|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 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 1](#_Toc132898401)

[Training a Multi-layer Perceptron (MLP) 3](#_Toc132898402)

[Using DCT Features 3](#_Toc132898403)

[Using PCA Features 5](#_Toc132898404)

[Using ICA Features 7](#_Toc132898405)

[2. Convolutional Neural Network (LeNet-5) 10](#_Toc132898406)

[LeNet-5 - No Variations 12](#_Toc132898407)

[Variation #1 - Adding Dropout Regularization 13](#_Toc132898408)

[Variation #2 - Increasing Number of Filters in Conv Layers 14](#_Toc132898409)

[Variation #3 - Adding "Same" Padding to Conv Layers 15](#_Toc132898410)

[Variation #4 - Using "Tanh" Activation 16](#_Toc132898411)

[3. Comparing the Results 17](#_Toc132898412)

[Notes 18](#_Toc132898413)

[4. Digit Spectrograms 19](#_Toc132898414)

# 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")

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

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **Features** | | | | | |
| **DCT** | | **PCA** | | **ICA** | |
| **Accuracy** | **Training Time** | **Accuracy** | **Training Time** | **Accuracy** | **Training Time** |
| **Classifier** | |
| **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