Cairo University
Faculty of engineering
Electronics and electrical communication department
DSP2 ELC459

Cairo University

Assignment 3
DSP

Submitted to Dr.: Mohsen Rashwan
Team 11

Submitted by

Name	section	Bench No.
Mohamed Maged Khalil	3	57
Mohamed Mahmoud Abdelmotaleb	3	63
Nour El-din Mohamed Sayed	4	39

May, 2021

Contents

List of Figures

List of tables

1. Introduction

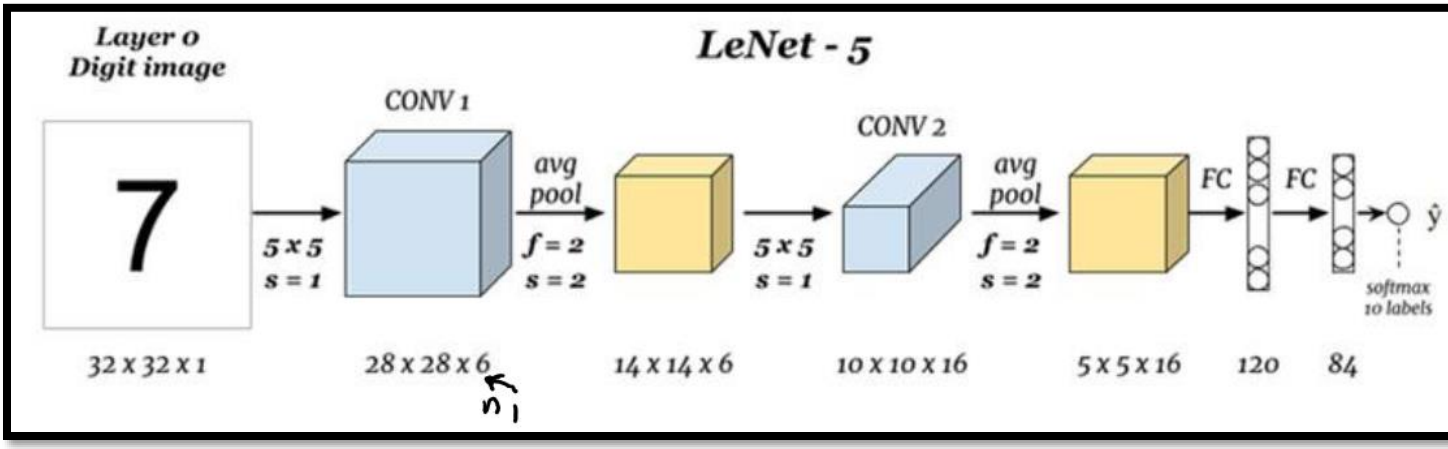


Figure 1-1: Problem Description

1. **Layer C1** is the first Conv-layer with 6 feature maps with strides of 1. The output dimension of this layer 28×28 with 156 trainable parameters. Activation function of this layer is ReLU.
2. **Layer S2** is an average pooling layer. This layer maps average values from the previous Conv layer to the next Conv layer. The Pooling layer is used to reduce the dependence of the model on the location of the features rather than the shape of the features. The pooling layer in LeNet model has a size of 2 and strides of 2.
3. **Layer C3** is the second set of the convolutional layer with 16 feature maps. The output dimension of this layer is 10 with 2,416 parameters. Activation function of this layer is ReLU.
4. **Layer S4** is another average pooling layer with dimension of 2 and stride size of 2.
5. The next layer is responsible for flattening the output of the previous layer into one dimensional array. The output dimension of this layer is 400 (5×5×16).
6. **Layer C5** is a dense block with 120 connections and 48,120 parameters (400×120). Activation function of this layer is ReLU.
7. **Layer F6** is another dense block with 84 parameters and 10,164 parameters (84×120+84). Activation function of this layer is ReLU.
8. **Output Layer** has 10 dimensions (equals number of classes in the database) with 850 parameters (10×84+10). Activation function of output layer is SoftMax.

2. CNN results

2.1. N1=6 with ReLU activation function

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 6)         156

average_pooling2d_8 (Average (None, 14, 14, 6)         0

conv2d_9 (Conv2D)            (None, 10, 10, 16)        2416

average_pooling2d_9 (Average (None, 5, 5, 16)          0

flatten_4 (Flatten)          (None, 400)               0

dense_12 (Dense)             (None, 120)               48120

dense_13 (Dense)             (None, 84)                10164

dense_14 (Dense)             (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

Figure 2-1: N1=6 with ReLU activation function CNN Architecture Details

```
Epoch 21/25
79/79 [==============================] - 5s 65ms/step - loss: 0.0292 - accuracy: 0.9900
Epoch 22/25
79/79 [==============================] - 5s 64ms/step - loss: 0.0200 - accuracy: 0.9942
Epoch 23/25
79/79 [==============================] - 5s 64ms/step - loss: 0.0388 - accuracy: 0.9867
Epoch 24/25
79/79 [==============================] - 5s 65ms/step - loss: 0.0176 - accuracy: 0.9948
Epoch 25/25
79/79 [==============================] - 5s 64ms/step - loss: 0.0228 - accuracy: 0.9930
Time elapsed for Training::  220.92942910400006  sec
```

Figure 2-2: N1=6 with ReLU activation function Training Accuracy and time Details

```
63/63 [==============================] - 1s 9ms/step - loss: 0.0542 - accuracy: 0.9805
Time elapsed for Testing::  0.9001041479996275  sec
```

Figure 2-3: N1=6 with ReLU activation function testing accuracy and time

2.2. N1=8 with Sigmoid activation function

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 8)         208
_________________________________________________________________
average_pooling2d_4 (Average (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 16)        3216
_________________________________________________________________
average_pooling2d_5 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_8 (Dense)              (None, 10)                850
=================================================================
Total params: 62,558
Trainable params: 62,558
Non-trainable params: 0
_________________________________________________________________
```

Figure 2-4: N1=8 with Sigmoid activation function CNN Architecture Details

```
79/79 [==============================] - 5s 64ms/step - loss: 0.0678 - accuracy: 0.9826
Epoch 37/40
79/79 [==============================] - 5s 64ms/step - loss: 0.0655 - accuracy: 0.9817
Epoch 38/40
79/79 [==============================] - 5s 64ms/step - loss: 0.0681 - accuracy: 0.9802
Epoch 39/40
79/79 [==============================] - 5s 63ms/step - loss: 0.0612 - accuracy: 0.9815
Epoch 40/40
79/79 [==============================] - 5s 63ms/step - loss: 0.0582 - accuracy: 0.9824
Time elapsed for Training::  348.6482722369997  sec
```

Figure 2-5: N1=8 with Sigmoid activation function Training Accuracy and time Details

```
63/63 [==============================] - 1s 9ms/step - loss: 0.0794 - accuracy: 0.9735
Time elapsed for Testing::  0.9081572720001532  sec
```

Figure 2-6: N1=8 with Sigmoid activation function Testing Accuracy and time Details

3. Summary results table

Classifier		DCT		PCA		ExtraTree	
		Features					
		Accuracy	Processing Time	Accuracy	Processing Time	Accuracy	Processing Time
K-means Clustering	1	66.95 %	15.961 sec	67.55 %	22.665 sec	73.0 %	14.253 sec
	4	89.85 %	30.722 sec	85.9 %	30.974 sec	90.5 %	33.521 sec
	16	93.2 %	61.533 sec	93.89 %	67.451 sec	93.89 %	59.199 sec
GMM	1	65.0 %	3.2491 sec	64.649 %	7.3275 sec	73.3 %	6.2704 sec
	4	85.1 %	13.423 sec	84.65 %	25.398 sec	88.75 %	26.285 sec
	16	92.45 %	149.31 sec	91.95 %	95.886 sec	93.6 %	83.579 sec
SVM	Linear	93.85 %	1.9082 sec	93.85 %	2.4262 sec	92.45 %	2.3834 sec
	Poly Kernel	96.95 %	1.7343 sec	97.5 %	6.5033 sec	97.6 %	2.3462 sec
	RBF Kernel	97.25 %	2.3123 sec	97.65 %	4.3273 sec	97.75 %	3.4911 sec
	Sigmoid Kernel	86.9 %	3.7651 sec	90.9 %	3.1046 sec	52.7 %	5.7334 sec

CNN		Accuracy	Training Time	Testing time
	N1=6	98.05 %	220.93 sec	0.9 sec
	N1=8	97.35 %	348.64	0.908 sec

Table 1: Summary of results

4. References

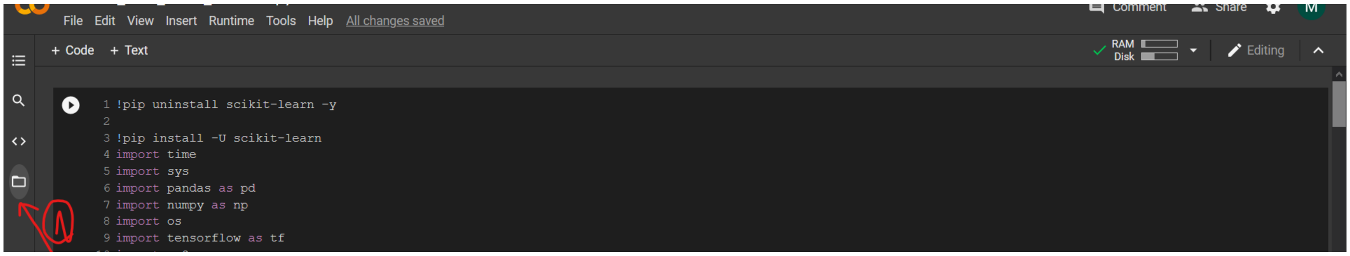
All codes for previous features can be accessed and run directly from google colab.

Steps to run the notebook from a link:

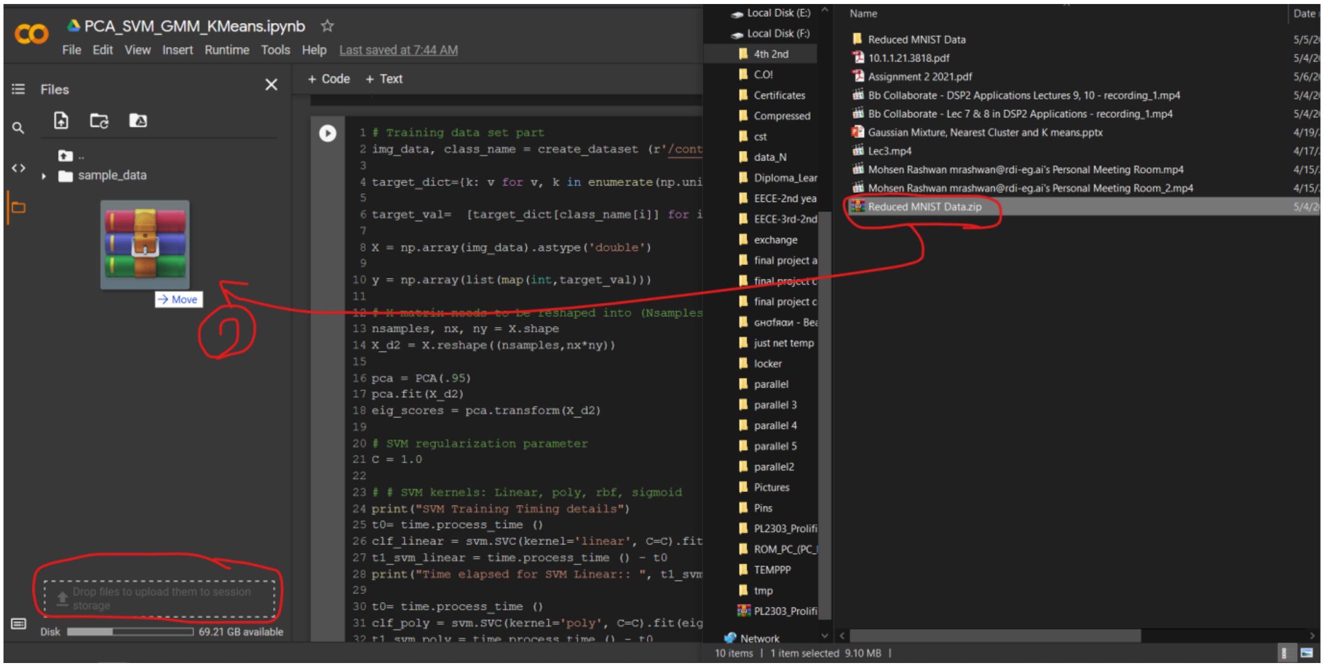
1. Open the Colab link i.e.

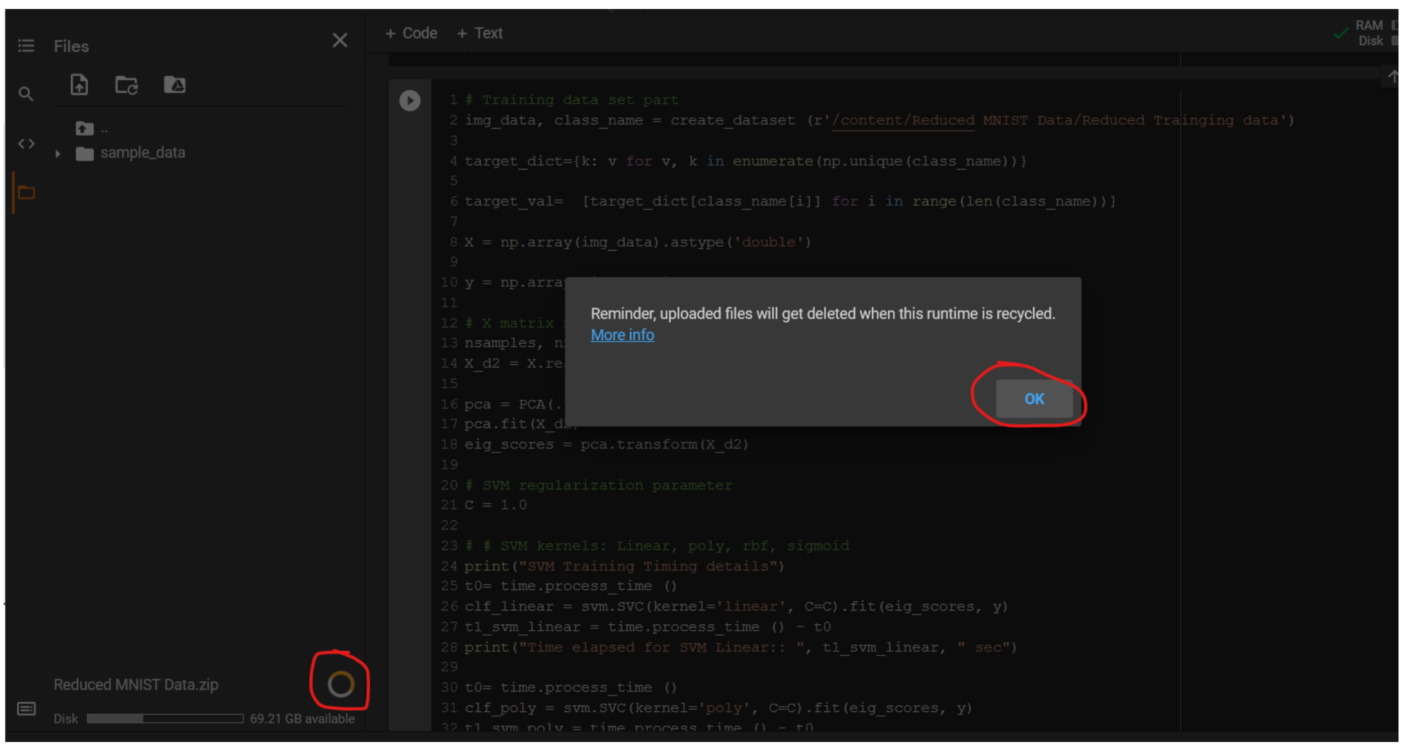
   https://colab.research.google.com/drive/12eaosgBH6RKBFaUc9q8gVWLWC7wNvkyA?usp=sharing#scrollTo=CCUmxJGeiSyH

2. Drag and drop the whole dataset file .zip as in the pictures



Then: Drag and drop the Zip file





After these steps just run the whole cells and the code will work fine. We attached 3 notebooks each one represents the implementation of a feature.

4.1. CNN Colab notebook

Link:

https://colab.research.google.com/drive/12eaosgBH6RKBFaUc9q8gVWLWC7wNvkyA?usp=sharing#scrollTo=CCUmxJGeiSyH

4.2. PCA Features Colab notebook

Link:
https://colab.research.google.com/drive/1GxIH6fa2u5ezw8pjsCZcIcQzAIO65xXf#scrollTo=edqNccj6xeMJ

4.3. DCT Features Colab notebook

Link:

https://colab.research.google.com/drive/1w_c0aICJMq0rSem5vjQOIW91cfAl5MoO

4.4. Extra Trees Features Colab notebook

Link:

https://colab.research.google.com/drive/15oAfAqeWWfPT2N6zG_HSK_SDLgiK-KKD#scrollTo=tXy53Z06xhC_

5. CNN Code

```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Conv2D, Dense, MaxPool2D, Dropout, Flatten, AveragePooling
2D
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
import os
from keras.utils.np_utils import to_categorical
import cv2
import time
import tensorflow as tf
def create_dataset(img_folder):

    img_data_array=[]
```

```
    class_name=[]

    for dir1 in os.listdir(img_folder):
        for file in os.listdir(os.path.join(img_folder, dir1)):
            image_path= os.path.join(img_folder, dir1,  file)
            image= cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            image=np.array(image)
            image = image.astype('double')
            image /= 255
            img_data_array.append(image)
            class_name.append(dir1)
    return img_data_array, class_name

activation_type = 'relu'

filters_tried = 6

model = Sequential()
model.add(Conv2D(filters=filters_tried, kernel_size=(5,5), padding='same', strides=
(1, 1), activation=activation_type, input_shape=(28, 28, 1)))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(filters=16, kernel_size=(5,5), strides=(1, 1), padding='valid', ac
tivation=activation_type))
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation=activation_type))
model.add(Dense(84, activation=activation_type))
model.add(Dense(10, activation='softmax'))

model.build()
model.summary()

filters_tried = 8
activation_type = 'sigmoid'
model2 = Sequential()
model2.add(Conv2D(filters=filters_tried, kernel_size=(5,5), padding='same', strides
=(1, 1), activation=activation_type, input_shape=(28, 28, 1)))
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))
model2.add(Conv2D(filters=16, kernel_size=(5,5), strides=(1, 1), padding='valid', a
ctivation=activation_type))
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))
model2.add(Flatten())
model2.add(Dense(120, activation=activation_type))
model2.add(Dense(84, activation=activation_type))
model2.add(Dense(10, activation='softmax'))

model2.build()
model2.summary()
```

```
adam = Adam(lr=0.01)
model.compile(optimizer='adam',loss=tf.keras.losses.categorical_crossentropy,metric
s=['accuracy'])
model2.compile(optimizer='adam',loss=tf.keras.losses.categorical_crossentropy,metri
cs=['accuracy'])
# model.fit(X_train ,Y_train, batch_size=128, steps_per_epoch=len(X_train)/100, epo
chs=30)
# Training data set part
img_data, class_name = create_dataset (r'/content/Reduced MNIST Data/Reduced Traing
ing data')

target_dict={k: v for v, k in enumerate(np.unique(class_name))}

target_val=  [target_dict[class_name[i]] for i in range(len(class_name))]

X_train = np.array(img_data).astype('double')

Y_train = np.array(list(map(int,target_val)))

nsamples, nx, ny = X_train.shape
X_train = X_train.reshape((nsamples,nx*ny))

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
Y_train = to_categorical(Y_train)

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

t0= time.process_time ()
model.fit(x= X_train, y=Y_train, epochs=25, batch_size=128, callbacks=[callback])
t1_svm_linear = time.process_time () - t0
print("Time elapsed for Training:: ", t1_svm_linear, " sec")

t0= time.process_time ()
model2.fit(x= X_train, y=Y_train, epochs=40, batch_size=128, callbacks=[callback])
t1_svm_linear = time.process_time () - t0
print("Time elapsed for Training:: ", t1_svm_linear, " sec")
## Test dataset part
img_data_test, class_name_test = create_dataset(r'/content/Reduced MNIST Data/Reduc
ed Testing data')

target_dict_test={k: v for v, k in enumerate(np.unique(class_name_test))}

target_val_test=  [target_dict_test[class_name_test[i]] for i in range(len(class_na
me_test))]

X_test = np.array(img_data_test)
```

```
y_test = np.array(list(map(int, target_val_test)))

nsamples, nx, ny = X_test.shape
X_d2_test = X_test.reshape((nsamples,nx*ny))

X_test = X_d2_test.reshape(X_d2_test.shape[0], 28, 28, 1)
Y_test = to_categorical(y_test)

t0= time.process_time ()
score = model.evaluate(X_test, Y_test, batch_size=32)
t1_svm_linear = time.process_time () - t0
print("Time elapsed for Testing:: ", t1_svm_linear, " sec")

t0= time.process_time ()
score = model2.evaluate(X_test, Y_test, batch_size=32)
t1_svm_linear = time.process_time () - t0
print("Time elapsed for Testing:: ", t1_svm_linear, " sec")
```