.6	68.18	0.62
Polynomial	.1	76.62	45.28	77.41	0.57
Polynomial	1	79.22	50.94	81.81	0.63
Polynomial	10	78.57	54.71	76.32	0.63

Analysis

A trend to observe from Table 1, is that as C increases, the Precision % goes up , but Recall % goes down. The mathematical formulas for all four performance indicators of accuracy, precision, recall and F1 are seen below.

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + True\ Negative + False\ Negative}$$

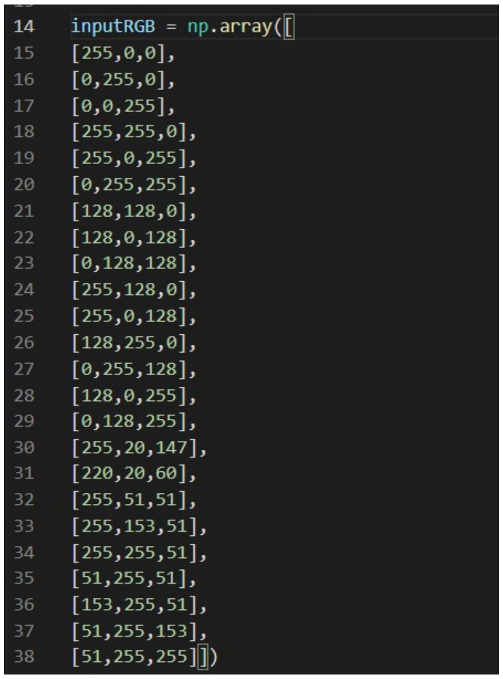
$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

For this case, recall is the most important factor to consider as it’s the ratio of true cases that the model got right. The recall performance measure should be tuned to be as high as possible because if the SVM model misses a true case, then in the real world an individual with diabetes is misdiagnosed. This misdiagnosis could potentially be fatal to the individual’s life if they don’t receive the proper medication.

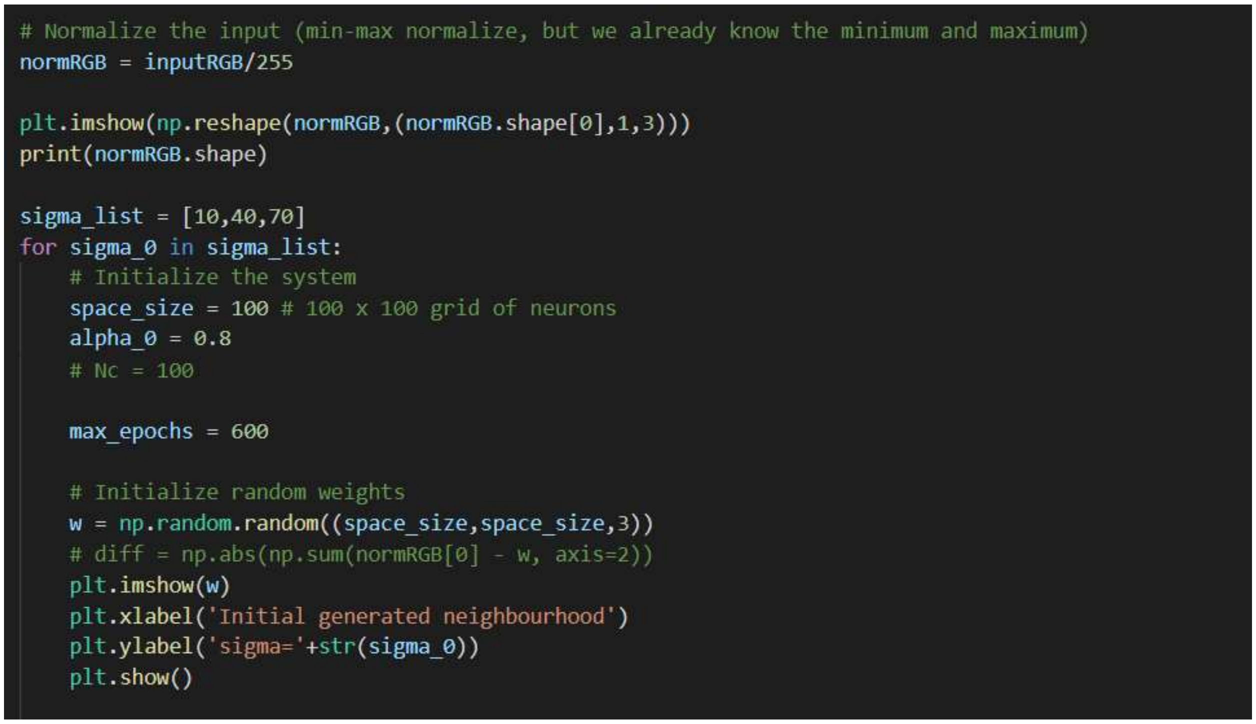
Question 2



```
inputRGB = np.array([
[255,0,0],
[0,255,0],
[0,0,255],
[255,255,0],
[255,0,255],
[0,255,255],
[128,128,0],
[128,0,128],
[0,128,128],
[255,128,0],
[255,0,128],
[128,255,0],
[0,255,128],
[128,0,255],
[0,128,255],
[255,20,147],
[220,20,60],
[255,51,51],
[255,153,51],
[255,255,51],
[51,255,51],
[153,255,51],
[51,255,153],
[51,255,255]])
```

Figure 10 24 colors choices

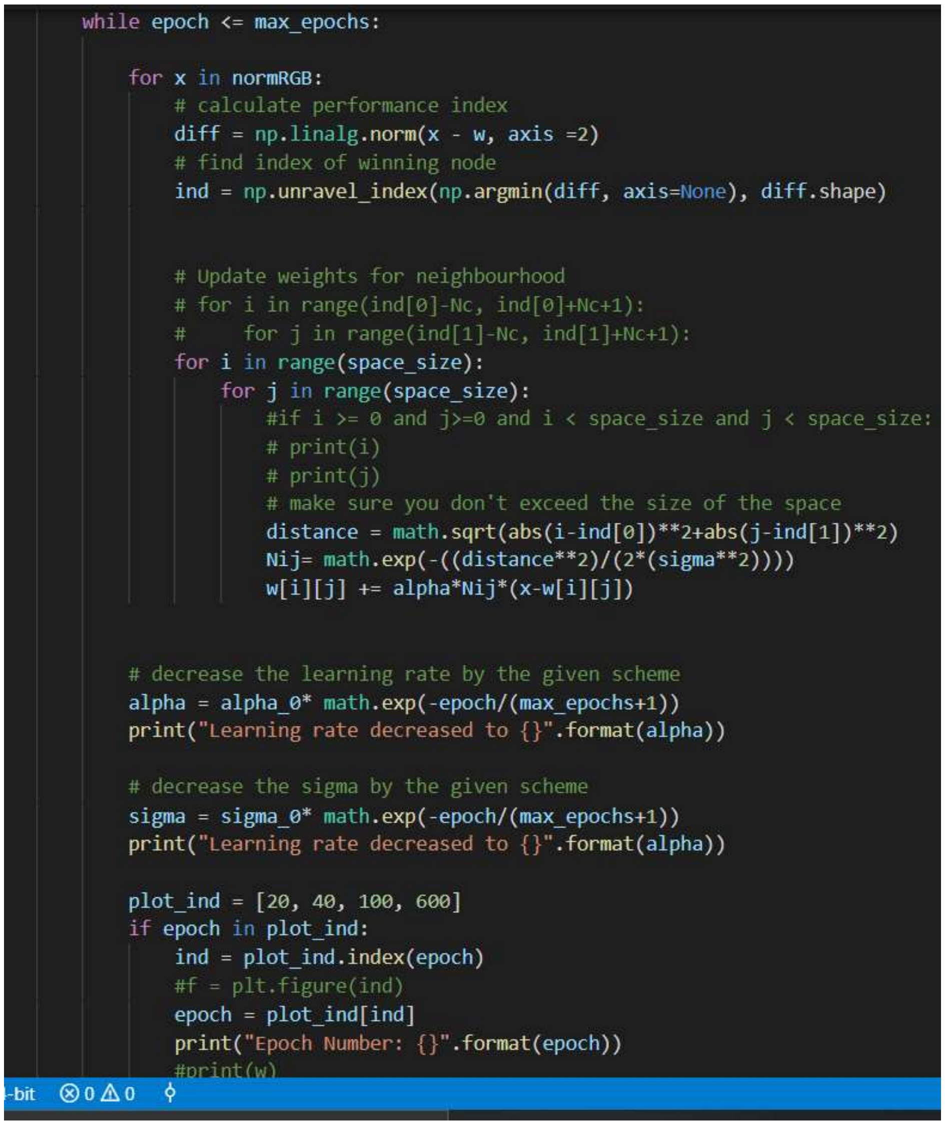
Figure 10 demonstrates the 24 colors that were chosen from rapid tables website that will be used to generate the training input.



```
# Normalize the input (min-max normalize, but we already know the minimum and maximum)
normRGB = inputRGB/255

plt.imshow(np.reshape(normRGB,(normRGB.shape[0],1,3)))
print(normRGB.shape)

sigma_list = [10,40,70]
for sigma_0 in sigma_list:
    # Initialize the system
    space_size = 100 # 100 x 100 grid of neurons
    alpha_0 = 0.8
    # Nc = 100

    max_epochs = 600

    # Initialize random weights
    w = np.random.random((space_size,space_size,3))
    # diff = np.abs(np.sum(normRGB[0] - w, axis=2))
    plt.imshow(w)
    plt.xlabel('Initial generated neighbourhood')
    plt.ylabel('sigma='+str(sigma_0))
    plt.show()
```

Figure 11 initial parameters and generate grid

Figure 11 shows the RGB values getting normalized by dividing 255, which is the standard range for the colors. The initial α is 0.8 and 600 training epochs is set to run. Then a grid of 100 by 100 is randomly generated to represent the randomized output nodes. This simulation is run three times for the parameters where σ is 10, 40 and 70.



```
    while epoch <= max_epochs:

        for x in normRGB:
            # calculate performance index
            diff = np.linalg.norm(x - w, axis =2)
            # find index of winning node
            ind = np.unravel_index(np.argmin(diff, axis=None), diff.shape)


            # Update weights for neighbourhood
            # for i in range(ind[0]-Nc, ind[0]+Nc+1):
            #     for j in range(ind[1]-Nc, ind[1]+Nc+1):
            for i in range(space_size):
                for j in range(space_size):
                    #if i >= 0 and j>=0 and i < space_size and j < space_size:
                    # print(i)
                    # print(j)
                    # make sure you don't exceed the size of the space
                    distance = math.sqrt(abs(i-ind[0])**2+abs(j-ind[1])**2)
                    Nij= math.exp(-((distance**2)/(2*(sigma**2))))
                    w[i][j] += alpha*Nij*(x-w[i][j])


        # decrease the learning rate by the given scheme
        alpha = alpha_0* math.exp(-epoch/(max_epochs+1))
        print("Learning rate decreased to {}".format(alpha))

        # decrease the sigma by the given scheme
        sigma = sigma_0* math.exp(-epoch/(max_epochs+1))
        print("Learning rate decreased to {}".format(alpha))

        plot_ind = [20, 40, 100, 600]
        if epoch in plot_ind:
            ind = plot_ind.index(epoch)
            #f = plt.figure(ind)
            epoch = plot_ind[ind]
            print("Epoch Number: {}".format(epoch))
            #print(w)
```

Figure 12 Weights, learning and sigma update

Figure 12, demonstrates the update rules that were implemented for the weights, alpha and sigma. Where the update rules are as follows:

Weights update:

$$w_{i,j,}^{new} = w_{i,j,}^{old} + \alpha(k)N_{i,j}(k)(x - w_{i,j,}^{old})$$

Where:

$$N_{i,j}(k) = \exp\left(\frac{-d_{i,j}^2}{2\sigma^2(k)}\right)$$

and $d_{i,j}$ is the distance from the node to the winning node.

Alpha update:

$$\alpha(k) = \alpha(0)\exp\left(-\frac{k}{T}\right)$$

Where:

$$\alpha(0) = 0.8$$

Sigma update:

$$\sigma(k) = \sigma_0 \exp\left(-\frac{k}{T}\right)$$

Results

The following figures are the results yielded from the code as seen in Appendix II.

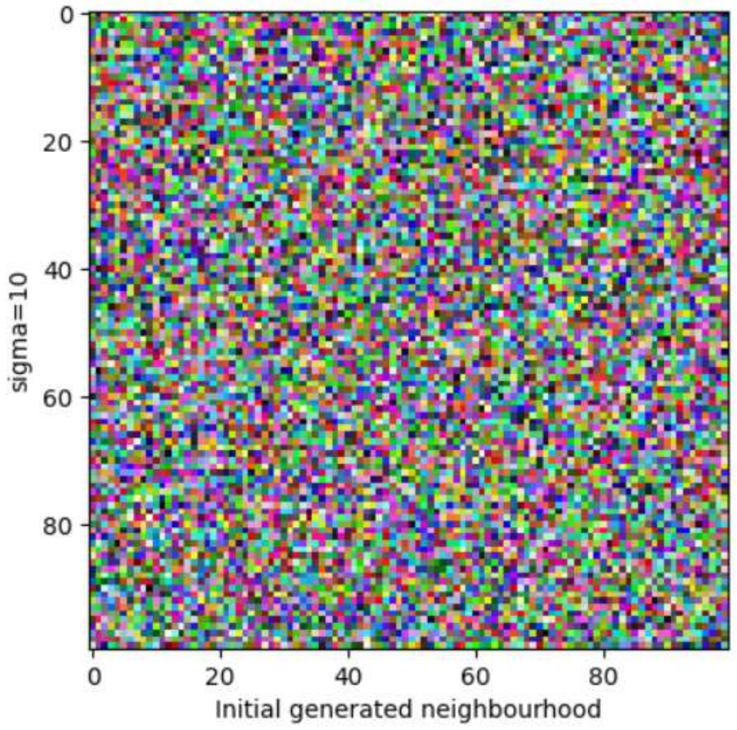


Figure 13 example of initially generated grid

Sigma = 10

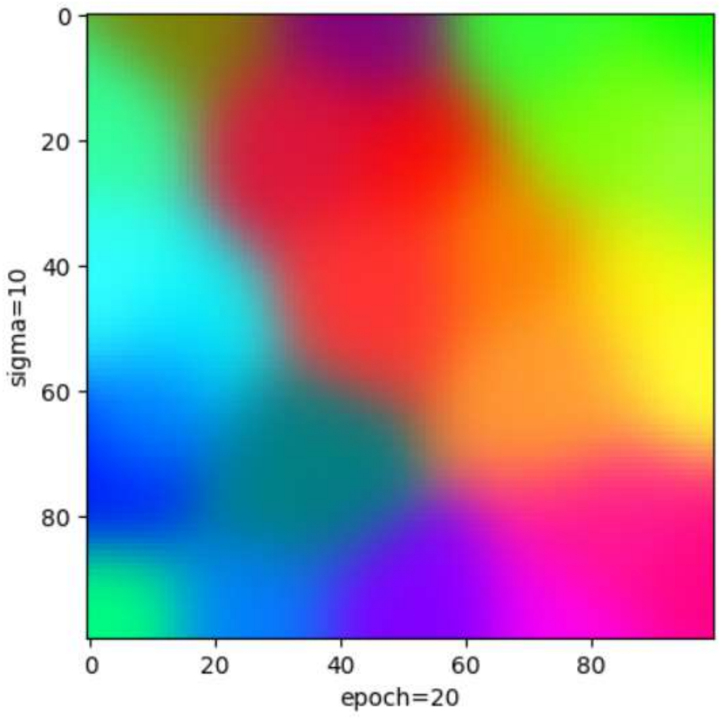


Figure 14 Sigma 10 Epoch 20 results

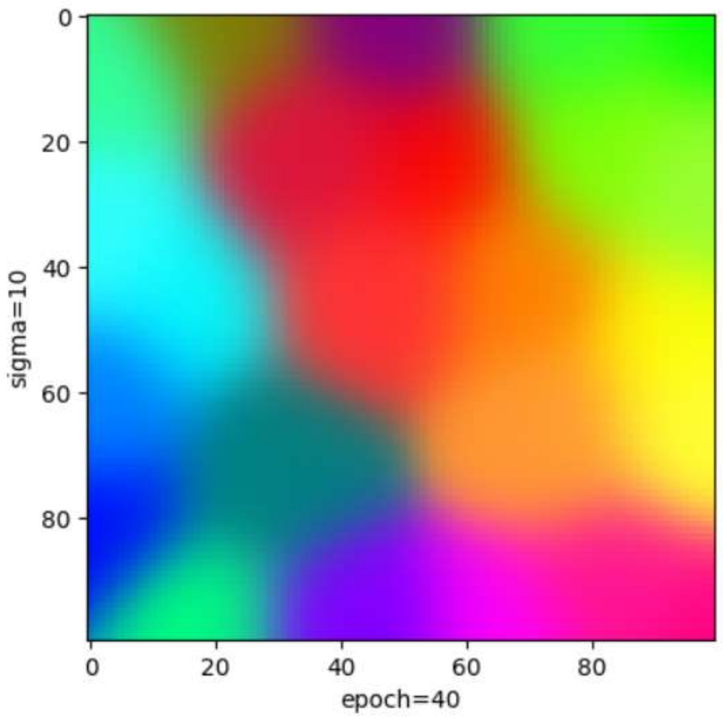


Figure 15 Sigma 10 Epoch 40 results

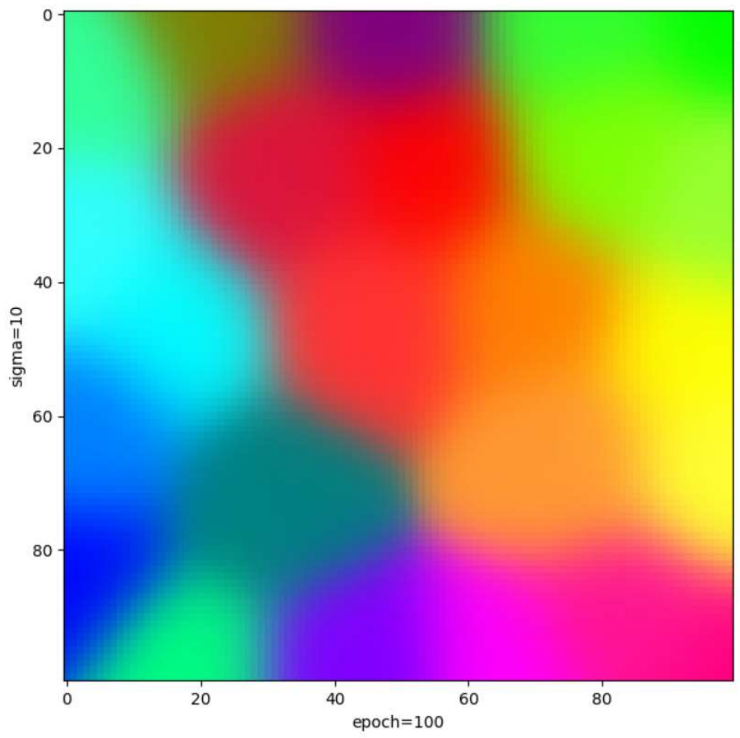


Figure 16 Sigma 10 Epoch 100 results

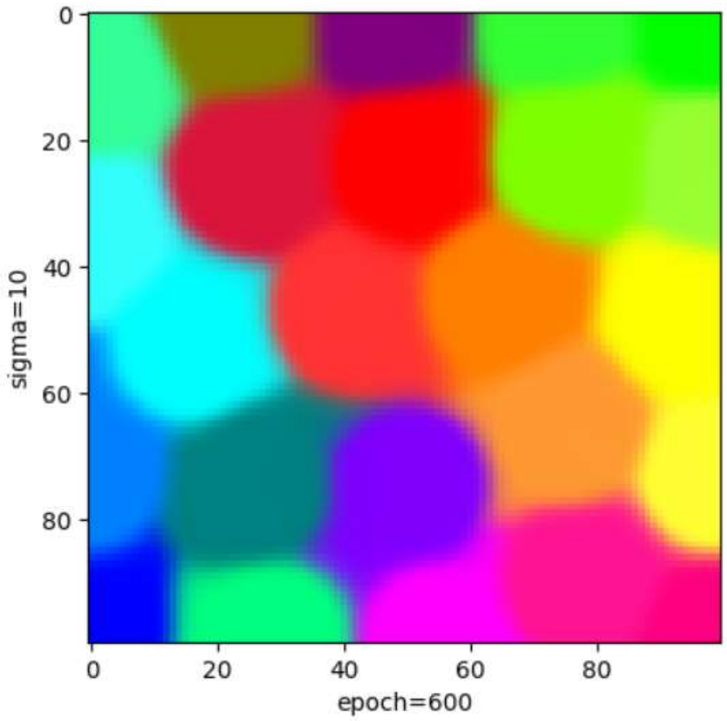


Figure 17 Sigma 10 Epoch 600 results

Sigma = 40

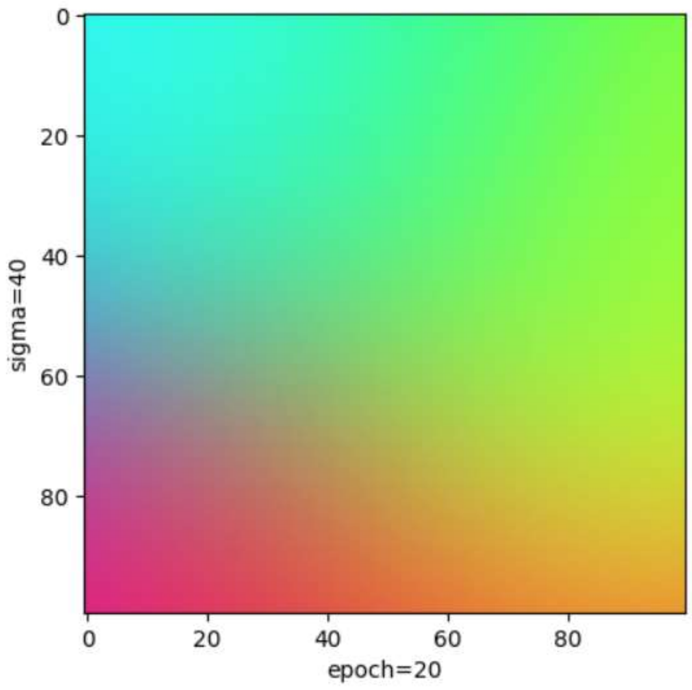


Figure 18 Sigma 40 Epoch 20 results

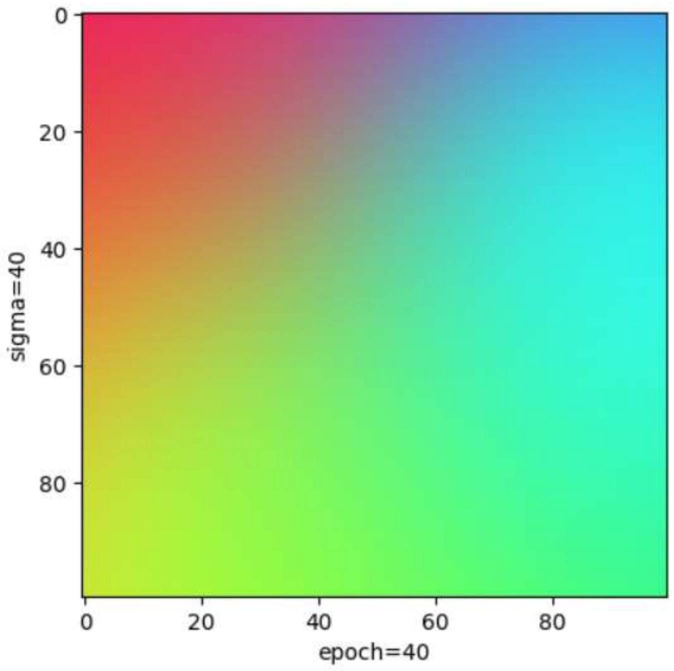


Figure 19 Sigma 40 Epoch 40 results

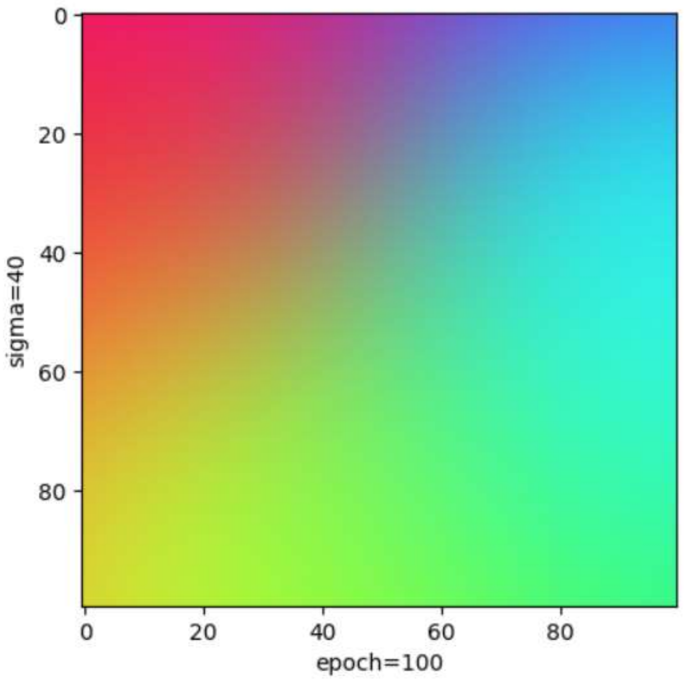


Figure 20 Sigma 40 Epoch 100 results

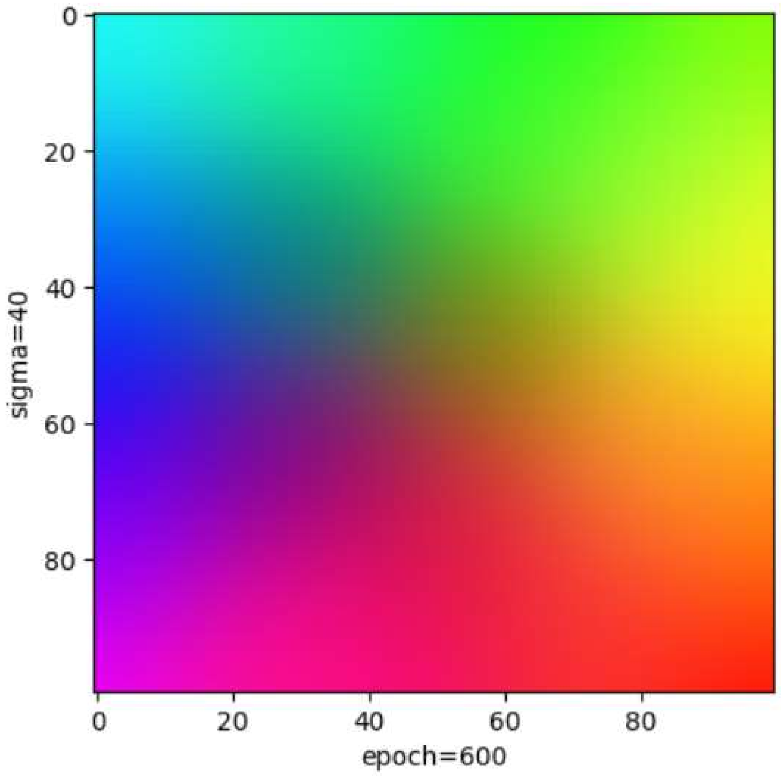


Figure 21 Sigma 40 Epoch 600 results

Sigma = 70

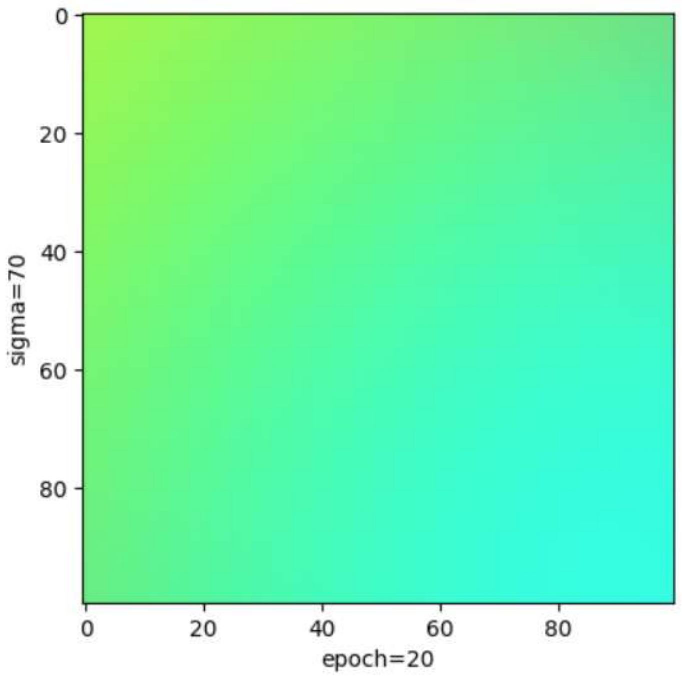


Figure 22 Sigma 70 Epoch 20 results

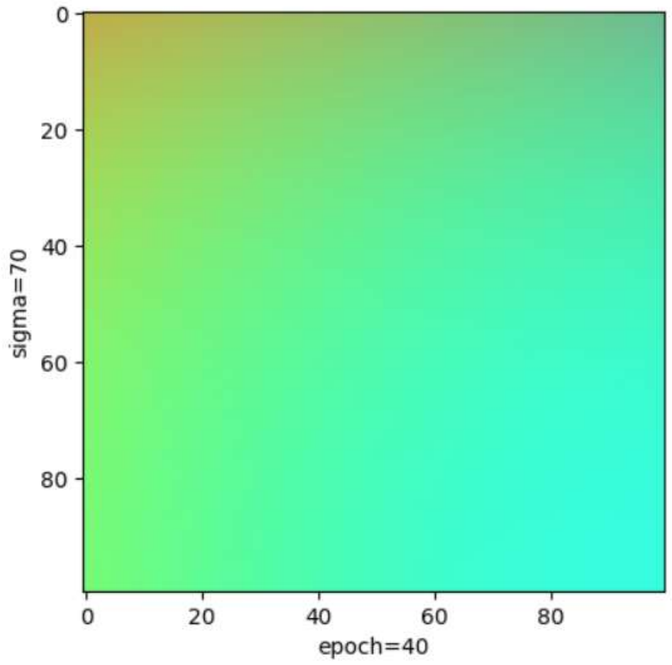


Figure 23 Sigma 70 Epoch 40 results

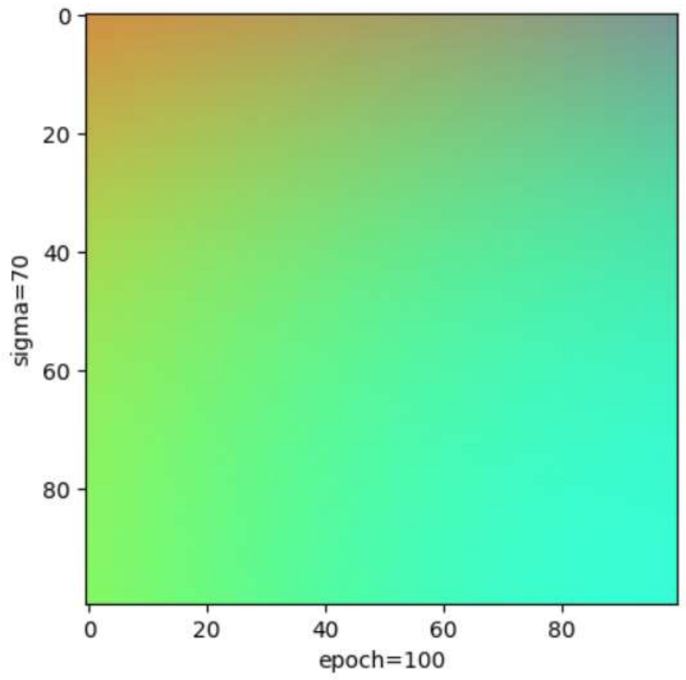


Figure 24 Sigma 70 Epoch 100 results

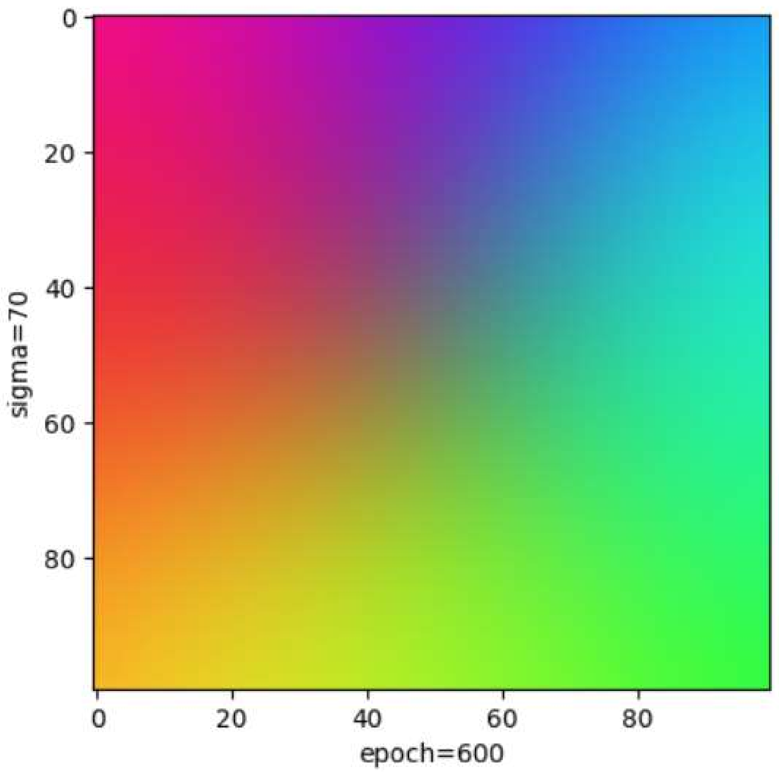


Figure 25 Sigma 70 Epoch 600 results

Analysis

As the sigma increases the variety of colors that shows up decreases, so much so that when the sigma is at 70 only a few color shades are prominent in the grid due to large amounts of clusterization. This can easily be explained as the formula weights update is

$$w_{i,j,}^{new} = w_{i,j,}^{old} + \alpha(k)N_{i,j}(k)(x - w_{i,j,}^{old})$$

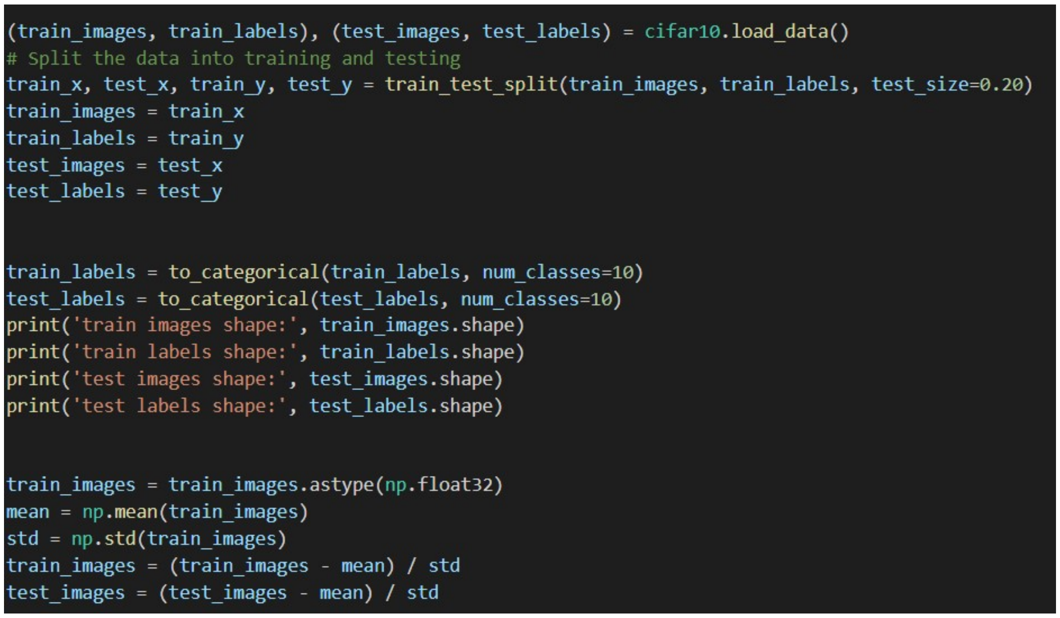
And has $N_{i,j}(k)$ term that when sigma increases the magnitude of the weight change is increased thus causing more clustering to occur thus yielding less shades of color.

$$N_{i,j}(k) = \exp\left(\frac{-d_{i,j}^2}{2\sigma^2(k)}\right)$$

However, even for a large sigma a high number of epochs results in good clusterization.

Question 3

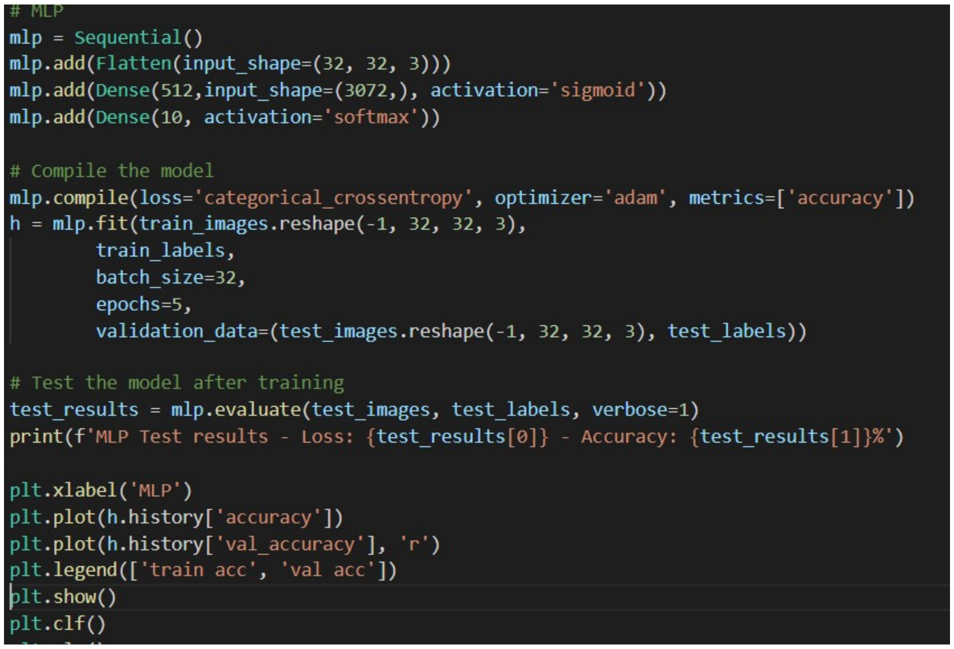
Figure 26, shows the loading of the CIFAR 10 dataset, a point to note is that the original data set is split in 80%/20% training and test data respectively.



```
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
# Split the data into training and testing
train_x, test_x, train_y, test_y = train_test_split(train_images, train_labels, test_size=0.20)
train_images = train_x
train_labels = train_y
test_images = test_x
test_labels = test_y


train_labels = to_categorical(train_labels, num_classes=10)
test_labels = to_categorical(test_labels, num_classes=10)
print('train images shape:', train_images.shape)
print('train labels shape:', train_labels.shape)
print('test images shape:', test_images.shape)
print('test labels shape:', test_labels.shape)


train_images = train_images.astype(np.float32)
mean = np.mean(train_images)
std = np.std(train_images)
train_images = (train_images - mean) / std
test_images = (test_images - mean) / std
```

Figure 26 Loading Cifar 10 dataset

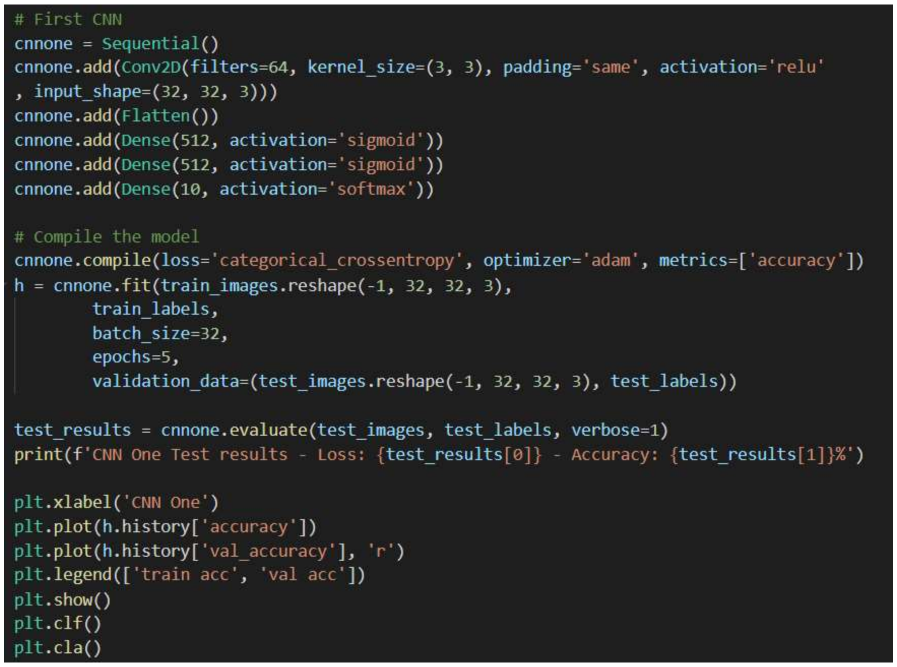
Figure 27 shows the implementation of the MLP with dense layer with 512 units and a sigmoid activation function and a dense layer (output layer) with 10 units (representing 10 classes in the dataset) and a softmax activation function for the classification task



```python
# MLP
mlp = Sequential()
mlp.add(Flatten(input_shape=(32, 32, 3)))
mlp.add(Dense(512,input_shape=(3072,), activation='sigmoid'))
mlp.add(Dense(10, activation='softmax'))

# Compile the model
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
h = mlp.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

# Test the model after training
test_results = mlp.evaluate(test_images, test_labels, verbose=1)
print(f'MLP Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%')

plt.xlabel('MLP')
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
plt.show()
plt.clf()
```

Figure 27 MLP

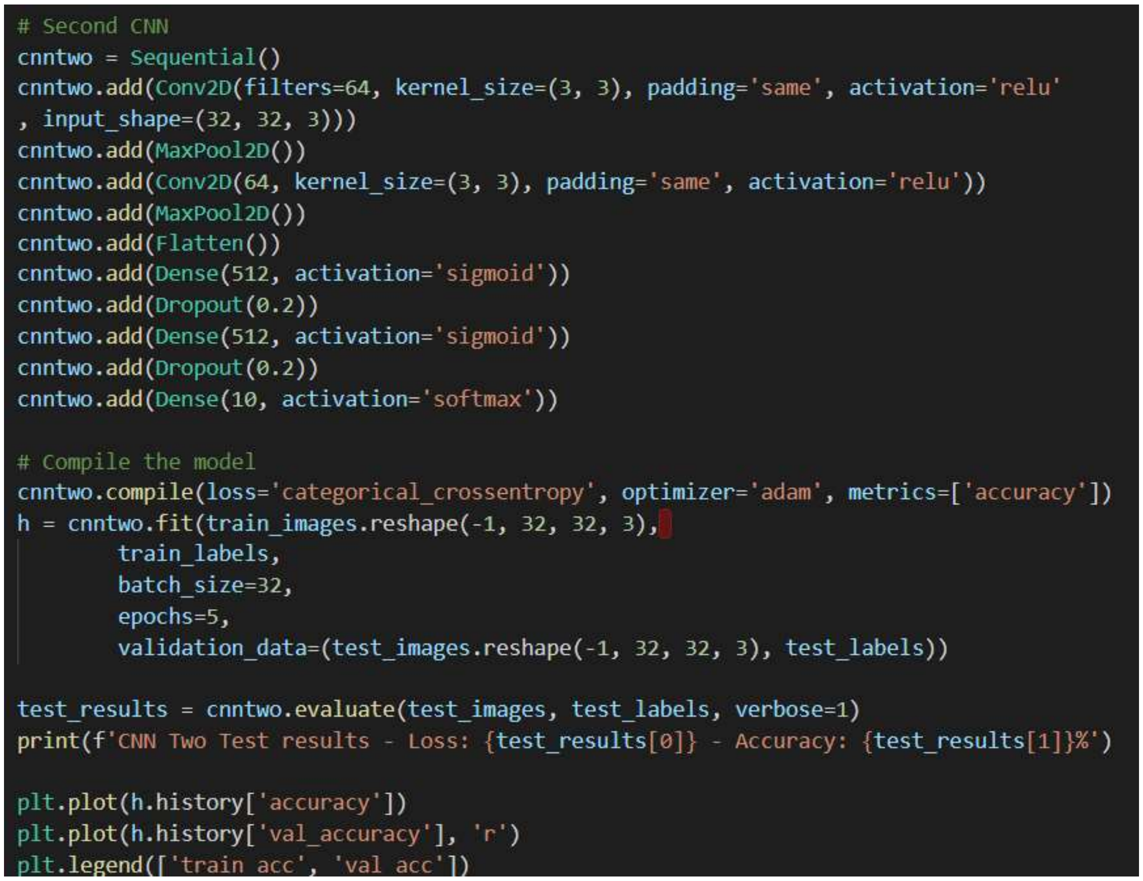
Figure 28 shows the implementation of the first CNN network consisting of 2D Convolutional layer with 64 filters (size of 3x3) and ReLU activation function, flatten layer, Fully connected Dense layer with 512 units and a sigmoid activation function, Fully connected layer with 512 units and a sigmoid activation function and a Dense layer (output layer) with 10 units and a softmax activation function for the classification task



```python
# First CNN
cnnone = Sequential()
cnnone.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'
, input_shape=(32, 32, 3)))
cnnone.add(Flatten())
cnnone.add(Dense(512, activation='sigmoid'))
cnnone.add(Dense(512, activation='sigmoid'))
cnnone.add(Dense(10, activation='softmax'))

# Compile the model
cnnone.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
h = cnnone.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

test_results = cnnone.evaluate(test_images, test_labels, verbose=1)
print(f'CNN One Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%')

plt.xlabel('CNN One')
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
plt.show()
plt.clf()
plt.cla()
```

Figure 28 First CNN Network

Figure 29 shows the implementation of the second CNN network consisting of a 2D Convolutional layer with 64 filters (size of 3x3) and ReLU activation function, a 2x2 Maxpooling layer, a 2D Convolutional layer with 64 filters (size of 3x3) and ReLU activation function, a 2x2 Maxpooling layer , a flatten layer, fully connected dense layer with 512 units and a sigmoid activation function, dropout layer with 0.2 dropout rate, fully connected layer with 512 units and a sigmoid activation function, dropout layer with 0.2 dropout rate, dense layer (output layer) with 10 units and a softmax activation function for the classification task



```
# Second CNN
cnntwo = Sequential()
cnntwo.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'
, input_shape=(32, 32, 3)))
cnntwo.add(MaxPool2D())
cnntwo.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
cnntwo.add(MaxPool2D())
cnntwo.add(Flatten())
cnntwo.add(Dense(512, activation='sigmoid'))
cnntwo.add(Dropout(0.2))
cnntwo.add(Dense(512, activation='sigmoid'))
cnntwo.add(Dropout(0.2))
cnntwo.add(Dense(10, activation='softmax'))

# Compile the model
cnntwo.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
h = cnntwo.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

test_results = cnntwo.evaluate(test_images, test_labels, verbose=1)
print(f'CNN Two Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%')

plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
```

Figure 29 Second CNN Network

Lastly all networks used a batch size of 32 and Adam as the optimizer and a categorical cross entropy loss function for 5 epochs.

Results

The following figures are the results yielded from the code as seen in Appendix III.

Table 2 Training vs Testing Accuracy

Network	Training Accuracy (%)	Testing Accuracy (%)
MLP	48	42
CNN 1	80	62
CNN 2	85	71

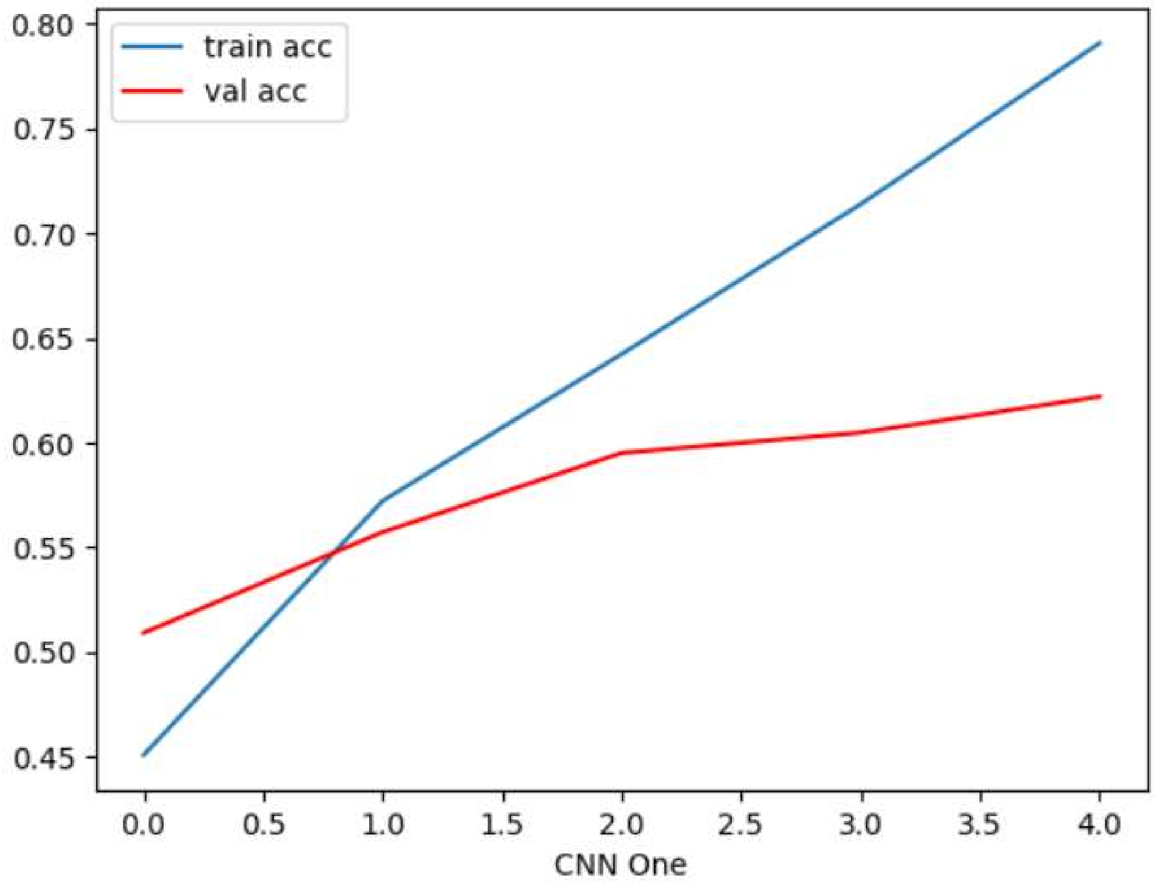


Figure 30 Training vs Validation Accuracy Plot CNN One

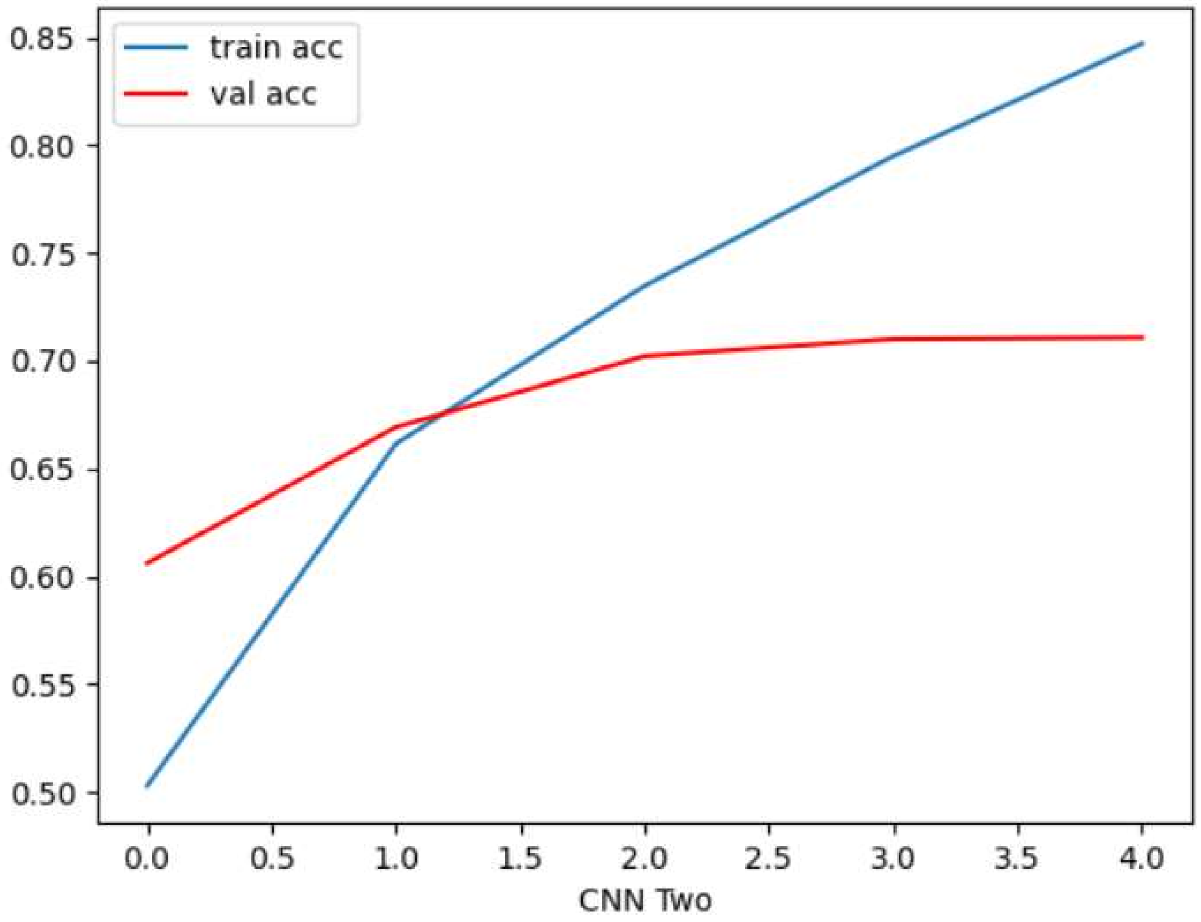


Figure 31 Training vs Validation Accuracy Plot CNN Two

Analysis

In terms of run time, it's quite clear that the more complexity there is within the network, the longer the run time is for the network. MLP had the shortest run time, but the lowest training and validation accuracy. Whereas the complex CNN one and two networks had a very long run time, but a high testing and validation accuracy. If more epochs were to be added, I think the accuracies will increase, but the rate at which the accuracies of the network increase would be slower than the graphs seen in Figure 30 and Figure 31. It's very clear also that as complexity increases the rate at which the accuracies increase is quite large as seen in the two above figures. In terms of architecture, CNN two has drop out layers to prevent overfitting which could play a role into why it has higher accuracy as well.

Appendix

Appendix I

```
# ECE 457b
# Kapilan Satkunanathan
# 20694418
# Assignment 2 Question 1

import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMat
rixDisplay


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

import csv
import matplotlib.pyplot as plt

x = []
y_ = []
with open('diabetes.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            x.append(row[:-1])
            y_.append(row[-1])
            line_count += 1
    print(f'Processed {line_count} lines.')


x = np.array(x).astype(np.float)
y_ = np.array(y_).astype(np.float)
# iris = load_iris()
# x = iris.data
# y_ = iris.target

print("Size of features: {}".format(x.shape))
print("Size of labels: {}".format(y_.shape))

print("Sample data: {}".format(x[:3]))
print("Sample labels: {}".format(y_))

class_names = ['No Diabetes', 'Has Diabetes']



# Normalize the data
X_norm = (x - x.min(axis=0)) / (x.max(axis=0) - x.min(axis=0))

# One-hot encode the labels
encoder = OneHotEncoder(sparse=False)

# Split the data into training and testing
train_x, test_x, train_y, test_y = train_test_split(X_norm, y_, test_size=0.20)

train_y_enc = encoder.fit_transform(train_y.reshape(-1,1))
test_y_enc = encoder.fit_transform(test_y.reshape(-1,1))

print("Sample train data: {}".format(train_x[:3]))
print("Sample train labels: {}".format(train_y_enc[:3]))

for C in [0.1, 1, 10]:
    svm_ = SVC(kernel='linear', C=C)
    svm_.fit(train_x, train_y)

    y_svm = svm_.predict(test_x)
    acc_svm = sum([1 for i in range(0,len(test_y)) if test_y[i] == y_svm[i] ])/le
n(test_y)

    print("SVM Accuracy: {}%".format(acc_svm*100))

    cm_svm = confusion_matrix(test_y, y_svm)
    print(cm_svm)
    print(classification_report(test_y, y_svm, target_names=class_names))

    disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_svm,display_labels=class_n
ames)
    disp2.plot()
    plt.xlabel('C='+str(C))
    plt.ylabel('Kernel type = Linear')
    plt.show()

for C in [0.1, 1, 10]:
    svm_ = SVC(kernel='poly', C=C)
    svm_.fit(train_x, train_y)

    y_svm = svm_.predict(test_x)
    acc_svm = sum([1 for i in range(0,len(test_y)) if test_y[i] == y_svm[i] ])/le
n(test_y)

    print("SVM Accuracy: {}%".format(acc_svm*100))

    cm_svm = confusion_matrix(test_y, y_svm)
    print(cm_svm)
    print(classification_report(test_y, y_svm, target_names=class_names))

    disp2 = ConfusionMatrixDisplay(confusion_matrix=cm_svm,display_labels=class_n
ames)
    disp2.plot()
    plt.xlabel('C='+str(C))
    plt.ylabel('Kernel type = Polynomial')
    plt.show()
```

Appendix II

```
# ECE 457b
# Kapilan Satkunanathan
# 20694418
# Assignment 2 Question 2

# Import packages
import numpy as np
import matplotlib.pyplot as plt
import time
from IPython import display
from minisom import MiniSom
import math

inputRGB = np.array([
[255,0,0],
[0,255,0],
[0,0,255],
[255,255,0],
[255,0,255],
[0,255,255],
[128,128,0],
[128,0,128],
[0,128,128],
[255,128,0],
[255,0,128],
[128,255,0],
[0,255,128],
[128,0,255],
[0,128,255],
[255,20,147],
[220,20,60],
[255,51,51],
[255,153,51],
[255,255,51],
[51,255,51],
[153,255,51],
[51,255,153],
[51,255,255]])

# Normalize the input (min-
max normalize, but we already know the minimum and maximum)
normRGB = inputRGB/255

plt.imshow(np.reshape(normRGB,(normRGB.shape[0],1,3)))
print(normRGB.shape)

sigma_list = [10,40,70]
for sigma_0 in sigma_list:
    # Initialize the system
    space_size = 100 # 100 x 100 grid of neurons
    alpha_0 = 0.8
    # Nc = 100

    max_epochs = 600

    # Initialize random weights
    w = np.random.random((space_size,space_size,3))
    # diff = np.abs(np.sum(normRGB[0] - w, axis=2))
    plt.imshow(w)
    plt.xlabel('Initial generated neighbourhood')
    plt.ylabel('sigma='+str(sigma_0))
    plt.show()

    epoch = 1
    alpha = alpha_0

    sigma = sigma_0
    while epoch <= max_epochs:

        for x in normRGB:
            # calculate performance index
            diff = np.linalg.norm(x - w, axis =2)
            # find index of winning node
            ind = np.unravel_index(np.argmin(diff, axis=None), diff.shape)


            # Update weights for neighbourhood
            # for i in range(ind[0]-Nc, ind[0]+Nc+1):
            #     for j in range(ind[1]-Nc, ind[1]+Nc+1):
            for i in range(space_size):
                for j in range(space_size):
                    #if i >= 0 and j>=0 and i < space_size and j < space_size:
                    # print(i)
                    # print(j)
                    # make sure you don't exceed the size of the space
                    distance = math.sqrt(abs(i-ind[0])**2+abs(j-ind[1])**2)
                    Nij= math.exp(-((distance**2)/(2*(sigma**2))))
                    w[i][j] += alpha*Nij*(x-w[i][j])


        # decrease the learning rate by the given scheme
        alpha = alpha_0* math.exp(-epoch/(max_epochs+1))
        print("Learning rate decreased to {}".format(alpha))

        # decrease the sigma by the given scheme
        sigma = sigma_0* math.exp(-epoch/(max_epochs+1))
        print("Learning rate decreased to {}".format(alpha))

        plot_ind = [20, 40, 100, 600]
        if epoch in plot_ind:
            ind = plot_ind.index(epoch)
            #f = plt.figure(ind)
            epoch = plot_ind[ind]
            print("Epoch Number: {}".format(epoch))
            #print(w)
            plt.imshow(w)
            # display.clear_output(wait=True)
            plt.xlabel('epoch='+str(epoch))
            plt.ylabel('sigma='+str(sigma_0))
            plt.show()


            #display.display(plt.gcf())


        epoch += 1

    plt.imshow(w)
```

Appendix III

```
# ECE 457b
# Kapilan Satkunanathan
# 20694418
# Assignment 2 Question 2

from __future__ import division
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split


(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
# Split the data into training and testing
train_x, test_x, train_y, test_y = train_test_split(train_images, train_labels, t
est_size=0.20)
train_images = train_x
train_labels = train_y
test_images = test_x
test_labels = test_y


train_labels = to_categorical(train_labels, num_classes=10)
test_labels = to_categorical(test_labels, num_classes=10)
print('train images shape:', train_images.shape)
print('train labels shape:', train_labels.shape)
print('test images shape:', test_images.shape)
print('test labels shape:', test_labels.shape)
```

```
train_images = train_images.astype(np.float32)
mean = np.mean(train_images)
std = np.std(train_images)
train_images = (train_images - mean) / std
test_images = (test_images - mean) / std

# MLP
mlp = Sequential()
mlp.add(Flatten(input_shape=(32, 32, 3)))
mlp.add(Dense(512,input_shape=(3072,), activation='sigmoid'))
mlp.add(Dense(10, activation='softmax'))

# Compile the model
mlp.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
'])
h = mlp.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

# Test the model after training
test_results = mlp.evaluate(test_images, test_labels, verbose=1)
print(f'MLP Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%
')

plt.xlabel('MLP')
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
plt.show()
plt.clf()
plt.cla()

# weightsmlp = mlp.get_weights()[0]
# print(weightsmlp.shape)
# figone = plt.figure(facecolor='w', edgecolor='w', figsize=(8, 8))
# figone.subplots_adjust(wspace=0.05, hspace=0.05)
# for i in range(64):
#  spmlp = figone.add_subplot(8, 8, i+1)
#  spmlp.set_axis_off()
#  plt.xlabel('MLP')
#  plt.imshow(weightsmlp[:, :, 0, i])
#  plt.set_cmap('gray')
# plt.show()
# plt.clf()
# plt.cla()

# First CNN
cnnone = Sequential()
cnnone.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='rel
u'
, input_shape=(32, 32, 3)))
cnnone.add(Flatten())
cnnone.add(Dense(512, activation='sigmoid'))
cnnone.add(Dense(512, activation='sigmoid'))
cnnone.add(Dense(10, activation='softmax'))

# Compile the model
cnnone.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur
acy'])
h = cnnone.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

test_results = cnnone.evaluate(test_images, test_labels, verbose=1)
print(f'CNN One Test results - Loss: {test_results[0]} - Accuracy: {test_results[
1]}%')

plt.xlabel('CNN One')
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
plt.show()
plt.clf()
plt.cla()
# weightsone = cnnone.get_weights()[0]
# print(weightsone.shape)
# figone = plt.figure(facecolor='w', edgecolor='w', figsize=(8, 8))
# figone.subplots_adjust(wspace=0.05, hspace=0.05)
# for i in range(64):
#  spone = figone.add_subplot(8, 8, i+1)
#  spone.set_axis_off()
#  plt.xlabel('CNN one')
#  plt.imshow(weightsone[:, :, 0, i])
#  plt.set_cmap('gray')
# plt.show()
# plt.clf()
# plt.cla()


# Second CNN
cnntwo = Sequential()
cnntwo.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='rel
u'
, input_shape=(32, 32, 3)))
cnntwo.add(MaxPool2D())
cnntwo.add(Conv2D(64, kernel_size=(3, 3), padding='same', activation='relu'))
cnntwo.add(MaxPool2D())
cnntwo.add(Flatten())
cnntwo.add(Dense(512, activation='sigmoid'))
cnntwo.add(Dropout(0.2))
cnntwo.add(Dense(512, activation='sigmoid'))
cnntwo.add(Dropout(0.2))
cnntwo.add(Dense(10, activation='softmax'))

# Compile the model
cnntwo.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accur
acy'])
h = cnntwo.fit(train_images.reshape(-1, 32, 32, 3),
        train_labels,
        batch_size=32,
        epochs=5,
        validation_data=(test_images.reshape(-1, 32, 32, 3), test_labels))

test_results = cnntwo.evaluate(test_images, test_labels, verbose=1)
print(f'CNN Two Test results - Loss: {test_results[0]} - Accuracy: {test_results[
1]}%')

plt.xlabel('CNN Two')
plt.plot(h.history['accuracy'])
plt.plot(h.history['val_accuracy'], 'r')
plt.legend(['train acc', 'val acc'])
plt.show()
plt.clf()
plt.cla()
# weightstwo = cnntwo.get_weights()[0]
# print(weightstwo.shape)
# figtwo = plt.figure(facecolor='w', edgecolor='w', figsize=(8, 8))
# figtwo.subplots_adjust(wspace=0.05, hspace=0.05)
# for i in range(64):
```

```
#   sptwo = figtwo.add_subplot(8, 8, i+1)
#   sptwo.set_axis_off()
#   plt.xlabel('CNN two')
#   plt.imshow(weightstwo[:, :, 0, i])
#   plt.set_cmap('gray')
# plt.show()
# plt.clf()
# plt.cla()
```