Faculty of Engineering
Cairo University

DSP Assignment #4

Name	Section	BN
Ahmed Radwan GadELRab	1	12
Mohamed Ismail Amer	3	42
Moamen Nasser Saad	3	37

Submitted to: Dr. Mohsen Rashawn

Part 1.

General Note: We used batch size of 32 and 100 epochs to train each variation and used Adam Optimizer

Confusion Matrix for Base LeNet-5 CNN

Predicted / Actual	0	1	2	3	4	5	6	7	8	9	All
0	200	0	0	0	0	0	0	0	0	0	200
1	0	199	0	0	0	0	0	0	1	0	200
2	0	0	197	0	0	0	0	1	2	0	200
3	0	0	0	199	0	0	0	1	0	0	200
4	0	0	0	0	197	0	0	1	1	1	200
5	1	1	0	2	0	193	3	0	0	0	200
6	0	1	0	0	0	0	199	0	0	0	200
7	0	0	1	0	0	0	0	199	0	0	200
8	0	0	0	0	0	0	1	0	199	0	200
9	0	0	0	0	1	1	0	2	2	194	200
All	201	201	198	201	198	194	203	204	205	195	2000

Confusion Matrix for the first Variation for LeNet-5 [using tanh as activation function]

Predicted / Actual	0	1	2	3	4	5	6	7	8	9	All
0	197	0	0	0	1	0	1	0	1	0	200
1	0	198	0	0	0	0	0	0	2	0	200
2	2	0	190	5	0	0	0	2	1	0	200
3	0	0	1	195	0	0	0	2	2	0	200
4	0	0	1	0	196	0	0	1	1	1	200
5	1	0	0	1	0	195	2	0	1	0	200
6	0	0	0	0	0	1	199	0	0	0	200
7	0	0	4	0	0	0	0	196	0	0	200
8	0	0	0	1	0	2	3	0	194	0	200
9	0	0	0	0	2	0	0	3	0	195	200
All	200	198	196	202	199	198	205	204	202	196	2000

Confusion Matrix for the second Variation for LeNet-5 [removing the last FC layer (84)]

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	200	0	0	0	0	0	0	0	0	0	200
1	0	200	0	0	0	0	0	0	0	0	200
2	0	1	193	0	0	0	0	1	5	0	200
3	0	0	1	196	0	0	0	1	1	1	200
4	0	0	0	0	200	0	0	0	0	0	200
5	1	0	0	1	0	192	3	0	3	0	200
6	1	0	2	0	0	0	197	0	0	0	200
7	0	0	5	0	0	0	0	195	0	0	200
8	0	0	0	0	0	0	1	0	199	0	200
9	0	0	0	0	3	1	0	3	1	192	200
All	202	201	201	197	203	193	201	200	209	193	2000

Confusion Matrix for the third Variation for LeNet-5 [Increasing the number of filters in first Conv layer to 16]

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	200	0	0	0	0	0	0	0	0	0	200
1	0	199	0	0	0	0	0	0	1	0	200
2	0	0	195	0	0	0	0	2	3	0	200
3	0	0	0	199	0	0	0	1	0	0	200
4	0	0	1	0	197	0	0	1	0	1	200
5	2	0	0	0	0	193	4	0	1	0	200
6	0	0	0	1	0	0	199	0	0	0	200
7	0	0	4	0	0	0	0	196	0	0	200
8	0	0	0	0	0	0	1	0	199	0	200
9	0	0	0	0	0	0	0	2	2	196	200
All	202	199	200	200	197	193	204	202	206	197	2000

		Features					
		DCT		PCA		Your features	
		Accuracy	Processing Time	Accuracy	Processing Time	Accuracy	Processing Time
Classifier							
K-means Clustering	1	62.85%	2.15 s	67.05%	1.8 s	89.45%	0.15 s
	4	86.5%	4.2 s	88.25%	4.3 s	89.25%	1.22 s
	16	93.3%	11.1 s	93%	8.94 s	90.2%	2.5 s
	32	95%	18.23 s	95.35%	14.96 s	91.5%	4.13 s
GMM	1	62.1%	32.26 s	50.3%	19.45 s	86.3%	2.27 s
	2	75.9%	32.01 s	67.55%	28.2 s	85.15%	5.41 s
	4	85.2%	74.6 s	80.9%	51.85 s	88%	9.81 s
SVM	Linear	93.9%	3.32 s	93.35%	3.49 s	89.9%	2.49 s
	nonlinear*	97.6%	2.46 s	97.85%	3.66 s	91.05%	0.653 s

In the CNN no Features are needed					
	Variations	**Accuracy**	**Training time**	**Testing time**	
CNN****	**Variation1:**	98.8%	692479.8	765.5	Base LeNet-5
	Variation2:	97.8%	670278.6	716.3	Using **tanh** as activation function instead of **ReLU**
	Variation3:	98.8%	624274.5	691.0	Removing the last FC layer (84)]
	Variation4:	98.7%	717424.1	616.3	Increasing the number of filters in first Conv layer to 16

CODE (Written on Colab):

```
#Loading Data and Importing Libraries
import numpy as np
```

```
import pandas as pd
import matplotlib as plt
from random import randint
import math
import time
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Input, Dense
from keras import backend as K
from keras.models import load_model
from keras.utils.vis_utils import plot_model
from keras import regularizers
from keras.layers import AveragePooling2D
import scipy as sp
from scipy import io as spio
import warnings
import shutil
from google.colab import drive
import glob
import numpy as np
import PIL
import re
from sklearn.utils import shuffle

warnings.filterwarnings('ignore')
# %matplotlib inline

drive.mount('/content/gdrive')

shutil.rmtree('/content/Reduced MNIST Data', ignore_errors=True)
!cp '/content/gdrive/MyDrive/dataset/MNIST.zip' '/content'

!unzip '/content/MNIST.zip'

#Loading Data
train_dir = "/content/Reduced MNIST Data/Reduced Trainging data"
test_dir = "/content/Reduced MNIST Data/Reduced Testing data"

# lists contains images paths
train_list = []
test_list = []

for i in range(10):
  train_list.append(glob.glob('{}/{}/*.jpg'.format(train_dir,i)))
  test_list.append(glob.glob('{}/{}/*.jpg'.format(test_dir,i)))

train_list = [item for sublist in train_list for item in sublist]
test_list = [item for sublist in test_list for item in sublist]


train_data = np.array([np.array(PIL.Image.open(fname)) \
                               for fname in train_list])
test_data = np.array([np.array(PIL.Image.open(fname)) \
                              for fname in test_list])

# create training and test data
train_label = np.array([x for x in range(10) for y in range(1000)])
test_label = np.array([x for x in range(10) for y in range(200)])

# Shuffle training and test data
train_data, train_label = shuffle(train_data/255, train_label)
test_data, test_label = shuffle(test_data/255, test_label)

X_Train,Y_Train= train_data,train_label
X_Test,Y_Test= test_data,test_label

#Defining some parameters
img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols,1)
```

```
batch_size = 32
num_classes = 10
epochs = 100
X_Train=X_Train.reshape(X_Train.shape[0], img_rows, img_cols, 1)
X_Test=X_Test.reshape(X_Test.shape[0], img_rows, img_cols, 1)

#One Hot Encoding the outputs
n_values = np.max(Y_Train) + 1
Y_Train = np.eye(n_values)[Y_Train.astype(int)]
n_values = np.max(Y_Test) + 1
Y_Test = np.eye(n_values)[Y_Test.astype(int)]



"""#Base Model"""

model= Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', input_shape=input_shape,
padding="same"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', padding='valid'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='ReLU'))
model.add(Dense(84, activation='ReLU'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train CNN
tic = time.time()
hist=model.fit(X_Train, Y_Train, batch_size=batch_size, epochs=epochs)
toc = time.time()
print("\n elapsed time to train Base Model=",round(toc*1000-tic*1000,1),"msec\n")

plt.pyplot.plot(hist.history['loss'])
plt.pyplot.title('Base model loss')
plt.pyplot.ylabel('loss')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

plt.pyplot.plot(hist.history['accuracy'])
plt.pyplot.title('Base model accuracy')
plt.pyplot.ylabel('acc')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

print(model.summary())

score= model.evaluate(X_Test, Y_Test)
print("Test Accuracy for Base Model=  "+ str(round(score[1]*100,1))+"%")

#predict for the test set
tic = time.time()
Y_predict=model.predict(X_Test)
toc = time.time()
print("elapsed time to predict using Base Model=",round(toc*1000-tic*1000,1),"msec\n")

#Confusion matrix
Y_pred=np.argmax(Y_predict, axis=1)
Y_orig=np.argmax(Y_Test, axis=1)
confusion_mat = pd.crosstab(Y_orig,Y_pred,\
                            rownames=['Actual'],colnames=['Predicted'],margins=True)
display(confusion_mat)

del model

"""#First Variation: [using tanh as activtion function]"""
```

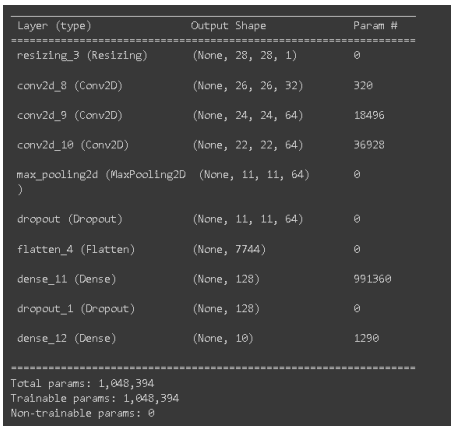
```
model= Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=input_shape,
padding="same"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='tanh'))
model.add(Dense(84, activation='tanh'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train CNN
tic = time.time()
hist=model.fit(X_Train, Y_Train, batch_size=batch_size, epochs=epochs)
toc = time.time()
print("\n elapsed time to train the first variation of the Model=",round(toc*1000-tic*1000,1),"msec\n")

plt.pyplot.plot(hist.history['loss'])
plt.pyplot.title('Base model loss')
plt.pyplot.ylabel('loss')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

plt.pyplot.plot(hist.history['accuracy'])
plt.pyplot.title('Base model accuracy')
plt.pyplot.ylabel('acc')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

print(model.summary())

score= model.evaluate(X_Test, Y_Test)
print("Test Accuracy for the first Variation of the Model=  "+ str(round(score[1]*100,1))+"%")

#predict for the test set
tic = time.time()
Y_predict=model.predict(X_Test)
toc = time.time()
print("elapsed time to predict using the first Variation of the Model=",round(toc*1000-
tic*1000,1),"msec\n")

#Confusion matrix
Y_pred=np.argmax(Y_predict, axis=1)
Y_orig=np.argmax(Y_Test, axis=1)
confusion_mat = pd.crosstab(Y_orig,Y_pred,\
                            rownames=['Actual'],colnames=['Predicted'],margins=True)
display(confusion_mat)

del model

"""#Second Variation: [Removing the last fully connected layer (84)]"""

model= Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', input_shape=input_shape,
padding="same"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', padding='valid'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='ReLU'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train CNN
tic = time.time()
hist=model.fit(X_Train, Y_Train, batch_size=batch_size, epochs=epochs)
```

```
toc = time.time()
print("\n elapsed time to train the second variation of the Model=",round(toc*1000-
tic*1000,1),"msec\n")

plt.pyplot.plot(hist.history['loss'])
plt.pyplot.title('Base model loss')
plt.pyplot.ylabel('loss')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

plt.pyplot.plot(hist.history['accuracy'])
plt.pyplot.title('Base model accuracy')
plt.pyplot.ylabel('acc')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

print(model.summary())

score= model.evaluate(X_Test, Y_Test)
print("Test Accuracy for the second variation of the Model=  "+ str(round(score[1]*100,1))+"%")

#predict for the test set
tic = time.time()
Y_predict=model.predict(X_Test)
toc = time.time()
print("elapsed time to predict using the second variation of the Model=",round(toc*1000-
tic*1000,1),"msec\n")

#Confusion matrix
Y_pred=np.argmax(Y_predict, axis=1)
Y_orig=np.argmax(Y_Test, axis=1)
confusion_mat = pd.crosstab(Y_orig,Y_pred,\
                            rownames=['Actual'],colnames=['Predicted'],margins=True)
display(confusion_mat)

del model

"""#Third Variation: [Changing the number of filters in first Conv layer to to 16]"""

model= Sequential()
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', input_shape=input_shape,
padding="same"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', padding='valid'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='ReLU'))
model.add(Dense(84, activation='ReLU'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train CNN
tic = time.time()
hist=model.fit(X_Train, Y_Train, batch_size=batch_size, epochs=epochs)
toc = time.time()
print("\n elapsed time to train the third variation of the Model=",round(toc*1000-tic*1000,1),"msec\n")

plt.pyplot.plot(hist.history['loss'])
plt.pyplot.title('Base model loss')
plt.pyplot.ylabel('loss')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

plt.pyplot.plot(hist.history['accuracy'])
plt.pyplot.title('Base model accuracy')
plt.pyplot.ylabel('acc')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')
```

```
print(model.summary())

score= model.evaluate(X_Test, Y_Test)
print("Test Accuracy for the third variation of the Model=  "+ str(round(score[1]*100,1))+"%")

#predict for the test set
tic = time.time()
Y_predict=model.predict(X_Test)
toc = time.time()
print("elapsed time to predict using the third variation of the Model=",round(toc*1000-
tic*1000,1),"msec\n")

#Confusion matrix
Y_pred=np.argmax(Y_predict, axis=1)
Y_orig=np.argmax(Y_Test, axis=1)
confusion_mat = pd.crosstab(Y_orig,Y_pred,\
                            rownames=['Actual'],colnames=['Predicted'],margins=True)
display(confusion_mat)

del model
```

Part 2.

We used the data given from the data team and trained it on all the variations we did in part one we also tried other networks like following one



Layer (type)	Output Shape	Param #
resizing_3 (Resizing)	(None, 28, 28, 1)	0
conv2d_8 (Conv2D)	(None, 26, 26, 32)	320
conv2d_9 (Conv2D)	(None, 24, 24, 64)	18496
conv2d_10 (Conv2D)	(None, 22, 22, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 11, 11, 64)	0
dropout (Dropout)	(None, 11, 11, 64)	0
flatten_4 (Flatten)	(None, 7744)	0
dense_11 (Dense)	(None, 128)	991360
dropout_1 (Dropout)	(None, 128)	0
dense_12 (Dense)	(None, 10)	1290

Total params: 1,048,394
Trainable params: 1,048,394
Non-trainable params: 0

and got an accuracy of 41.7%

but we got the best accuracy with the Base Model (LeNet-5)

Also, we changed the input size to match the data (50,500,2) per example and used batch size to 16 and trained on 30 epochs to get a Test accuracy of **73.8%**

This low accuracy could be because of the very little data given (160 spectrograms per digit for the training)

So, this accuracy could be better if there's enough data for this problem

CODE (Written on Colab):

```
#Loading Data
import scipy.io
mat = scipy.io.loadmat('/content/gdrive/MyDrive/SoundData(Version 3).mat')

X_Train =  mat['XTrain'].reshape(mat['XTrain'].shape[0],50,500,1)
X_Test  =  mat['XTest'].reshape(mat['XTest'].shape[0],50,500,1)
Y_Train =  mat['YTrain']
Y_Test =   mat['YTest']

#One Hot Encoding
n_values = np.max(Y_Train) + 1
Y_Train = np.eye(n_values)[Y_Train.astype(int)].reshape(Y_Train.shape[0],10)
n_values = np.max(Y_Test) + 1
Y_Test = np.eye(n_values)[Y_Test.astype(int)].reshape(Y_Test.shape[0],10)

#Spliting real and imagnary into 2 channels
X_Train_real = (X_Train.real.reshape(X_Train.shape[0],50,500))
X_Train_imag = (X_Train.imag.reshape(X_Train.shape[0],50,500))
X_Train = np.stack((X_Train_real,X_Train_imag), axis=3)

X_Test_real = (X_Test.real.reshape(X_Test.shape[0],50,500))
X_Test_imag = (X_Test.imag.reshape(X_Test.shape[0],50,500))
X_Test = np.stack((X_Test_real,X_Test_imag), axis=3)

num_classes = 10

model= Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', input_shape=(50,500,2),
padding="same"))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='ReLU', padding='valid'))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
model.add(Flatten())
model.add(Dense(120, activation='ReLU'))
model.add(Dense(84, activation='ReLU'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train CNN
tic = time.time()
hist=model.fit(X_Train, Y_Train, batch_size=16, epochs=30)
toc = time.time()
print("\n elapsed time to train the Model=",round(toc*1000-tic*1000,1),"msec\n")

plt.pyplot.plot(hist.history['loss'])
plt.pyplot.title('Base model loss')
plt.pyplot.ylabel('loss')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

plt.pyplot.plot(hist.history['accuracy'])
plt.pyplot.title('Base model accuracy')
plt.pyplot.ylabel('acc')
plt.pyplot.xlabel('epoch')
plt.pyplot.legend(['train'], loc='upper left')

print(model.summary())

score= model.evaluate(X_Test, Y_Test)
print("Test Accuracy for the Model=  "+ str(round(score[1]*100,1))+"%")

#predict for the test set
tic = time.time()
Y_predict=model.predict(X_Test)
toc = time.time()
```

```
print("elapsed time to predict using the Model=",round(toc*1000-tic*1000,1),"msec\n")

#Confusion matrix
Y_pred=np.argmax(Y_predict, axis=1)
Y_orig=np.argmax(Y_Test, axis=1)
confusion_mat = pd.crosstab(Y_orig,Y_pred,\
                            rownames=['Actual'],colnames=['Predicted'],margins=True)
display(confusion_mat)

del model
```

You can find the code for the whole assignment on Google Colab with outputs at the following link:

https://colab.research.google.com/drive/1u0jCboFRa7PrWN9dTsQPxEq9yGmsRct-?usp=sharing