	Academic Year:	2022/2023	Term:	Spring 2023	
	Course Code:	ELC 4028	Course Title:	Artificial Neural Networks and its Applications	

Cairo University

Faculty of Engineering

Electronics and Communications Engineering Department – 4th Year

Neural Networks Applications

- Assignment 2 -

Submitted to: Dr. Mohsen Rashwan

Name	BN	Sec	ID	رقم الجلوس
احمد محمود حسيني عطية	22	1	9180178	34022
علي ماهر عبدالسلام نبيه	5	3	9190067	34117
محمد احمد طه السيد	30	3	9191043	34142
محمد حسام عثمان يسن	35	3	9191083	34147
محمد عاطف ربيع	43	3	9190924	34155

Table of Contents

1. Multi-layer Perceptron

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.utils import shuffle
from scipy.fftpack import dct ,idct
import sklearn
from sklearn.decomposition import PCA, FastICA
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense, Flatten, InputLayer, Dropout
import time
from tensorflow.keras.callbacks import EarlyStopping

training_path = "Reduced_MNIST_Data\Reduced_Training_data"
testing_path = "Reduced_MNIST_Data\Reduced_Testing_data"
# Define the list of classes
classes = os.listdir(training_path)
print(classes)
classes = list(map(int, classes))
print(classes)
# Define an empty list to store the data and labels
X_train = []
y_train = []
# Loop over the classes
for class_name in classes:
    class_path = os.path.join(training_path, str(class_name))
    # Loop over the images in the class folder
    for image_name in os.listdir(class_path):
        image_path = os.path.join(class_path, image_name)
        # Load the image and append it to the data list
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        X_train.append(image)
        # Append the label to the labels list
        y_train.append(class_name)
# Convert the data and labels lists to NumPy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
# Print the shape of the data and labels arrays
print("Training Data shape:", X_train.shape)
print("Training Labels shape:", y_train.shape)
X_test = []
y_test = []
for class_name in classes:
    class_path = os.path.join(testing_path, str(class_name))

    for image_name in os.listdir(class_path):
        image_path = os.path.join(class_path, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        X_test.append(image)
        y_test.append(class_name)
```

```
# Convert the data and labels lists to NumPy arrays
X_test = np.array(X_test)
y_test = np.array(y_test)
print("Testing Data shape:", X_test.shape)
print("Testing Labels shape:", y_test.shape)

X_train,y_train = shuffle(X_train, y_train, random_state=4)
X_test,y_test = shuffle(X_test, y_test, random_state=4)

#check if shuffling worked correctly
plt.figure()
plt.subplot(121)
plt.title("Is this {} ?".format(y_train[1050]))
plt.imshow(X_train[1050])

plt.subplot(122)
plt.title("Is this {} ?".format(y_test[1050]))
plt.imshow(X_test[1050])
plt.show()

# ## DCT Features
# Functions used to extract DCT features
def zigzag(a):
    comp=np.concatenate([np.diagonal(a[::-1,:], i)[::(2*(i % 2)-1)] for i in range(1-
a.shape[0], a.shape[0])])
    return comp[:200]

def dct_extract(a):
    features=np.zeros((a.shape[0],200))
    for i in range(a.shape[0]):
        z_features=zigzag(dct(dct(a[i].T, norm='ortho').T, norm='ortho'))
        features[i]=z_features

    extracted=features.reshape((a.shape[0],-1))

    return extracted

#Extract DCT features for training and testing data
X_train_DCT=dct_extract(X_train)
X_test_DCT=dct_extract(X_test)

X_train_DCT.shape

# ## PCA Features
pca_model = PCA(.95) #we want a 95% variance
pca_model.fit(X_train.reshape((X_train.shape[0],28*28)))
X_train_PCA = pca_model.transform(X_train.reshape((X_train.shape[0],28*28)))
X_test_PCA = pca_model.transform(X_test.reshape((X_test.shape[0],28*28)))
print("For 95% varinace, there are {} components".format(pca_model.n_components_))

X_train_PCA.shape

# ## ICA Features
ica_model = FastICA(n_components=200)
```

```
X_train_ICA = ica_model.fit_transform(X_train.reshape((X_train.shape[0],784)), y_train)
X_test_ICA = ica_model.transform(X_test.reshape((X_test.shape[0],784)))

X_train_ICA.shape
```

Training a Multi-layer Perceptron (MLP)

Using DCT Features

```
# ### 1 Hidden Layer

# Define the model architecture
model_MLP1_DCT = Sequential(name='MLP1_DCT')

model_MLP1_DCT.add(Dense(256, activation='relu', input_shape=(200,)))  # hidden layer
model_MLP1_DCT.add(Dropout(0.2)) #dropout regularization
model_MLP1_DCT.add(Dense(10, activation='softmax'))  # Output layer

model_MLP1_DCT.summary()

#Early Stopping to avoid fitting issues
early_stopping = EarlyStopping(monitor='accuracy', patience=3)

# Compile the model
model_MLP1_DCT.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP1_DCT.fit(X_train_DCT, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP1_DCT.evaluate(X_test_DCT, y_test)

X_test_DCT[0].shape

tic=time.time()
model_MLP1_DCT.predict(X_test_DCT[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 1 Hidden Layer-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

### 2 Hidden Layers

# Define the model architecture
model_MLP2_DCT = Sequential(name='MLP2_DCT')
```

```
model_MLP2_DCT.add(Dense(256, activation='relu', input_shape=(200,)))  # 1st hidden layer
model_MLP2_DCT.add(Dropout(0.2))
model_MLP2_DCT.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP2_DCT.add(Dropout(0.2))
model_MLP2_DCT.add(Dense(10, activation='softmax'))  # Output layer

model_MLP2_DCT.summary()

# Compile the model
model_MLP2_DCT.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP2_DCT.fit(X_train_DCT, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP2_DCT.evaluate(X_test_DCT, y_test)

tic=time.time()
model_MLP2_DCT.predict(X_test_DCT[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 2 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

# ### 3 Hidden Layers

# Define the model architecture
model_MLP3_DCT = Sequential(name='MLP3_DCT')

model_MLP3_DCT.add(Dense(256, activation='relu', input_shape=(200,)))  # 1st hidden layer
model_MLP3_DCT.add(Dropout(0.2))
model_MLP3_DCT.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP3_DCT.add(Dropout(0.2))
model_MLP3_DCT.add(Dense(64, activation='relu'))  # 3rd hidden layer
model_MLP3_DCT.add(Dropout(0.2))
model_MLP3_DCT.add(Dense(10, activation='softmax'))  # Output layer

model_MLP3_DCT.summary()

# Compile the model
model_MLP3_DCT.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
```

```
model_MLP3_DCT.fit(X_train_DCT, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP3_DCT.evaluate(X_test_DCT, y_test)

tic=time.time()
model_MLP3_DCT.predict(X_test_DCT[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 3 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Using PCA Features

```
# ### 1 Hidden Layer

# Define the model architecture
model_MLP1_PCA = Sequential(name='MLP1_PCA')

model_MLP1_PCA.add(Dense(256, activation='relu', input_shape=(262,)))  # hidden layer
model_MLP1_PCA.add(Dropout(0.2)) #dropout regularization
model_MLP1_PCA.add(Dense(10, activation='softmax'))  # Output layer

model_MLP1_PCA.summary()

# Compile the model
model_MLP1_PCA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP1_PCA.fit(X_train_PCA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP1_PCA.evaluate(X_test_PCA, y_test)

tic=time.time()
model_MLP1_PCA.predict(X_test_PCA[0].reshape(1,262))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 1 Hidden Layer-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
```

```
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))


# ### 2 Hidden Layers

# Define the model architecture
model_MLP2_PCA = Sequential(name='MLP2_PCA')

model_MLP2_PCA.add(Dense(256, activation='relu', input_shape=(262,)))  # 1st hidden layer
model_MLP2_PCA.add(Dropout(0.2))
model_MLP2_PCA.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP2_PCA.add(Dropout(0.2))
model_MLP2_PCA.add(Dense(10, activation='softmax'))  # Output layer

model_MLP2_PCA.summary()

# Compile the model
model_MLP2_PCA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP2_PCA.fit(X_train_PCA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP2_PCA.evaluate(X_test_PCA, y_test)

tic=time.time()
model_MLP2_PCA.predict(X_test_PCA[0].reshape(1,262))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 2 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

# ### 3 Hidden Layers

# Define the model architecture
model_MLP3_PCA = Sequential(name='MLP3_PCA')

model_MLP3_PCA.add(Dense(256, activation='relu', input_shape=(262,)))  # 1st hidden layer
model_MLP3_PCA.add(Dropout(0.2))
model_MLP3_PCA.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP3_PCA.add(Dropout(0.2))
model_MLP3_PCA.add(Dense(64, activation='relu'))  # 3rd hidden layer
model_MLP3_PCA.add(Dropout(0.2))
model_MLP3_PCA.add(Dense(10, activation='softmax'))  # Output layer
```

```
model_MLP3_PCA.summary()

# Compile the model
model_MLP3_PCA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP3_PCA.fit(X_train_PCA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP3_PCA.evaluate(X_test_PCA, y_test)

tic=time.time()
model_MLP3_PCA.predict(X_test_PCA[0].reshape(1,262))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 3 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Using ICA Features

```
# ### 1 Hidden Layer

# Define the model architecture
model_MLP1_ICA = Sequential(name='MLP1_ICA')

model_MLP1_ICA.add(Dense(256, activation='relu', input_shape=(200,)))  # hidden layer
model_MLP1_ICA.add(Dropout(0.2)) #dropout regularization
model_MLP1_ICA.add(Dense(10, activation='softmax'))  # Output layer

model_MLP1_ICA.summary()

# Compile the model
model_MLP1_ICA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP1_ICA.fit(X_train_ICA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP1_ICA.evaluate(X_test_ICA, y_test)
```

```
tic=time.time()
model_MLP1_ICA.predict(X_test_ICA[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 1 Hidden Layer-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

# ### 2 Hidden Layers

# Define the model architecture
model_MLP2_ICA = Sequential(name='MLP2_ICA')

model_MLP2_ICA.add(Dense(256, activation='relu', input_shape=(200,)))  # 1st hidden layer
model_MLP2_ICA.add(Dropout(0.2))
model_MLP2_ICA.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP2_ICA.add(Dropout(0.2))
model_MLP2_ICA.add(Dense(10, activation='softmax'))  # Output layer

model_MLP2_ICA.summary()

# Compile the model
model_MLP2_ICA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP2_ICA.fit(X_train_ICA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP2_ICA.evaluate(X_test_ICA, y_test)

tic=time.time()
model_MLP2_ICA.predict(X_test_ICA[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 2 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

# ### 3 Hidden Layers

# Define the model architecture
model_MLP3_ICA = Sequential(name='MLP3_ICA')

model_MLP3_ICA.add(Dense(256, activation='relu', input_shape=(200,)))  # 1st hidden layer
```

```
model_MLP3_ICA.add(Dropout(0.2))
model_MLP3_ICA.add(Dense(128, activation='relu'))  # 2nd hidden layer
model_MLP3_ICA.add(Dropout(0.2))
model_MLP3_ICA.add(Dense(64, activation='relu'))  # 3rd hidden layer
model_MLP3_ICA.add(Dropout(0.2))
model_MLP3_ICA.add(Dense(10, activation='softmax'))  # Output layer

model_MLP3_ICA.summary()

# Compile the model
model_MLP3_ICA.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
tic=time.time()
model_MLP3_ICA.fit(X_train_ICA, y_train, epochs=30, batch_size=32,
callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
test_loss, test_acc = model_MLP3_ICA.evaluate(X_test_ICA, y_test)

tic=time.time()
model_MLP3_ICA.predict(X_test_ICA[0].reshape(1,200))
toc=time.time()
proc_time=toc-tic

print("-----MLP With 3 Hidden Layers-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Processing Time for 1 example = {} ms".format(np.round(proc_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

2. Convolutional Neural Network (LeNet-5)

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.utils import shuffle
from scipy.fftpack import dct ,idct
import sklearn
from sklearn.decomposition import PCA, FastICA
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense, Flatten, InputLayer, Dropout, Conv2D, AveragePooling2D
import time
from tensorflow.keras.callbacks import EarlyStopping

training_path = "Reduced_MNIST_Data\Reduced_Training_data"
testing_path = "Reduced_MNIST_Data\Reduced_Testing_data"

# Define the list of classes
classes = os.listdir(training_path)
print(classes)
classes = list(map(int, classes))
print(classes)
# Define an empty list to store the data and labels
X_train = []
y_train = []
# Loop over the classes
for class_name in classes:
    class_path = os.path.join(training_path, str(class_name))
    # Loop over the images in the class folder
    for image_name in os.listdir(class_path):
        image_path = os.path.join(class_path, image_name)
        # Load the image and append it to the data list
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        X_train.append(image)
        # Append the label to the labels list
        y_train.append(class_name)
```

```
# Convert the data and labels lists to NumPy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
# Print the shape of the data and labels arrays
print("Training Data shape:", X_train.shape)
print("Training Labels shape:", y_train.shape)
X_test = []
y_test = []

for class_name in classes:
    class_path = os.path.join(testing_path, str(class_name))

    for image_name in os.listdir(class_path):
        image_path = os.path.join(class_path, image_name)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        X_test.append(image)
        y_test.append(class_name)

# Convert the data and labels lists to NumPy arrays
X_test = np.array(X_test)
y_test = np.array(y_test)
print("Testing Data shape:", X_test.shape)
print("Testing Labels shape:", y_test.shape)

X_train,y_train = shuffle(X_train, y_train, random_state=4)
X_test,y_test = shuffle(X_test, y_test, random_state=4)
#check if shuffling worked correctly
plt.figure()
plt.subplot(121)
plt.title("Is this {} ?".format(y_train[1050]))
plt.imshow(X_train[1050])

plt.subplot(122)
plt.title("Is this {} ?".format(y_test[1050]))
plt.imshow(X_test[1050])
plt.show()

#reshaping the dataset to fit CNN architectures
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
```

```
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
print(X_train.shape)
print(X_test.shape)
```

LeNet-5 - No Variations

```
model = Sequential()
# Convolutional layer 1
model.add(Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1), padding='valid'))
# Average pooling layer 1
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model.add(Conv2D(16, (5, 5), activation='relu', padding='valid'))
# Average pooling layer 2
model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model.add(Flatten())
# Fully connected layer 1
model.add(Dense(120, activation='relu'))
# Fully connected layer 2
model.add(Dense(84, activation='relu'))
# Output layer
model.add(Dense(10, activation='softmax'))

#Early Stopping to avoid fitting issues
early_stopping = EarlyStopping(monitor='accuracy', patience=3)
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Train the model
tic=time.time()
model.fit(X_train, y_train, epochs=30, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model.evaluate(X_test, y_test)
toc=time.time()
test_time=toc-tic
print("-----LeNet-5 - No Variations-----\n")
```

```python
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Variation #1 - Adding Dropout Regularization

```python
model1 = Sequential()
# Convolutional layer 1
model1.add(Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1),
padding='valid'))
#dropout regularization
model1.add(Dropout(0.2))
# Average pooling layer 1
model1.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model1.add(Conv2D(16, (5, 5), activation='relu', padding='valid'))
model1.add(Dropout(0.2))
# Average pooling layer 2
model1.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model1.add(Flatten())
# Fully connected layer 1
model1.add(Dense(120, activation='relu'))
model1.add(Dropout(0.2))
# Fully connected layer 2
model1.add(Dense(84, activation='relu'))
model1.add(Dropout(0.2))
# Output layer
model1.add(Dense(10, activation='softmax'))

model1.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
tic=time.time()
model1.fit(X_train, y_train, epochs=30, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic
# Evaluate the model on the test data
tic=time.time()
```

```
test_loss, test_acc = model1.evaluate(X_test, y_test)
toc=time.time()
test_time=toc-tic
print("-----Variation #1 - Adding Dropout-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Variation #2 - Increasing Number of Filters in Conv Layers

```
model2 = Sequential()
# Convolutional layer 1
model2.add(Conv2D(12, (5, 5), activation='relu', input_shape=(28, 28, 1),
padding='valid'))
# Average pooling layer 1
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model2.add(Conv2D(32, (5, 5), activation='relu', padding='valid'))
# Average pooling layer 2
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model2.add(Flatten())
# Fully connected layer 1
model2.add(Dense(120, activation='relu'))
# Fully connected layer 2
model2.add(Dense(84, activation='relu'))
# Output layer
model2.add(Dense(10, activation='softmax'))

model2.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Train the model
tic=time.time()
model2.fit(X_train, y_train, epochs=30, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic
# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model2.evaluate(X_test, y_test)
toc=time.time()
```

```
test_time=toc-tic
print("-----Variation #2 - Increasing no. of Filters-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time= {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Variation #3 - Adding "Same" Padding to Conv Layers

```
model3 = Sequential()
# Convolutional layer 1
model3.add(Conv2D(12, (5, 5), activation='relu', input_shape=(28, 28, 1),
padding='same'))
# Average pooling layer 1
model3.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model3.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
# Average pooling layer 2
model3.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model3.add(Flatten())
# Fully connected layer 1
model3.add(Dense(120, activation='relu'))
# Fully connected layer 2
model3.add(Dense(84, activation='relu'))
# Output layer
model3.add(Dense(10, activation='softmax'))

model3.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Train the model
tic=time.time()
model3.fit(X_train, y_train, epochs=30, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic
# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model3.evaluate(X_test, y_test)
toc=time.time()
test_time=toc-tic
print("-----Variation #3 - Adding 'Same' Padding-----\n")
```

```
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

Variation #4 - Using "Tanh" Activation

```
model4 = Sequential()
# Convolutional layer 1
model4.add(Conv2D(12, (5, 5), activation='tanh', input_shape=(28, 28, 1),
padding='valid'))
# Average pooling layer 1
model4.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model4.add(Conv2D(32, (5, 5), activation='tanh', padding='valid'))
# Average pooling layer 2
model4.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model4.add(Flatten())
# Fully connected layer 1
model4.add(Dense(120, activation='tanh'))
# Fully connected layer 2
model4.add(Dense(84, activation='tanh'))
# Output layer
model4.add(Dense(10, activation='softmax'))
model4.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# Train the model
tic=time.time()
model4.fit(X_train, y_train, epochs=30, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic
# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model4.evaluate(X_test, y_test)
toc=time.time()
test_time=toc-tic
print("-----Variation #4 - Using Tanh Activation-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))
```

3. Comparing the Results

Table 1: Comparative Analysis for Different Models

Classifier		Features					
		DCT		PCA		ICA	
		Accuracy	**Training Time**	**Accuracy**	**Training Time**	**Accuracy**	**Training Time**
K-means Clustering	1	62.65%	0.619s	63.15%	0.903s	64.4%	0.166s
	4	89%	1.221s	88.65%	1.812s	81.5%	0.542s
	16	93.15%	3.504s	93.25%	4.783s	89.35%	1.424s
	32	**95.4%**	6.798s	94.75%	9.291s	89.1%	1.872s
SVM	Linear	94.35%	1.808s	93.85%	3.814s	77.8%	6.240s
	Non-Linear (RBF)	97.35%	2.617s	**97.65%**	7.158s	93.8%	0.783s
Multi-layer Perceptron (MLP)							
		DCT		PCA		ICA	
	Variations	**Accuracy**	**Processing Time**	**Accuracy**	**Processing Time**	**Accuracy**	**Processing Time**
MLP	1-Hidden	95.0%	271.3 ms	**95.20%**	246.3 ms	93.20%	70.8 ms
	2-Hidden	94.30%	187.5 ms	93.80%	413.9 ms	**95.00%**	88.5 ms
	3-Hidden	**95.70%**	197.5 ms	94.70%	218.4 ms	94.70%	93.5 ms
CNN – No Features							
	Variations	**Accuracy**		**Training Time**		**Testing Time**	
CNN	No Variations	97.40%		41.9 s		515.6 ms	
	Dropout	**98.60%**		76.5 s		505.6 ms	
	Increasing Number of Filters	98.50%		40.9 s		493.2 ms	
	“Same” Padding	97.80%		61.7 s		614.4 ms	
	Tanh Activation	98.30%		50.9 s		1004.3 ms	

Notes

- The Multi-layer perceptron processing time measurements are based on how much time it takes the model to predict the class of one image.
- The Convolutional Neural Network training time measurements are for different number of epochs, considering Early Stopping was used to avoid fitting issues.
- The time measurements throughout the experiments are heavily dependent on the machine the models are running on and the processes that run on that machine.
- Adding more layers in Fully connected Networks might not always be the best option, as there will be diminishing returns in the accuracy.
- Dropout regularization (and regularization in general) increases the performance of the model, even slightly, as it reduces overfitting, and therefore the model generalizes better.

4. Digit Spectrograms

Import Libraries

```
import os
from matplotlib import pyplot as plt
import tensorflow as tf
!pip install tensorflow_io
import tensorflow_io as tfio
from tensorflow import keras
from keras import backend as k
import cv2
import numpy as np
import random
from sklearn.utils import shuffle
from scipy.fftpack import dct ,idct
import sklearn
from sklearn.decomposition import PCA, FastICA
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense, Flatten, InputLayer, Dropout, Conv2D, AveragePooling2D,MaxPool2D
import time
from tensorflow.keras.callbacks import EarlyStopping
```

```
from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", f
orce_remount=True).
```

Variables: batch: the process of splitting the training dataset in n batches (mini-batches), classes: number of classifications (labels) of the data, epochs: variations, one epoch is one forward pass + one backward pass on training

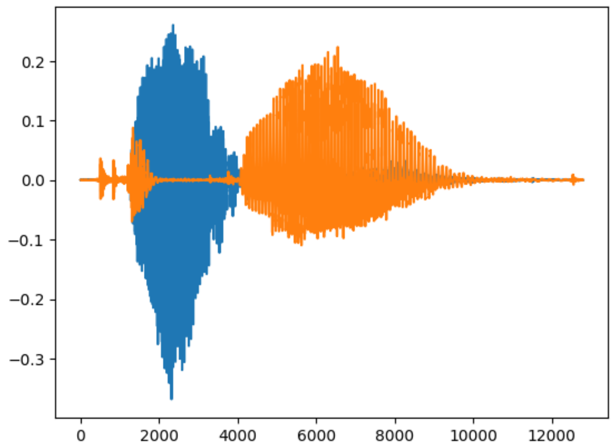
```
#batch_size = 20
num_classes = 10
epochs = 4
```

a function that returns audio in numeric representation

```
def load_wav_16k_mono(filename):
    # Load encoded wav file
    file_contents = tf.io.read_file(filename)
    # Decode wav (tensors by channels)
    wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
    # Removes trailing axis
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    # Goes from 44100Hz to 16000hz - amplitude of the audio signal
    #wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav
```

```
TRAIN_FILE = os.path.join('/content','drive','MyDrive','audio-data','Train','C03n_0.wav')
TEST_FILE = os.path.join('/content','drive','MyDrive','audio-data','Test','C04n_2.wav')

wave = load_wav_16k_mono(TRAIN_FILE)
nwave = load_wav_16k_mono(TEST_FILE)
plt.plot(wave)
plt.plot(nwave)
plt.show()
```



Check a sample of audio

```
TRAIN = os.path.join('/content','drive','MyDrive','audio-data', 'Train')
TEST = os.path.join('/content','drive','MyDrive','audio-data', 'Test')
```

Read all audio files and sort

```
train = tf.data.Dataset.list_files(TRAIN+'/*.wav')
train = sorted(list(train.as_numpy_iterator()))
train = tf.data.Dataset.from_tensor_slices(train)
test = tf.data.Dataset.list_files(TEST+'/*.wav')
test = sorted(list(test.as_numpy_iterator()))
test = tf.data.Dataset.from_tensor_slices(test)
```

Add Labels

```
iterations = 0
i = 0
train_label = []
while iterations!=len(train):
    iterations +=1
    train_label.append(i)
    i += 1
    if i == 10 :
        i = 0
train_label=keras.utils.to_categorical(train_label,num_classes)
trainings = tf.data.Dataset.zip((train, tf.data.Dataset.from_tensor_slices(train_label)))
#----------------------------------------------------------------#
iterations = 0
i = 0
test_label=[]
while iterations!=len(test):
    iterations +=1
    test_label.append(i)
    i += 1
    if i == 10 :
        i = 0
test_label=keras.utils.to_categorical(test_label,num_classes)
testings = tf.data.Dataset.zip((test, tf.data.Dataset.from_tensor_slices(test_label)))
```

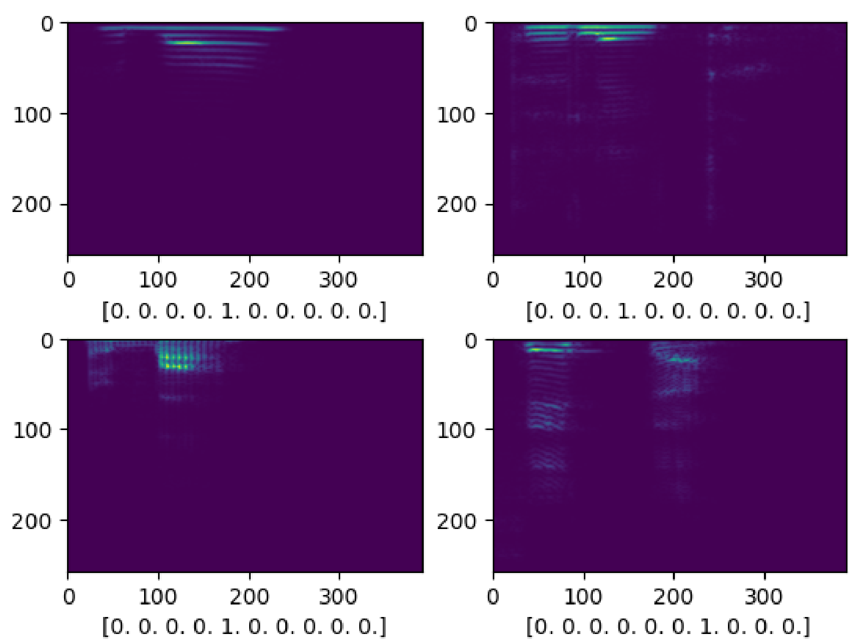
Build Preprocessing Function to get spectogram

```
def preprocess(file_path, label):
    wav = load_wav_16k_mono(file_path)
    #wav = wav[:48000]
    #zero_padding = tf.zeros([48000] - tf.shape(wav), dtype=tf.float32)
    #wav = tf.concat([zero_padding, wav],0)
    spectrogram = tf.signal.stft(wav, frame_length=320, frame_step=32)
    spectrogram = tf.abs(spectrogram)
```

```
    spectrogram = tf.expand_dims(spectrogram, axis=2)
    return spectrogram, label
```

Draw examples of spectogram

```
for i in range(4):
    filepath, label = trainings.shuffle(buffer_size=10000).as_numpy_iterator().next()
    spectrogram, label = preprocess(filepath, label)
    plt.subplot(2,2,i+1)
    plt.imshow(tf.transpose(spectrogram)[0])
    plt.xlabel(label)
plt.show()
```



Convert all to Spectogram

```
# train data
x_train = trainings.map(preprocess)
x_train = x_train.cache()
x_train = x_train.shuffle(buffer_size=1000)
x_train = x_train.batch(16) # 16 at a time
x_train = x_train.prefetch(8)
# test data
x_test = testings.map(preprocess)
x_test = x_test.cache()
x_test = x_test.shuffle(buffer_size=1000)
x_test = x_test.batch(16) # 16 at a time
x_test = x_test.prefetch(8)

# test one batch
samples, labels = x_train.as_numpy_iterator().next()
print(samples.shape)
print('\n',labels)
```

```
(16, 391, 257, 1)

 [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

Design the CNN architecture

the 1st model

```
model1 = Sequential()
# Convolutional layer 1
model1.add(Conv2D(6, (5, 5), activation='relu', input_shape=(391, 257, 1), padding='valid'))
# Average pooling layer 1
model1.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model1.add(Conv2D(16, (5, 5), activation='relu', padding='valid'))
# Average pooling layer 2
model1.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model1.add(Flatten())
# Fully connected layer 1
model1.add(Dense(120, activation='relu'))
# Fully connected layer 2
model1.add(Dense(84, activation='relu'))
# Output layer
model1.add(Dense(10, activation='softmax'))
model1.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 387, 253, 6)       156

 average_pooling2d_24 (Avera  (None, 193, 126, 6)      0
 gePooling2D)

 conv2d_25 (Conv2D)          (None, 189, 122, 16)      2416

 average_pooling2d_25 (Avera  (None, 94, 61, 16)       0
 gePooling2D)

 flatten_12 (Flatten)        (None, 91744)             0

 dense_36 (Dense)            (None, 120)               11009400

 dense_37 (Dense)            (None, 84)                10164

 dense_38 (Dense)            (None, 10)                850

=================================================================
Total params: 11,022,986
Trainable params: 11,022,986
Non-trainable params: 0
_________________________________________________________________
```

```
#Early Stopping to avoid fitting issues
early_stopping = EarlyStopping(monitor='accuracy', patience=5)

model1.compile(loss = keras.losses.CategoricalCrossentropy(), optimizer='adam', metrics=['accuracy'])
```

```
# Train the model
tic=time.time()
model1.fit(x_train, epochs=80, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

Epoch 1/80
75/75 [==============================] - 18s 27ms/step - loss: 1.3095 - accuracy: 0.6408
Epoch 2/80
75/75 [==============================] - 2s 23ms/step - loss: 0.4219 - accuracy: 0.8983
Epoch 3/80
75/75 [==============================] - 2s 21ms/step - loss: 0.1560 - accuracy: 0.9658
Epoch 4/80
75/75 [==============================] - 2s 21ms/step - loss: 0.1012 - accuracy: 0.9792
Epoch 5/80
75/75 [==============================] - 1s 18ms/step - loss: 0.0735 - accuracy: 0.9850
Epoch 6/80
75/75 [==============================] - 1s 18ms/step - loss: 0.1282 - accuracy: 0.9808
Epoch 7/80
75/75 [==============================] - 1s 18ms/step - loss: 0.1137 - accuracy: 0.9792
Epoch 8/80
75/75 [==============================] - 1s 16ms/step - loss: 0.0534 - accuracy: 0.9950
Epoch 9/80
75/75 [==============================] - 1s 16ms/step - loss: 0.0187 - accuracy: 0.9967
Epoch 10/80
75/75 [==============================] - 1s 16ms/step - loss: 0.0024 - accuracy: 1.0000
Epoch 11/80
75/75 [==============================] - 1s 16ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 12/80
75/75 [==============================] - 1s 16ms/step - loss: 7.9277e-04 - accuracy: 1.0000
Epoch 13/80
75/75 [==============================] - 1s 16ms/step - loss: 5.9054e-04 - accuracy: 1.0000
Epoch 14/80
75/75 [==============================] - 1s 16ms/step - loss: 4.7252e-04 - accuracy: 1.0000
Epoch 15/80
75/75 [==============================] - 1s 17ms/step - loss: 3.8069e-04 - accuracy: 1.0000

# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model1.evaluate(x_test)
toc=time.time()
test_time=toc-tic



print("----- #1 - orignal model-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

19/19 [==============================] - 4s 9ms/step - loss: 2.2276 - accuracy: 0.8567
----- #1 - orignal model-----

Training Time = 38.5 s
Testing Time = 3582.1 ms
Test Accuracy = 85.70 %:
```

the 2nd model

```
model2 = Sequential()
# Convolutional layer 1
model2.add(Conv2D(6, (5, 5), activation='relu', input_shape=(391, 257, 1), padding='valid'))
#dropout regularization
model2.add(Dropout(0.2))
```

```
# Average pooling layer 1
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model2.add(Conv2D(16, (5, 5), activation='relu', padding='valid'))
model2.add(Dropout(0.2))
# Average pooling layer 2
model2.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model2.add(Flatten())
# Fully connected layer 1
model2.add(Dense(120, activation='relu'))
model2.add(Dropout(0.2))
# Fully connected layer 2
model2.add(Dense(84, activation='relu'))
model2.add(Dropout(0.2))
# Output layer
model2.add(Dense(10, activation='softmax'))

model2.summary()

Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 387, 253, 6)       156

 dropout_12 (Dropout)        (None, 387, 253, 6)       0

 average_pooling2d_26 (Avera  (None, 193, 126, 6)      0
 gePooling2D)

 conv2d_27 (Conv2D)          (None, 189, 122, 16)      2416

 dropout_13 (Dropout)        (None, 189, 122, 16)      0

 average_pooling2d_27 (Avera  (None, 94, 61, 16)       0
 gePooling2D)

 flatten_13 (Flatten)        (None, 91744)             0

 dense_39 (Dense)            (None, 120)               11009400

 dropout_14 (Dropout)        (None, 120)               0

 dense_40 (Dense)            (None, 84)                10164

 dropout_15 (Dropout)        (None, 84)                0

 dense_41 (Dense)            (None, 10)                850

=================================================================
Total params: 11,022,986
Trainable params: 11,022,986
Non-trainable params: 0
_________________________________________________________________
```

```
#Early Stopping to avoid fitting issues
early_stopping = EarlyStopping(monitor='accuracy', patience=3)

model2.compile(loss= keras.losses.CategoricalCrossentropy(), optimizer='adam', metrics=['accuracy'])

# Train the model
tic=time.time()
```

```
model2.fit(x_train, epochs=80, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

Epoch 1/80
75/75 [==============================] - 4s 23ms/step - loss: 1.6791 - accuracy: 0.4958
Epoch 2/80
75/75 [==============================] - 2s 21ms/step - loss: 0.7131 - accuracy: 0.7817
Epoch 3/80
75/75 [==============================] - 2s 22ms/step - loss: 0.5013 - accuracy: 0.8533
Epoch 4/80
75/75 [==============================] - 2s 21ms/step - loss: 0.3383 - accuracy: 0.9192
Epoch 5/80
75/75 [==============================] - 2s 23ms/step - loss: 0.2278 - accuracy: 0.9350
Epoch 6/80
75/75 [==============================] - 2s 23ms/step - loss: 0.2231 - accuracy: 0.9408
Epoch 7/80
75/75 [==============================] - 2s 21ms/step - loss: 0.1004 - accuracy: 0.9742
Epoch 8/80
75/75 [==============================] - 2s 21ms/step - loss: 0.0944 - accuracy: 0.9733
Epoch 9/80
75/75 [==============================] - 2s 21ms/step - loss: 0.1767 - accuracy: 0.9600
Epoch 10/80
75/75 [==============================] - 2s 21ms/step - loss: 0.0773 - accuracy: 0.9733

# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model2.evaluate(x_test)
toc=time.time()
test_time=toc-tic



print("---- #2 the 2nd model-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

19/19 [==============================] - 0s 7ms/step - loss: 0.7661 - accuracy: 0.8633
---- #2 the 2nd model-----

Training Time = 20.2 s
Testing Time = 281.0 ms
Test Accuracy = 86.30 %:
```

the 3rd model

```
model3 = Sequential()
# Convolutional layer 1
model3.add(Conv2D(12, (5, 5), activation='relu', input_shape=(391, 257, 1), padding='same'))
# Average pooling layer 1
model3.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model3.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
# Average pooling layer 2
model3.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model3.add(Flatten())
# Fully connected layer 1
model3.add(Dense(120, activation='relu'))
# Fully connected layer 2
model3.add(Dense(84, activation='relu'))
# Output layer
model3.add(Dense(10, activation='softmax'))
```

```
model3.summary()

Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_28 (Conv2D)          (None, 391, 257, 12)      312       
                                                                 
 average_pooling2d_28 (Avera  (None, 195, 128, 12)     0         
 gePooling2D)                                                    
                                                                 
 conv2d_29 (Conv2D)          (None, 195, 128, 32)      9632      
                                                                 
 average_pooling2d_29 (Avera  (None, 97, 64, 32)       0         
 gePooling2D)                                                    
                                                                 
 flatten_14 (Flatten)        (None, 198656)            0         
                                                                 
 dense_42 (Dense)            (None, 120)               23838840  
                                                                 
 dense_43 (Dense)            (None, 84)                10164     
                                                                 
 dense_44 (Dense)            (None, 10)                850       
                                                                 
=================================================================
Total params: 23,859,798
Trainable params: 23,859,798
Non-trainable params: 0
_________________________________________________________________
```

```
#Early Stopping to avoid fitting issues
early_stopping = EarlyStopping(monitor='accuracy', patience=5)

model3.compile(loss=keras.losses.CategoricalCrossentropy(), optimizer='adam', metrics=['accuracy'])



# Train the model
tic=time.time()
model3.fit(x_train, epochs=80, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic


Epoch 1/80
75/75 [==============================] - 4s 28ms/step - loss: 1.3406 - accuracy: 0.6283
Epoch 2/80
75/75 [==============================] - 2s 25ms/step - loss: 0.2879 - accuracy: 0.9225
Epoch 3/80
75/75 [==============================] - 2s 24ms/step - loss: 0.2405 - accuracy: 0.9417
Epoch 4/80
75/75 [==============================] - 2s 24ms/step - loss: 0.1124 - accuracy: 0.9717
Epoch 5/80
75/75 [==============================] - 2s 24ms/step - loss: 0.1616 - accuracy: 0.9700
Epoch 6/80
75/75 [==============================] - 2s 25ms/step - loss: 0.0184 - accuracy: 0.9967
Epoch 7/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0040 - accuracy: 1.0000
Epoch 8/80
75/75 [==============================] - 2s 30ms/step - loss: 0.0016 - accuracy: 1.0000
Epoch 9/80
```

```
75/75 [==============================] - 2s 28ms/step - loss: 9.5883e-04 - accuracy: 1.0000
Epoch 10/80
75/75 [==============================] - 2s 24ms/step - loss: 6.4324e-04 - accuracy: 1.0000
Epoch 11/80
75/75 [==============================] - 2s 24ms/step - loss: 4.5650e-04 - accuracy: 1.0000
Epoch 12/80
75/75 [==============================] - 2s 24ms/step - loss: 3.5766e-04 - accuracy: 1.0000

# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model3.evaluate(x_test)
toc=time.time()
test_time=toc-tic



print("----- #3 - the 3rd model-----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

19/19 [==============================] - 0s 8ms/step - loss: 1.0015 - accuracy: 0.9100
----- #3 - the 3rd model-----

Training Time = 27.5 s
Testing Time = 439.6 ms
Test Accuracy = 91.00 %:
```

the 4th model

```
model4 = Sequential()
# Convolutional layer 1
model4.add(Conv2D(12, (5, 5), activation='tanh', input_shape=(391, 257, 1), padding='valid'))
# Average pooling layer 1
model4.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Convolutional layer 2
model4.add(Conv2D(32, (5, 5), activation='tanh', padding='valid'))
# Average pooling layer 2
model4.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
# Flatten layer
model4.add(Flatten())
# Fully connected layer 1
model4.add(Dense(120, activation='tanh'))
# Fully connected layer 2
model4.add(Dense(84, activation='tanh'))
# Output layer
model4.add(Dense(10, activation='softmax'))
model4.summary()


Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 387, 253, 12)      312

 average_pooling2d_30 (Avera  (None, 193, 126, 12)     0
 gePooling2D)

 conv2d_31 (Conv2D)          (None, 189, 122, 32)      9632

 average_pooling2d_31 (Avera  (None, 94, 61, 32)       0
 gePooling2D)
```

```
 flatten_15 (Flatten)        (None, 183488)            0

 dense_45 (Dense)            (None, 120)               22018680

 dense_46 (Dense)            (None, 84)                10164

 dense_47 (Dense)            (None, 10)                850

=================================================================
Total params: 22,039,638
Trainable params: 22,039,638
Non-trainable params: 0
_________________________________________________________________
```

```
early_stopping = EarlyStopping(monitor='accuracy', patience=5)

model4.compile(loss = keras.losses.CategoricalCrossentropy(), optimizer='adam', metrics=['accuracy'])

# Train the model
tic=time.time()
model4.fit(x_train, epochs=80, batch_size=32, callbacks=[early_stopping])
toc=time.time()
training_time=toc-tic

Epoch 1/80
75/75 [==============================] - 5s 27ms/step - loss: 1.2450 - accuracy: 0.6425
Epoch 2/80
75/75 [==============================] - 2s 26ms/step - loss: 0.2714 - accuracy: 0.9442
Epoch 3/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0994 - accuracy: 0.9850
Epoch 4/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0397 - accuracy: 0.9983
Epoch 5/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0149 - accuracy: 1.0000
Epoch 6/80
75/75 [==============================] - 2s 27ms/step - loss: 0.0085 - accuracy: 1.0000
Epoch 7/80
75/75 [==============================] - 2s 28ms/step - loss: 0.0063 - accuracy: 1.0000
Epoch 8/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0042 - accuracy: 1.0000
Epoch 10/80
75/75 [==============================] - 2s 26ms/step - loss: 0.0035 - accuracy: 1.0000


# Evaluate the model on the test data
tic=time.time()
test_loss, test_acc = model4.evaluate(x_test)
toc=time.time()
test_time=toc-tic

print("----- #4 - the 4th model -----\n")
print("Training Time = {} s".format(np.round(training_time, 1)))
print("Testing Time = {} ms".format(np.round(test_time*1000, 1)))
print('Test Accuracy = {:.2f} %:'.format(np.round(test_acc, 3)*100))

19/19 [==============================] - 0s 11ms/step - loss: 0.3012 - accuracy: 0.9067
----- #4 - the 4th model -----

Training Time = 23.6 s
Testing Time = 440.5 ms
Test Accuracy = 90.70 %:
```

Notes

Part 4				
	Variations	**Accuracy**	**Training Time**	**Testing Time**
CNN	1st model	85.7 %	38.4 s	3582.1 ms
	2nd model	86.30 %	20.2 s	281 ms
	3rd model	91 %	27.5 s	439.6 ms
	4th model	90.7 %	23.6 s	440.5 ms

For this problem we choose the following architecture that have the most accuracy: **91 %** using the dataset 'digits_audio_from0to10 'which have **120 speakers for training** set and **30 speakers for test set**.

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 391, 257, 12)      312

 average_pooling2d_28 (Avera  (None, 195, 128, 12)     0
 gePooling2D)

 conv2d_29 (Conv2D)          (None, 195, 128, 32)      9632

 average_pooling2d_29 (Avera  (None, 97, 64, 32)       0
 gePooling2D)

 flatten_14 (Flatten)        (None, 198656)            0

 dense_42 (Dense)            (None, 120)               23838840

 dense_43 (Dense)            (None, 84)                10164

 dense_44 (Dense)            (None, 10)                850

=================================================================
Total params: 23,859,798
Trainable params: 23,859,798
Non-trainable params: 0
_________________________________________________________________
```