Diagram

Description automatically generated with medium confidence

**Homework 3: Keras and CNNs**

**Objectives:**

* Design a Dense Neural Network
* Implement CNN for image classification
* Utilizing Pre-Trained CNN models for required classification

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **ID** |
| Youssef Ghoneim | B00087523 |
| Mohamed Fawaz | B00087431 |

**Submitted to:**

Dr. Imran Zualkernan

Mr. Ali Reza Sajun

**Refer to the Keras and Convolutional Neural Network Slides for this Homework.**

**Note:** For this assignment you need to use the provided 3-class dataset (on ilearn as Dataset.zip) which classifies between images of cookies, dimsums and sushi.

**In your solution document, for each question please insert the code, a screenshot of the code running and report the required evaluation metrics for the 60-20-20 split and the k-fold cross validation.**

**Question 1: Designing a Dense Neural Network**

Design a dense neural network to classify your dataset into different classes. Your network should have the following specifications:

a) Input layer with an appropriate input size for the images.

b) At least two hidden layers with a sufficient number of neurons.

c) An output layer with the appropriate number of units for the classification task.

d) Use the ReLU activation function for the hidden layers and an appropriate activation function for the output layer.

e) Choose an appropriate loss function and optimization algorithm.

* Train your neural network using a 60-20-20 split on your chosen dataset and report the following metrics:
* Accuracy: percentage of correctly classified images.
* Loss: the value of the loss function after training.
* Precision, Recall, and F1-score for each class.
* ROC curve and AUC.
* Perform k-fold cross-validation (k = 5) and report the average and standard deviation of the metrics across all folds.

**Code below, forgot to take SS of code running for Q1 and Q2, but one is provided for Q3. See results below code:**

# -\*- coding: utf-8 -\*-

"""87523\_87431\_HW3\_Q1.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1AEXmvZgiJAe\_oEP\_AxCQ0fE-6nrDbcPG

"""

# Commented out IPython magic to ensure Python compatibility.

# %pip install tensorflow

from google.colab import drive

drive.mount('/content/drive')

import keras, os

import cv2

from sklearn.model\_selection import train\_test\_split

import numpy as np

img\_size = 128

def load\_data():

x = []

y = [] # 0 for Cookie, 1 for Dimsum, and 2 for Sushi

coo = os.listdir("drive/MyDrive/Dataset/Cookie")

dim = os.listdir("drive/MyDrive/Dataset/Dimsum")

sus = os.listdir("drive/MyDrive/Dataset/Sushi")

for pic in coo:

img\_path = os.path.join("drive/MyDrive/Dataset/Cookie", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(0)

else:

print(f"Image not loaded correctly: {img\_path}")

for pic in dim:

img\_path = os.path.join("drive/MyDrive/Dataset/Dimsum", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(1)

else:

print(f"Image not loaded correctly: {img\_path}")

for pic in sus:

img\_path = os.path.join("drive/MyDrive/Dataset/Sushi", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(2)

else:

print(f"Image not loaded correctly: {img\_path}")

return np.array(x), np.array(y)

data\_x, data\_y = load\_data()

print(data\_x.shape)

print(data\_y.shape)

dataset\_size = 3000

dataset\_indices = np.random.choice(np.arange(data\_x.shape[0]), size=dataset\_size, replace=True)

x\_dataset = data\_x[dataset\_indices]

y\_dataset = data\_y[dataset\_indices]

split\_train = int(0.6 \* dataset\_size)

split\_test = int(0.2 \* dataset\_size)

# split\_valid is not necessary as it can be calculated from the remaining indices.

indices = np.arange(x\_dataset.shape[0])

np.random.shuffle(indices)

train\_indices = indices[:split\_train]

test\_indices = indices[split\_train:split\_train + split\_test]

valid\_indices = indices[split\_train + split\_test:]

x\_train = x\_dataset[train\_indices].astype(np.float32)

y\_train = y\_dataset[train\_indices].astype(np.float32)

x\_test = x\_dataset[test\_indices].astype(np.float32)

y\_test = y\_dataset[test\_indices].astype(np.float32)

x\_valid = x\_dataset[valid\_indices].astype(np.float32)

y\_valid = y\_dataset[valid\_indices].astype(np.float32)

print(x\_train.shape)

print(x\_test.shape)

print(x\_valid.shape)

print(y\_train.shape)

print(y\_test.shape)

print(y\_valid.shape)

# Data normalization and reshaping for NN

# Normalize the image pixel values.

x\_train /= 127.5

x\_train -= 1.0

x\_test /= 127.5

x\_test -= 1.0

x\_valid /= 127.5

x\_valid -= 1.0

# Flatten the image dimensions

pixels = img\_size \* img\_size \* 3 # calculating total number of pixels in one image

x\_train = x\_train.reshape(-1, pixels)

x\_test = x\_test.reshape(-1, pixels)

x\_valid = x\_valid.reshape(-1, pixels)

print(x\_train.shape)

print(x\_test.shape)

print(x\_valid.shape)

# One hot encode the y data

from keras.utils import to\_categorical

y\_train = y\_train.reshape(-1,1)

y\_test = y\_test.reshape(-1,1)

y\_valid = y\_valid.reshape(-1,1)

#One hot encoding

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

y\_valid= to\_categorical(y\_valid)

print(y\_train.shape)

print(y\_test.shape)

print(y\_valid.shape)

from keras.models import Sequential, Model

from keras.layers import Dense

def makeModel():

model = Sequential()

model.add(Dense(64, input\_dim = pixels,activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(3, activation='softmax')) # because there are 3 possible classes

return model

model = makeModel()

model.summary()

from keras import backend as K

def F1\_Score(y\_true, y\_pred):

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

recall = true\_positives / (possible\_positives + K.epsilon())

f1\_val = 2\*(precision\*recall)/(precision+recall+K.epsilon())

return f1\_val

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy',F1\_Score],

)

history = model.fit(x\_train,y\_train,epochs=10,batch\_size=32,validation\_data=(x\_valid, y\_valid),verbose=1)

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend(['Training Accuracy','Validation Accuracy'])

plt.title("Model accuracy")

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend(['Training Loss','Validation Loss'])

plt.title("Loss")

plt.show()

plt.plot(history.history['F1\_Score'])

plt.plot(history.history['val\_F1\_Score'])

plt.xlabel("Epochs")

plt.ylabel("F1\_Score")

plt.legend(['Training F1\_Score','Validation F1\_Score'])

plt.title("F1\_Score")

plt.show()

score = model.evaluate(x\_test, y\_test)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

print('Test F1: ',score[2])

preds=model.predict(x\_test)

preds.shape

def generate\_actual\_predicted(model, X\_test, Y\_test):

# Get the predictions

Y\_pred = model.predict(X\_test)

# Create a list of predictions

#Converting predictions to label

predicted = list()

for i in range(len(Y\_pred)):

predicted.append(np.argmax(Y\_pred[i]))

#Converting one hot encoded test label back to label

actual = list()

for i in range(len(Y\_test)):

actual.append(np.argmax(Y\_test[i]))

return actual, predicted

from keras.optimizers import Adam

from sklearn.model\_selection import KFold

from sklearn.metrics import f1\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

import tensorflow as tf

n\_split=5

results = list()

for train\_index,test\_index in KFold(n\_split).split(x\_train):

# use the index to generate training an testing sets

kx\_train,kx\_test=x\_train[train\_index],x\_train[test\_index]

ky\_train,ky\_test=y\_train[train\_index],y\_train[test\_index]

# create and fit the model

model=makeModel()

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy',F1\_Score])

model.fit(kx\_train, ky\_train,epochs=10)

# print and show generic metrics available

scores = model.evaluate(kx\_test,ky\_test)

# print the model metrics

for i,names in zip(np.arange(0,len(model.metrics\_names)),model.metrics\_names):

print(model.metrics\_names[i],'=',scores[i])

# calculate and print more metrics

actual, predicted = generate\_actual\_predicted(model, kx\_test, ky\_test)

print("precision ",precision\_score(actual, predicted, average='macro'))

print("recall ", recall\_score(actual, predicted, average='macro'))

print("F1-Score ", f1\_score(actual, predicted, average='macro'))

results.append([precision\_score(actual, predicted, average='macro'),

recall\_score(actual, predicted, average='macro'),

f1\_score(actual, predicted, average='macro') ])

from sklearn.metrics import classification\_report

print(classification\_report(actual, predicted))

# Plot the curves for precision

res = np.array(results)

#print(res)

prec = res[:,0]

print(prec)

recl = res[:,1]

print(recl)

import matplotlib.pyplot as plt

plt.scatter(prec, recl)

plt.xlabel('precision')

plt.ylabel('recall')

plt.title('K-Fold')

plt.show()

print('min precision:', prec.min())

print('min recall:', recl.min())

print('max precision:', prec.max())

print('max recall:', recl.max())

print('average precision:', prec.mean(), 'sd=', prec.std())

print('average recall:', recl.mean(),'sd=', recl.std())

# show f1-score for k-fold

res = np.array(results)

f1\_score = res[:,2]

plt.hist(x=f1\_score, bins='auto', color='#0504aa',

alpha=0.7, rwidth=0.85)

plt.title('K-Fold F1-Score')

plt.ylabel('Frequency')

plt.xlabel('F1-Score')

plt.show()

print('F1-Score:', f1\_score.mean(), 'sd=', f1\_score.std())

res\_model1 = res

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = y\_test.shape[1] # one hot encoded

y\_score=model.predict(x\_test, batch\_size=None, verbose=0, steps=None)

# compare each class's probabilities one by one

# each acts like a single column

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:,i], y\_score[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Print the AUC scores

from IPython.display import display

import pandas as pd

auc\_array = np.array(list(roc\_auc.items()))

df = pd.DataFrame(auc\_array[:,1])

df.columns = ['AUC']

display(df)

# Draw the pairwise ROC curves

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], lw=2,label='ROC curve of class {0} (area = {1:0.3f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

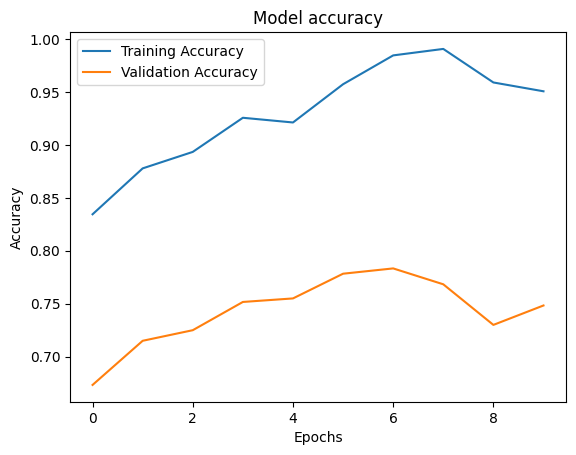
plt.title('ROC')

plt.legend(loc="lower right")

plt.tight\_layout()

plt.savefig('ROC\_Dense.png')

plt.show()



A graph with orange lines and blue lines

Description automatically generated

A graph of a number of people

Description automatically generated with medium confidence

A graph with blue dots

Description automatically generated

A graph with numbers and a number of blue squares

Description automatically generated with medium confidence

A graph of a curve

Description automatically generated

**Question 2: Convolutional Neural Network for Image Classification**

Design a Convolutional Neural Network (CNN) to classify the images in your dataset into their respective classes. Your CNN should have the following architecture:

a) Input layer for the images.

b) A sequence of Convolutional layers with appropriate kernel sizes, strides, and activation functions. You can also include pooling layers if needed.

c) At least two fully connected (dense) layers with an appropriate number of neurons.

d) Use the appropriate activation function for the fully connected layers.

e) Choose an appropriate loss function and optimization algorithm.

* Train your neural network using a 60-20-20 split on your chosen dataset and report the following metrics:
* Accuracy: percentage of correctly classified images.
* Loss: the value of the loss function after training.
* Precision, Recall, and F1-score for each class.
* ROC curve and AUC.
* Perform k-fold cross-validation (k = 5) and report the average and standard deviation of the metrics across all folds.

**Code:**

# -\*- coding: utf-8 -\*-

"""87523\_87431\_HW3\_Q2.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1yhMcmFS5lMsJ00okJKWEyCFC52nTEojC

"""

from google.colab import drive

drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.

# %pip install tensorflow

import keras, os

import cv2

from sklearn.model\_selection import train\_test\_split

import numpy as np

img\_size = 128

def load\_data():

x = []

y = [] # 0 for Cookie, 1 for Dimsum, and 2 for Sushi

coo = os.listdir("drive/MyDrive/Dataset/Cookie")

dim = os.listdir("drive/MyDrive/Dataset/Dimsum")

sus = os.listdir("drive/MyDrive/Dataset/Sushi")

for pic in coo:

img\_path = os.path.join("drive/MyDrive/Dataset/Cookie", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(0)

else:

print(f"Image not loaded correctly: {img\_path}")

for pic in dim:

img\_path = os.path.join("drive/MyDrive/Dataset/Dimsum", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(1)

else:

print(f"Image not loaded correctly: {img\_path}")

for pic in sus:

img\_path = os.path.join("drive/MyDrive/Dataset/Sushi", pic)

temp = cv2.imread(img\_path)

if temp is not None:

temp = temp.astype('float32')

temp = cv2.resize(temp, (img\_size, img\_size))

x.append(temp)

y.append(2)

else:

print(f"Image not loaded correctly: {img\_path}")

return np.array(x), np.array(y)

data\_x, data\_y = load\_data()

print(data\_x.shape)

print(data\_y.shape)

dataset\_size = 3000

dataset\_indices = np.random.choice(np.arange(data\_x.shape[0]), size=dataset\_size, replace=True)

x\_dataset = data\_x[dataset\_indices]

y\_dataset = data\_y[dataset\_indices]

split\_train = int(0.6 \* dataset\_size)

split\_test = int(0.2 \* dataset\_size)

# split\_valid is not necessary as it can be calculated from the remaining indices.

indices = np.arange(x\_dataset.shape[0])

np.random.shuffle(indices)

train\_indices = indices[:split\_train]

test\_indices = indices[split\_train:split\_train + split\_test]

valid\_indices = indices[split\_train + split\_test:]

x\_train = x\_dataset[train\_indices].astype(np.float32)

y\_train = y\_dataset[train\_indices].astype(np.float32)

x\_test = x\_dataset[test\_indices].astype(np.float32)

y\_test = y\_dataset[test\_indices].astype(np.float32)

x\_valid = x\_dataset[valid\_indices].astype(np.float32)

y\_valid = y\_dataset[valid\_indices].astype(np.float32)

print(x\_train.shape)

print(x\_test.shape)

print(x\_valid.shape)

print(y\_train.shape)

print(y\_test.shape)

print(y\_valid.shape)

# Normalize the image pixel values.

x\_train /= 127.5

x\_train -= 1.0

x\_test /= 127.5

x\_test -= 1.0

x\_valid /= 127.5

x\_valid -= 1.0

# One hot encode the y data

from keras.utils import to\_categorical

y\_train = y\_train.reshape(-1,1)

y\_test = y\_test.reshape(-1,1)

y\_valid = y\_valid.reshape(-1,1)

#One hot encoding

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

y\_valid= to\_categorical(y\_valid)

from keras.layers import Conv2D, Dropout,Dense,MaxPool2D, Flatten,BatchNormalization,Input

from keras.models import Model, Sequential

def makeModel():

model = Sequential()

model.add(Conv2D(32,(4,4),padding='same',activation='relu',input\_shape=(128,128,3),strides=1))

model.add(MaxPool2D(2,2))

model.add(Conv2D(64,(3,3),padding='same',activation='relu',strides=1))

model.add(MaxPool2D(2,2))

model.add(Conv2D(128,(2,2),padding='same',activation='relu',strides=1))

model.add(MaxPool2D(2,2))

model.add(Flatten())

model.add(Dense(1024,activation='relu'))

model.add(Dense(3,activation='softmax'))# 3 outputs

return model

model = makeModel()

model.summary()

from keras import backend as K

def F1\_Score(y\_true, y\_pred):

true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

precision = true\_positives / (predicted\_positives + K.epsilon())

recall = true\_positives / (possible\_positives + K.epsilon())

f1\_val = 2\*(precision\*recall)/(precision+recall+K.epsilon())

return f1\_val

from keras.optimizers import Adam

model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy',F1\_Score])

history = model.fit(x\_train,y\_train,epochs=10,batch\_size=32,validation\_data=(x\_valid, y\_valid),verbose=1)

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend(['Training Accuracy','Validation Accuracy'])

plt.title("Model accuracy")

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend(['Training Loss','Validation Loss'])

plt.title("Loss")

plt.show()

plt.plot(history.history['F1\_Score'])

plt.plot(history.history['val\_F1\_Score'])

plt.xlabel("Epochs")

plt.ylabel("F1\_Score")

plt.legend(['Training F1\_Score','Validation F1\_Score'])

plt.title("F1\_Score")

plt.show()

score = model.evaluate(x\_test, y\_test)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

print('Test F1: ',score[2])

preds=model.predict(x\_test)

preds.shape

def generate\_actual\_predicted(model, X\_test, Y\_test):

# Get the predictions

Y\_pred = model.predict(X\_test)

# Create a list of predictions

#Converting predictions to label

predicted = list()

for i in range(len(Y\_pred)):

predicted.append(np.argmax(Y\_pred[i]))

#Converting one hot encoded test label back to label

actual = list()

for i in range(len(Y\_test)):

actual.append(np.argmax(Y\_test[i]))

return actual, predicted

from keras.optimizers import Adam

from sklearn.model\_selection import KFold

from sklearn.metrics import f1\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

import tensorflow as tf

#k = 5 split is taking too long to run, so doing 3 instead :)

n\_split=3

results = list()

for train\_index,test\_index in KFold(n\_split).split(x\_train):

# use the index to generate training an testing sets

kx\_train,kx\_test=x\_train[train\_index],x\_train[test\_index]

ky\_train,ky\_test=y\_train[train\_index],y\_train[test\_index]

# create and fit the model

model=makeModel()

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy',F1\_Score])

model.fit(kx\_train, ky\_train,epochs=10)

# print and show generic metrics available

scores = model.evaluate(kx\_test,ky\_test)

# print the model metrics

for i,names in zip(np.arange(0,len(model.metrics\_names)),model.metrics\_names):

print(model.metrics\_names[i],'=',scores[i])

# calculate and print more metrics

actual, predicted = generate\_actual\_predicted(model, kx\_test, ky\_test)

print("precision ",precision\_score(actual, predicted, average='macro'))

print("recall ", recall\_score(actual, predicted, average='macro'))

print("F1-Score ", f1\_score(actual, predicted, average='macro'))

results.append([precision\_score(actual, predicted, average='macro'),

recall\_score(actual, predicted, average='macro'),

f1\_score(actual, predicted, average='macro') ])

from sklearn.metrics import classification\_report

print(classification\_report(actual, predicted))

# Plot the curves for precision

res = np.array(results)

#print(res)

prec = res[:,0]

print(prec)

recl = res[:,1]

print(recl)

import matplotlib.pyplot as plt

plt.scatter(prec, recl)

plt.xlabel('precision')

plt.ylabel('recall')

plt.title('K-Fold')

plt.show()

print('min precision:', prec.min())

print('min recall:', recl.min())

print('max precision:', prec.max())

print('max recall:', recl.max())

print('average precision:', prec.mean(), 'sd=', prec.std())

print('average recall:', recl.mean(),'sd=', recl.std())

# show f1-score for k-fold

res = np.array(results)

f1\_score = res[:,2]

plt.hist(x=f1\_score, bins='auto', color='#0504aa',

alpha=0.7, rwidth=0.85)

plt.title('K-Fold F1-Score')

plt.ylabel('Frequency')

plt.xlabel('F1-Score')

plt.show()

print('F1-Score:', f1\_score.mean(), 'sd=', f1\_score.std())

res\_model1 = res

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = y\_test.shape[1] # one hot encoded

y\_score=model.predict(x\_test, batch\_size=None, verbose=0, steps=None)

# compare each class's probabilities one by one

# each acts like a single column

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test[:,i], y\_score[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Print the AUC scores

from IPython.display import display

import pandas as pd

auc\_array = np.array(list(roc\_auc.items()))

df = pd.DataFrame(auc\_array[:,1])

df.columns = ['AUC']

display(df)

# Draw the pairwise ROC curves

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], lw=2,label='ROC curve of class {0} (area = {1:0.3f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

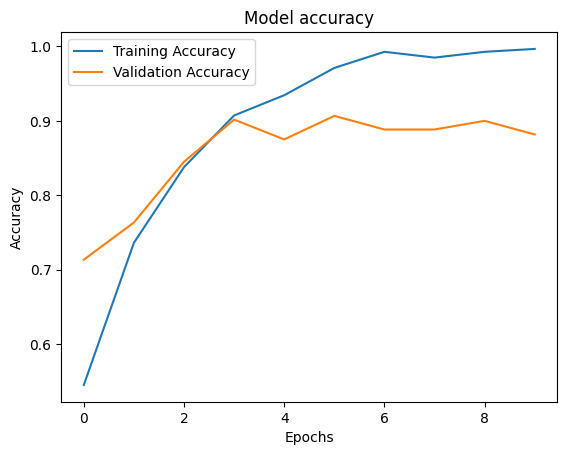
plt.title('ROC')

plt.legend(loc="lower right")

plt.tight\_layout()

plt.savefig('ROC\_CNN.png')

plt.show()



A graph of loss and validation

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and a number of points

Description automatically generated with medium confidence

A graph with numbers and a number of blue squares

Description automatically generated with medium confidence

A graph of a curve

Description automatically generated

**Question 3: Using ImageNet Pretrained CNNs**

Instead of designing a CNN from scratch, you want to leverage the knowledge from large-scale datasets like ImageNet. Choose a pre-trained CNN and follow these steps:

a) Load the pre-trained CNN without the top classification layer.

b) Add a new classification head on top of the pre-trained network to match the number of classes in your dataset.

c) Freeze the weights of the pre-trained layers and only train the newly added classification head.

* Train your neural network using a 60-20-20 split on your chosen dataset and report the following metrics:
* Accuracy: percentage of correctly classified images.
* Loss: the value of the loss function after training.
* Precision, Recall, and F1-score for each class.
* ROC curve and AUC.
* Perform k-fold cross-validation (k = 5) and report the average and standard deviation of the metrics across all folds.

**Code:**

# -\*- coding: utf-8 -\*-

"""87523\_87431\_HW3\_Q3.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1Fog6AfCl5hwdF-tT151SyjI1Cb0MfXUg

"""

from google.colab import drive

drive.mount('/content/drive')

import numpy as np

import keras

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from keras.utils import to\_categorical

dataset = tf.keras.utils.image\_dataset\_from\_directory(directory='drive/MyDrive/Dataset', seed=42, image\_size=(96, 96), batch\_size=None)

X = []

Y = []

for image, label in dataset:

X.append(image)

Y.append(label)

X = np.array(X)

Y = np.array(Y)

print(X[0].shape)

print(Y[0].shape)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 42)

print(X\_train.shape)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255

Y\_train = to\_categorical(Y\_train, 3)

Y\_test = to\_categorical(Y\_test, 3)

x\_train = X\_train.reshape(X\_train.shape[0], 96,96,3)

x\_test = X\_test.reshape(X\_test.shape[0], 96,96,3)

print(X\_train.shape)

print(Y\_train.shape)

print(X\_train.shape[0])

import pandas as pd

import numpy as np

import os

import keras

import matplotlib.pyplot as plt

from keras.layers import Dense,GlobalAveragePooling2D

from keras.applications import MobileNet

from keras.preprocessing import image

from keras.applications.mobilenet import preprocess\_input

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Model

from keras.optimizers import Adam

EPOCHS = 10

CLASSES = 3

TENSORBOARD = False

BATCH\_SIZE = 32

FREEZE\_NR\_LAYERS = -1

IMG\_SHAPE = (96, 96)

SHUFFLE\_BUFFER\_SIZE = 1000

CHECKPOINTS = False

SIZE\_DATA = 3000

net = 'MobileNet\_25'

print('Training %s ...' % net)

if net == 'VGG16':

base\_model = VGG16(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg')

elif net == 'ResNet50':

base\_model = ResNet50(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg')

elif net == 'InceptionV3':

# original size: 299x299

base\_model = InceptionV3(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg')

elif net == 'MobileNet\_100':

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg',

alpha=1.0, depth\_multiplier=1, dropout=.2)

elif net == 'MobileNet\_75':

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg',

alpha=.75, depth\_multiplier=1, dropout=.2)

elif net == 'MobileNet\_50':

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg',

alpha=.5, depth\_multiplier=1, dropout=.2)

elif net == 'MobileNet\_25':

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=IMG\_SHAPE + (3,), pooling='avg',

alpha=.25, depth\_multiplier=1, dropout=.2)

elif net == 'DenseNet121':

tf\_weights = 'cnn\_finetune/imagenet\_models/densenet121\_weights\_tf.h5'

base\_model = densenet121\_model(dim, dim, 3, dropout\_rate=0.2, weights\_path=tf\_weights)

else:

print('Unknown model, will train MobileNet instead.')

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=(dim, dim, 3), pooling='avg',

alpha=.5, depth\_multiplier=1, dropout=.2)

x\_top = base\_model.output

x\_out = Dense(3, name='output', activation='softmax')(x\_top)

model = Model(base\_model.input, x\_out)

model.summary()

base\_model.save("base\_model")

for layer in base\_model.layers:

layer.trainable = False

model.compile(optimizer=Adam(learning\_rate=0.001),

loss='categorical\_crossentropy', metrics = ['accuracy'])

from keras.callbacks import ModelCheckpoint

# save the best model

filepath='best\_model'

checkpoint = ModelCheckpoint(filepath, monitor='val\_accuracy', verbose=1, save\_best\_only=True, mode='max')

callbacks\_list = [checkpoint]

# Fit the model

hist = model.fit(X\_train, Y\_train,

epochs=30,

validation\_split=0.2,

steps\_per\_epoch=2,

validation\_steps=2,

callbacks = callbacks\_list)

test\_score = model.evaluate(X\_test, Y\_test)

print(test\_score[0], test\_score[1] \* 100)

def generate\_actual\_predicted(model, X\_test, Y\_test):

# Get the predictions

Y\_pred = model.predict(X\_test)

# Create a list of predictions

#Converting predictions to label

predicted = list()

for i in range(len(Y\_pred)):

predicted.append(np.argmax(Y\_pred[i]))

#Converting one hot encoded test label back to label

actual = list()

for i in range(len(Y\_test)):

actual.append(np.argmax(Y\_test[i]))

return actual, predicted

actual, predicted = generate\_actual\_predicted(model, X\_test, Y\_test)

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(actual,predicted)

print(cm)

from sklearn.metrics import classification\_report

print(classification\_report(actual, predicted))

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = Y\_test.shape[1]

Y\_score=model.predict(X\_test, batch\_size=None, verbose=0, steps=None)

# compare each class's probabilities one by one

# each acts like a single column

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(Y\_test[:,i], Y\_score[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Print the AUC scores

from IPython.display import display

import pandas as pd

auc\_array = np.array(list(roc\_auc.items()))

df = pd.DataFrame(auc\_array[:,1])

df.columns = ['AUC']

display(df)

# Draw the pairwise ROC curves

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], lw=2,label='ROC curve of class {0} (area = {1:0.3f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC')

plt.legend(loc="lower right")

plt.tight\_layout()

plt.savefig('ROC\_CNN.png')

plt.show()

plt.plot(hist.history['loss'])

plt.plot(hist.history['val\_loss'])

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend(['Training Loss','Validation Loss'])

plt.title("Loss")

plt.show()

plt.plot(hist.history['accuracy'])

plt.plot(hist.history['val\_accuracy'])

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend(['Training Accuracy','Validation Accuracy'])

plt.title("Model accuracy")

plt.show()

import tensorflow

base\_model = MobileNet(include\_top=False, weights='imagenet', input\_shape=IMG\_SHAPE + (3,), pooling='avg', alpha=.25, depth\_multiplier=1, dropout=.2)

for layer in base\_model.layers:

layer.trainable = False

def create\_model(base\_model):

x\_top = base\_model.output

x\_out = Dense(3, name='output', activation='softmax')(x\_top)

model = Model(base\_model.input, x\_out)

model.summary()

model.save("base\_model")

model.compile(optimizer=Adam(learning\_rate=0.001),

loss='categorical\_crossentropy', metrics = ['accuracy'])

return model

from sklearn.model\_selection import KFold

from sklearn.metrics import f1\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

import tensorflow as tf

n\_split=5

results = list()

for train\_index,test\_index in KFold(n\_split).split(X\_train):

x\_train,x\_test=X\_train[train\_index],X\_train[test\_index]

y\_train,y\_test=Y\_train[train\_index],Y\_train[test\_index]

model=create\_model(base\_model)

model.fit(x\_train, y\_train,epochs=30)

scores = model.evaluate(x\_test,y\_test)

for i,names in zip(np.arange(0,len(model.metrics\_names)),model.metrics\_names):

print(model.metrics\_names[i],'=',scores[i])

actual, predicted = generate\_actual\_predicted(model, x\_test, y\_test)

print("precision ",precision\_score(actual, predicted, average='macro'))

print("recall ", recall\_score(actual, predicted, average='macro'))

print("F1-Score ", f1\_score(actual, predicted, average='macro'))

results.append([precision\_score(actual, predicted, average='macro'),

recall\_score(actual, predicted, average='macro'),

f1\_score(actual, predicted, average='macro') ])

# Plot the curves for precision

res = np.array(results)

#print(res)

prec = res[:,0]

print(prec)

recl = res[:,1]

print(recl)

import matplotlib.pyplot as plt

plt.scatter(prec, recl)

plt.xlabel('precision')

plt.ylabel('recall')

plt.title('K-Fold')

plt.show()

print('min precision:', prec.min())

print('min recall:', recl.min())

print('max precision:', prec.max())

print('max recall:', recl.max())

print('average precision:', prec.mean(), 'sd=', prec.std())

print('average recall:', recl.mean(),'sd=', recl.std())

# show f1-score for k-fold

res = np.array(results)

f1\_score = res[:,2]

plt.hist(x=f1\_score, bins='auto', color='#0504aa',

alpha=0.7, rwidth=0.85)

plt.title('K-Fold F1-Score')

plt.ylabel('Frequency')

plt.xlabel('F1-Score')

plt.show()

print('F1-Score:', f1\_score.mean(), 'sd=', f1\_score.std())

res\_model1 = res

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes = Y\_test.shape[1]

Y\_score=model.predict(X\_test, batch\_size=None, verbose=0, steps=None)

# compare each class's probabilities one by one

# each acts like a single column

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(Y\_test[:,i], Y\_score[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Print the AUC scores

from IPython.display import display

import pandas as pd

auc\_array = np.array(list(roc\_auc.items()))

df = pd.DataFrame(auc\_array[:,1])

df.columns = ['AUC']

display(df)

# Draw the pairwise ROC curves

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], lw=2,label='ROC curve of class {0} (area = {1:0.3f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

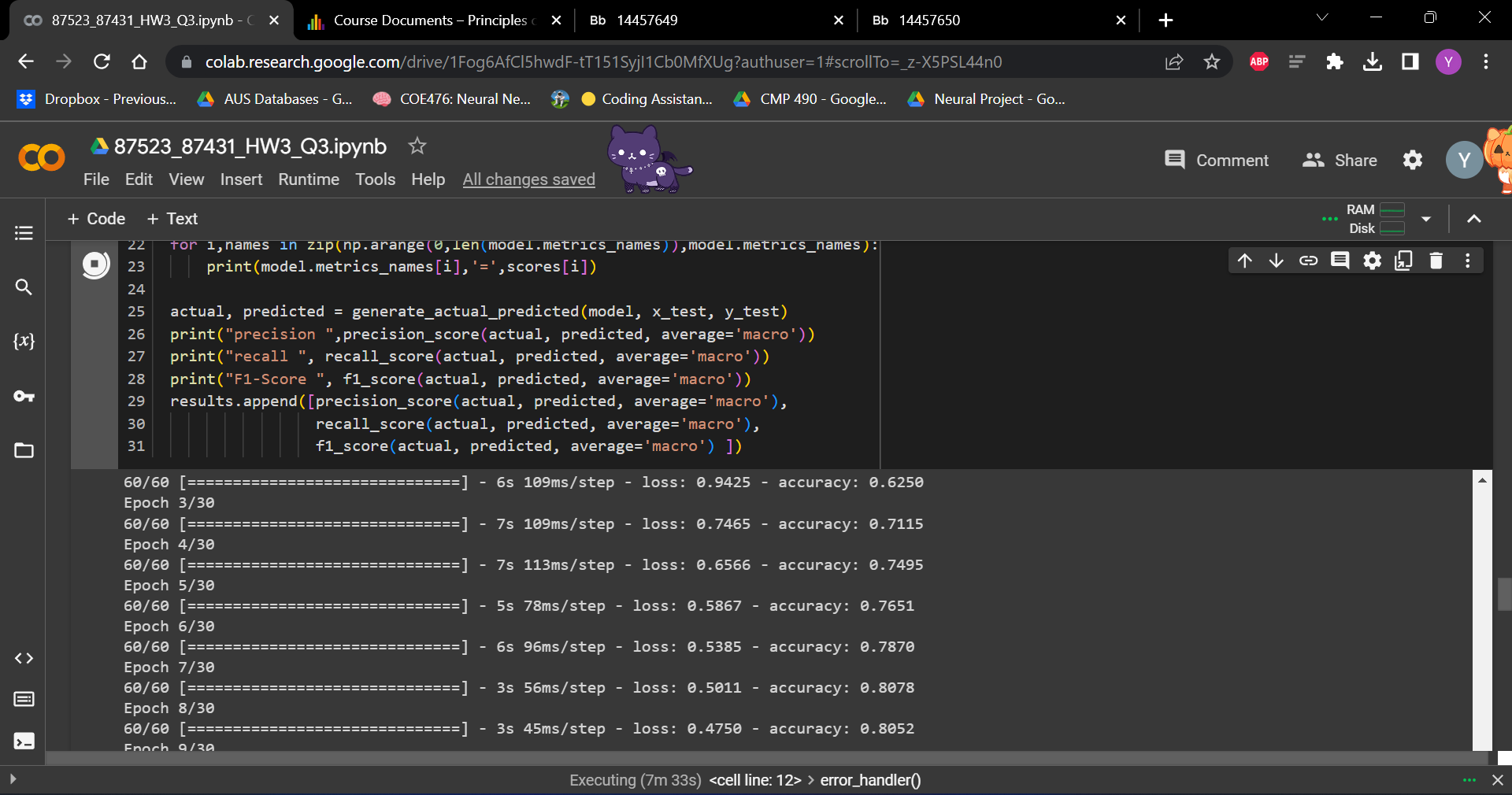
plt.title('ROC')

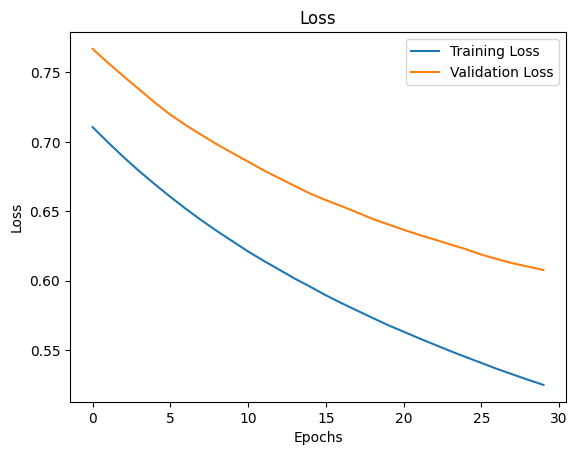
plt.legend(loc="lower right")

plt.tight\_layout()

plt.savefig('ROC\_CNN.png')

plt.show()

****



A graph of a graph showing a line

Description automatically generated

A graph with blue dots

Description automatically generated

A graph of a number of blue bars

Description automatically generated

A graph of a curve

Description automatically generated