```
# Computer Vision Coursework Submission (INM460)

# **Student name, ID and cohort:** Sam Clastine Jesumuthu (220038747) - PG

# ****************** /code/training_cv.ipynb **********************

# -*- coding: utf-8 -*-
"""training_cv.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1tYurtYZECtimLx8lyIGYm58OmZeKpq7O
"""

!pip install imblearn

!pip install -q -U keras-tuner

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

"""# Importing Libraries"""

import matplotlib.pyplot as plt
import numpy as np
import sys
from PIL import Image
sys.modules['Image'] = Image
import tensorflow as tf


from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Input, BatchNormalization, Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
from tensorflow.keras.models import Model
import tensorflow as tf
import keras_tuner as kt

from keras.utils.data_utils import Sequence
from imblearn.over_sampling import RandomOverSampler
```

```
from imblearn.tensorflow import balanced_batch_generator

from sklearn.model_selection import train_test_split
from skimage.feature import hog
import skimage
from sklearn import svm
from sklearn.metrics import accuracy_score
from matplotlib import patches
from collections import Counter



import pathlib
import os
import glob
import pandas as pd
import cv2 as cv
import numpy as np
# import face_recognition
from google.colab.patches import cv2_imshow
from PIL import Image

"""# Loading, Splitting and Preprocessing the Dataset"""

train_path="/content/drive/MyDrive/Computer-vision/cv-coursework/CW_Dataset/train"

#  extract and return the first line of a text file as an integer
def txtFileExtract(path):
    f=open(path,'r')
    content=f.readlines()
    return content[0]

# This function loads a dataset from a given path and reads the targets from .txt
files in the subdirectories of the given path
# and stores them in a list, then finds all the .jpeg images in the subdirectories
and returns their paths along with
# the targets list.
def load_dataset(path):
  targets = []
  labels=os.listdir(path)
  _image_path=glob.glob(path+'/*/*.jpeg')
  txt_path=glob.glob(path+'/*/*.txt')
  targets = [txtFileExtract(p) for p in txt_path]
  return _image_path, targets

# storing the image path and labels in a variable
im_path, labels = load_dataset(train_path)

# changing the datatype of labels to float32
labels = np.array(labels).astype('float32')
```

```
# check the length of each classes
len(np.where(labels == 1.0)[0])

# saving the labels as y_train.npy for reusability
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/y_train.npy
',labels)

# loads images from file paths and preprocesses
def load_image(img_path):
  _images = []
  for image_path in img_path:
    image = cv.imread(image_path)
    image = cv.resize(image, (64,64))
    image = cv.normalize(image, None, 0, 1.0, cv.NORM_MINMAX, dtype=cv.CV_32F)
    _images.append(image)
  return _images

#this function takes argument all image path and retun hog converted images as an
array
def load_image_to_hog(img_path):
  _images = []
  winStride = (8,8)
  padding = (8,8)
  locations = ((10,20),)
  HOG = cv.HOGDescriptor()
  for image_path in img_path:
    image = cv.imread(image_path)
    image = cv.resize(image, (64,64))
    fd, image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, channel_axis = -1)
    image = skimage.color.gray2rgb(image)
    _images.append(image)
  return _images

# applying the fuction to convert path to original images and hog images as list of
arrays
or_images = load_image(im_path)
hog_images = load_image_to_hog(im_path)

# converting the list to numpy array
or_images = np.array(or_images)
hog_images = np.array(hog_images)

# saving the numpy arrays
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/hog_images_
train.npy',hog_images)
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/or_images_t
rain.npy',or_images)
```

```
# loading the hog and original images which can be said as x_train

hog_images =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/hog_images_
train.npy')
or_images =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/or_images_t
rain.npy')

#
https://www.linkedin.com/pulse/some-tricks-handling-imbalanced-dataset-image-m-farh
an-tandia/
datagen = ImageDataGenerator(
    rotation_range=10,  # rotate the image by a maximum of 10 degrees
    width_shift_range=0.1,  # shift the image horizontally by a maximum of 10%
    height_shift_range=0.1,  # shift the image vertically by a maximum of 10%
    shear_range=0.2,  # shear the image by a maximum of 20%
    zoom_range=0.2,  # zoom into the image by a maximum of 20%
    horizontal_flip=True,  # flip the image horizontally
    vertical_flip=False  # do not flip the image vertically
)

# this method creates and balnced dataset with image aggumentation technique
applied for a sample of image the library used for balancing are keras and imblearn
#https://www.linkedin.com/pulse/some-tricks-handling-imbalanced-dataset-image-m-far
han-tandia/
class BalancedDataGenerator(Sequence):
  def __init__(self, x, y, datagen, batch_size=64):
      self.datagen = datagen
      self.batch_size = min(batch_size, x.shape[0])
      datagen.fit(x)
      self.gen, self.steps_per_epoch =
balanced_batch_generator(x.reshape(x.shape[0], -1), y, sampler=RandomOverSampler(),
batch_size=self.batch_size, keep_sparse=True)
      self._shape = (self.steps_per_epoch * batch_size, *x.shape[1:])

  def __len__(self):
      return self.steps_per_epoch


  def __getitem__(self, idx):
      x_batch, y_batch = self.gen.__next__()
      x_batch = x_batch.reshape(-1, *self._shape[1:])
      return self.datagen.flow(x_batch, y_batch, batch_size=self.batch_size).next()

#splitting traing and validation set which are 80% and 20% respectively
x_train, x_val, y_train, y_val = train_test_split(or_images, labels, test_size =
0.2, random_state=42, stratify=labels, shuffle=True)
x_train_hog, x_val_hog, y_train_hog, y_val_hog = train_test_split(hog_images,
labels, test_size = 0.2, random_state=42, stratify=labels, shuffle=True)
```

```
# balancing the splitted datset and applying batches for train and validation set
balanced_gen = BalancedDataGenerator(x_train, y_train, datagen, batch_size=64)
balanced_gen_val = BalancedDataGenerator(x_val, y_val, datagen, batch_size=64)

# balancing the splitted datset and applying batches for train and validation set
for HOG images
balanced_gen_hog = BalancedDataGenerator(x_train_hog, y_train_hog, datagen,
batch_size=64)
balanced_gen_val_hog = BalancedDataGenerator(x_val_hog, y_val_hog, datagen,
batch_size=64)

# extracting x and y values from the generator
x, y = [], []
for i in range(len(balanced_gen)):
    batch_x, batch_y = balanced_gen[i]
    x.append(batch_x)
    y.append(batch_y)
x = np.concatenate(x)
y = np.concatenate(y)


# ploting the frequency for Balanced dataset
plt.hist(y)
plt.xlabel('integer labels')
plt.ylabel('frequency')
plt.show()

counter = Counter(labels)
counts = list(counter.values())
categories = list(counter.keys())

# ploting the frequency for imbalanced train dataset
plt.bar(categories, counts)
# add labels
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.title('Train set data frequency')

# show plot
plt.show()

counter = Counter(y_val)
counts = list(counter.values())
categories = list(counter.keys())

# ploting the frequency for imbalanced test dataset

plt.bar(categories, counts)
# add labels
```

```
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.title('Test set data frequency')

# show plot
plt.show()

# it converts the xtrain and y_train arrays into tensor dataset

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1000,
reshuffle_each_iteration=True)
val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val))
val_dataset = val_dataset.shuffle(buffer_size=1000, reshuffle_each_iteration=True)

# it converts the xtrain and y_train arrays into tensor hog dataset
train_dataset_hog = tf.data.Dataset.from_tensor_slices((x_train_hog, y_train_hog))
train_dataset_hog = train_dataset_hog.shuffle(buffer_size=1000,
reshuffle_each_iteration=True)
val_dataset_hog = tf.data.Dataset.from_tensor_slices((x_val_hog, y_val_hog))
val_dataset_hog = val_dataset_hog.shuffle(buffer_size=1000,
reshuffle_each_iteration=True)

# apply batches for all datasets
train_dataset = train_dataset.batch(64)
val_dataset = val_dataset.batch(64)
train_dataset_hog = train_dataset_hog.batch(64)
val_dataset_hog = val_dataset_hog.batch(64)

"""# MLP with Hog Feature Descriptor

---


"""

# https://www.tensorflow.org/tutorials/keras/keras_tuner

# this function creates MLP architecture with hyperparameter optimization
def model_mlp_ar(hp):
  # initializing the dropout value from range 0 to 1
  hp_rate = hp.Float('rate', min_value=0, max_value=1.0, step=0.25)
  # initializing the input neurons from range 32 to 512
  hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
  # initializing the keras sequential layers
  model_mlp = Sequential()
  #flattening the input tensor
  model_mlp.add(Flatten())
  # Adding dense layer with batch normalization and dropouts
  model_mlp.add(Dense(units = hp_units, activation='relu'))
```

```
  model_mlp.add(BatchNormalization())
  model_mlp.add(Dropout(rate=hp_rate))

  model_mlp.add(Dense(units = hp_units, activation='sigmoid'))
  model_mlp.add(BatchNormalization())
  model_mlp.add(Dropout(rate=hp_rate))

  model_mlp.add(Dense(units = hp_units, activation='sigmoid'))
  model_mlp.add(BatchNormalization())
  model_mlp.add(Dropout(rate=hp_rate))

  model_mlp.add(Dense(units = hp_units, activation='sigmoid'))
  model_mlp.add(BatchNormalization())
  model_mlp.add(Dropout(rate=hp_rate))

  model_mlp.add(Dense(units = hp_units, activation='sigmoid'))
  model_mlp.add(BatchNormalization())
  model_mlp.add(Dropout(rate=hp_rate))

  model_mlp.add(Dense(3, activation='sigmoid'))
  # initializing the learning rate
  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
  Apply optimization function to the model

model_mlp.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

  return model_mlp

# initializing a Hyperband tuner from Keras Tuner for hyperparameter optimization
for imbalanced dataset
tuner_mlp_unbal = kt.Hyperband(model_mlp_ar,
                     objective='val_accuracy',
                     max_epochs=100,
                     factor=3,
                     directory='my_dir_n1',
                     project_name='mlp_unbal')

# initializing a Hyperband tuner from Keras Tuner for hyperparameter optimization
for balanced dataset
tuner_mlp_bal = kt.Hyperband(model_mlp_ar,
                     objective='val_accuracy',
                     max_epochs=100,
                     factor=3,
                     directory='my_dirn2',
                     project_name='mlp_bal2')

# initializing early stopping with patience 10 and monitoring validation loss
early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10)
```

```
tuner_mlp_unbal.search(train_dataset_hog,epochs=100,validation_data=val_dataset_hog
, callbacks=[early_stopping])

# Geta the best hyperparameters
best_hps=tuner_mlp_unbal.get_best_hyperparameters(num_trials=1)[0]

print(f""" units: {best_hps.get('units')} , Learning Rate:
{best_hps.get('learning_rate')}, Dropout: {best_hps.get('rate')}
""")

# load the best optimized model and traing with unbalanced dataset
model_mlp_unbal = tuner_mlp_unbal.hypermodel.build(best_hps)
model_mlp_unbal_hist = model_mlp_unbal.fit(train_dataset, epochs=100,
validation_data=val_dataset, callbacks=[early_stopping] )

val_acc_per_epoch = model_mlp_unbal_hist.history['val_accuracy']
best_epoch_unbal = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch in TrainSet_imbal: %d' % (best_epoch_unbal))

# saving the best model
model_mlp_unbal.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/mo
del_mlp_hyp_unbal.h5')

#evaluating the model
model_mlp_unbal.evaluate(train_dataset)

# applying Hyperband searchin balanced datset
early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10)
try:

tuner_mlp_bal.search(balanced_gen_hog,epochs=100,validation_data=balanced_gen_val_h
og, callbacks=[early_stopping])
except:
  pass
best_hps_mlp_balanced=tuner_mlp_bal.get_best_hyperparameters(num_trials=1)[0]

# load the best optimized model and traing with unbalanced dataset

model_mlp_bal = tuner_mlp_bal.hypermodel.build(best_hps_mlp_balanced)
model_mlp_bal_hist = model_mlp_bal.fit(train_dataset, epochs=100,
validation_data=val_dataset, callbacks=[early_stopping] )

val_acc_per_epoch_bal = model_mlp_bal_hist.history['val_accuracy']
best_epoch_bal = val_acc_per_epoch_bal.index(max(val_acc_per_epoch_bal)) + 1
print('Best epoch: %d' % (best_epoch_bal,))

# saving the best model
model_mlp_bal.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/mode
l_mlp_hyp_bal.h5')
```

```
"""# CNN Baseline

"""

#initializing the baseline CNN model
model_cnn = Sequential()
model_cnn.add(layers.Conv2D(64,(3,3),activation='relu', input_shape=(64,64,3)))
model_cnn.add(layers.MaxPooling2D((2, 2)))

model_cnn.add(Flatten())
model_cnn.add(layers.Dense(64, activation='relu'))
model_cnn.add(Dense(3))

# Model summary
model_cnn.summary()

# Initializing ealy stooping
early_stopping= tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,

)
# Apply optimization function to the model
model_cnn.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
# traning the model with imbalanced datset
model_cnn_unbal = model_cnn.fit(train_dataset, epochs=100,
callbacks=[early_stopping],validation_data=val_dataset)

#saving the model
model_cnn.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/model_cn
n.h5')

# training model with balanced datset
model_cnn_bal = model_cnn.fit(balanced_gen, epochs=100,
callbacks=[early_stopping],validation_data=balanced_gen_val)

#saving the model
model_cnn.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/model_cn
n_bal.h5')

"""# RESNET 50"""

# initalizing input size for resnet 50
input_res = keras.Input(shape=(64,64,3))

# loading resnet 50 from keras
model_res = keras.applications.ResNet50(weights=None,input_tensor=input_res)
```

```
model_res.summary()

# Compile the model
model_res.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

# initalizing early stooping with patience as 10 and monitor on val_accuracy

early_stopping= tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=10,

)

# Training the model on imbalanced dataset
model_res_imbal.fit(train_dataset, epochs=100, callbacks=[early_stopping],
validation_data=val_dataset)

#saving the model trained on imbalanced dataset
model_res_imbal.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/mo
del_res.h5')

# Training the model on balanced dataset
model_res_bal.fit(balanced_gen, epochs=100, callbacks=[early_stopping_monitor],
validation_data=balanced_gen_val,)

#saving the model trained on balanced dataset
model_res_bal.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Models/mode
l_resv2.1.h5')



# ********** /code/test_cv.ipynb **********
# -*- coding: utf-8 -*-
"""test_cv.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1ceHnbzdXcqKCUWBmTq2AINnShmLorIfD
"""

!pip install retina-face

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

```
import matplotlib.pyplot as plt
from matplotlib import patches
from retinaface import RetinaFace
import itertools
import numpy as np
import sys
from PIL import Image
sys.modules['Image'] = Image
from collections import Counter


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
from tensorflow.keras.models import Model
from retinaface import RetinaFace


import torch
from sklearn.model_selection import train_test_split
from skimage.feature import hog
import skimage
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import sklearn
from sklearn.metrics import multilabel_confusion_matrix, ConfusionMatrixDisplay,
classification_report



import pathlib
import os
import glob
import pandas as pd
import cv2 as cv
import numpy as np
# import face_recognition
from google.colab.patches import cv2_imshow
from PIL import Image
```

```
"""# Loading Test Dataset"""

test_path="/content/drive/MyDrive/Computer-vision/cv-coursework/CW_Dataset/test"

#  extract and return the first line of a text file as an integer
def txtFileExtract(path):
    f=open(path,'r')
    content=f.readlines()
    return content[0]

# This function loads a dataset from a given path and reads the targets from .txt
files in the subdirectories of the given path
# and stores them in a list, then finds all the .jpeg images in the subdirectories
and returns their paths along with
# the targets list.
def load_dataset(path):
  targets = []
  labels=os.listdir(path)
  _image_path=glob.glob(path+'/*/*.jpeg')
  txt_path=glob.glob(path+'/*/*.txt')
  targets = [txtFileExtract(p) for p in txt_path]
  return _image_path, targets

# loads images from file paths and preprocesses
def load_image(img_path):
  _images = []
  for image_path in img_path:
    image = cv.imread(image_path)
    image = cv.resize(image, (64,64))
    image = cv.normalize(image, None, 0, 1.0, cv.NORM_MINMAX, dtype=cv.CV_32F)
    _images.append(image)
  return _images

#this function takes argument all image path and retun hog converted images as an
array
def load_image_to_hog(img_path):
  _images = []
  winStride = (8,8)
  padding = (8,8)
  locations = ((10,20),)
  HOG = cv.HOGDescriptor()
  for image_path in img_path:
    image = cv.imread(image_path)
    image = cv.resize(image, (64,64))
    fd, image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, channel_axis = -1)
    image = skimage.color.gray2rgb(image)
    _images.append(image)
  return _images
```

```
# storing the image path and labels in a variable
im_path, labels = load_dataset(test_path)

# changing the datatype of labels to float32
labels = np.array(labels).astype('float32')

# check the length of each classes
np.where( labels == 2.0)[0]

# saving the labels as y_test.npy for reusability
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/y_test.npy'
,labels)

# loading the labels
labels=np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/y_te
st.npy')

# applying the fuction to convert path to original images and hog images as list of
arrays
_or_images = load_image(im_path)
hog_images = load_image_to_hog(im_path)

# saving the numpy arrays
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/hog_images_
test.npy',hog_images)
np.save('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/or_images_t
est.npy',_or_images)

# converting the list to numpy array
or_images = np.array(_or_images)
hog_images = np.array(hog_images)

# loading the hog and original images which can be said as x_test
or_images =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/or_images_t
est.npy')
hog_images =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/hog_images_
test.npy')

# it converts the xtrain and y_train arrays into tensor dataset
test_dataset = tf.data.Dataset.from_tensor_slices((or_images, labels))
test_dataset = test_dataset.shuffle(buffer_size=1000,
reshuffle_each_iteration=True)

# it converts the xtrain and y_train arrays into tensor hog dataset
test_dataset_hog = tf.data.Dataset.from_tensor_slices((hog_images, labels))
test_dataset_hog = test_dataset_hog.shuffle(buffer_size=1000,
reshuffle_each_iteration=True)
```

```
# apply batches for all datasets
test_dataset = test_dataset.batch(64)
test_dataset_hog = test_dataset_hog.batch(64)

"""# HOG + MLP

### HOG+MLP with imbalanced Dataset
"""

#loading the model
model_mlp =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_mlp_hyp_unbal.h5')

#evaluating the model
model_mlp.evaluate(test_dataset_hog)

pred_mlp_unbal = model_mlp.predict(hog_images)

# getting the highest value using argmax and round off
predicted_classes = np.argmax(pred_mlp_unbal, axis=1)
predicted_classes

predicted_classes = np.array(predicted_classes)

print(classification_report(labels,predicted_classes))

cmx = confusion_matrix(labels.tolist(),predicted_classes.tolist())

#https://datascience.stackexchange.com/a/40068
classes = ['without_mask', 'with_mask', 'with_Improper_mask']
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

plot_confusion_matrix(cmx,classes)

"""### HOG+MLP with balanced Dataset"""

model_mlp_bal =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_mlp_hyp_bal.h5')

model_mlp_bal.evaluate(test_dataset_hog)

pred_mlp_bal = model_mlp_bal.predict(hog_images)
predicted_mlp_bal = np.argmax(pred_mlp_bal, axis=1)
cmx = confusion_matrix(labels.tolist(),predicted_mlp_bal.tolist())

#https://datascience.stackexchange.com/a/40068
classes = ['without_mask', 'with_mask', 'with_Improper_mask']
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

print(classification_report(labels,predicted_mlp_bal))

plot_confusion_matrix(cmx,classes)

"""# CNN Baseline
```

```
### CNN baseline with imbalanced dataset
"""

model_cnn =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_cnn.h5')

model_cnn.summary()

model_cnn.evaluate(test_dataset)

pred_cnn_unbal = model_cnn.predict(or_images)
predicted_cnn_unbal = np.argmax(pred_cnn_unbal, axis=1)
cm_cnn_unbal= confusion_matrix(labels.tolist(),predicted_cnn_unbal.tolist())

print(classification_report(labels,predicted_cnn_unbal))

plot_confusion_matrix(cm_cnn_unbal,classes)

"""### CNN baseline with Balanced Dataset"""

model_cnn_bal =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_cnn_bal.h5')

model_cnn_bal.evaluate(test_dataset)

pred_cnn_bal = model_cnn_bal.predict(or_images)
predicted_cnn_bal = np.argmax(pred_cnn_bal, axis=1)
cm_cnn_bal= confusion_matrix(labels.tolist(),predicted_cnn_bal.tolist())

print(classification_report(labels,predicted_cnn_bal))

plot_confusion_matrix(cm_cnn_bal,classes)

"""# RESNET 50

### RESNET 50 with imbalanced dataset
"""

model_res =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_res.h5')

model_res.evaluate(test_dataset)

pred_res_unbal = model_res.predict(or_images)
predicted_res_unbal = np.argmax(pred_res_unbal, axis=1)
cm_res_unbal= confusion_matrix(labels.tolist(),predicted_res_unbal.tolist())
```

```
print(classification_report(labels,predicted_res_unbal))

plot_confusion_matrix(cm_res_unbal,classes)

class_names = {0: "without mask", 1: "with mask", 2: "improper mask"}
idx = [8,255,166,350,455]
for i in idx:
  fig, axs = plt.subplots(1, 2, figsize=(10, 5))
  # Plot the original image in the first subplot
  class_name = class_names[int(labels[i])]
  axs[0].imshow(or_images[i])
  print(class_name)
  axs[0].set_title("Original:"+str(class_name))

  # Plot the predicted label in the second subplot
  class_name_pred = class_names[int(predicted_res_unbal[i])]
  axs[1].imshow(or_images[i])
  axs[1].set_title("predicted:"+ str(class_name_pred))

  # Display the plot
  plt.show()



"""### RESNET 50 with balanced dataset"""

model_res_balanced =
tf.keras.models.load_model('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Mod
els/model_resv2.1.h5')

model_res_balanced.evaluate(test_dataset)

pred_res_bal = model_res_balanced.predict(or_images)
predicted_res_bal = np.argmax(pred_res_bal, axis=1)
cm_res_bal= confusion_matrix(labels.tolist(),predicted_res_bal.tolist())

print(classification_report(labels,predicted_res_bal))

plot_confusion_matrix(cm_res_bal,classes)

"""# Processing Video"""

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Loading the video
cap =
cv.VideoCapture('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Video/TestVide
o.mp4')

# Getting the frames per second and size of the video
```

```
fps = int(cap.get(cv.CAP_PROP_FPS))
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
size = (frame_width, frame_height)

# Creating a dictionary of class names for the face mask classification
class_names = {0: "without mask", 1: "with mask", 2: "improper mask"}
print('FPS:', fps)
print('size:',size)
# Create a VideoWriter object for the output file
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('output.mp4', fourcc, fps, size)

# Initializing counters for frame and detected faces
count = 0
number_of_frames = int(cap.get(cv.CAP_PROP_FRAME_COUNT))

# Starting the video processing loop
while True:
  # Read the current frame
  s, img = cap.read()
  if not s:
    break
  # Detecting faces in the current frame using a RetinaFace model
  ret_faces = RetinaFace.detect_faces(img)
  # If faces are detected, classify them and draw boxes with labels on the image
  if ret_faces is not None:
    count += 1
    try:
      # Get the coordinates of the detected face
      x1,y1,x2,y2 = ret_faces['face_1']['facial_area'][0],
ret_faces['face_1']['facial_area'][1], ret_faces['face_1']['facial_area'][2],
ret_faces['face_1']['facial_area'][3]
      # Drawing a rectangle around the face on the image
      img = cv.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (128, 0, 0),
2)
      # Croping the detected face
      croped_faces = img[y1:y2, x1:x2]
      # Creating a frame count string
      frame_count = 'Frame: ' + str(count)
    except:
      continue
    # Resizing the cropped face to 64x64 pixels and normalize its values
    _face = cv.resize(croped_faces,(64,64))
    _face = cv.normalize(_face, None, 0, 1.0, cv.NORM_MINMAX, dtype=cv.CV_32F)
    _face = _face[tf.newaxis,:,:,:]
    # Classifying the face using a pre-trained model and get its class index
    pred = model_res.predict(_face)
    class_idx = tf.argmax(pred, axis=-1)
    class_idx = class_idx.numpy()
```

```
    # Get the class name from the dictionary using the class index
    class_name = class_names[int(class_idx)]
    # Get the class name from the dictionary using the class index
    w,h= cv.getTextSize(class_name,cv.FONT_HERSHEY_SIMPLEX, 0.5, 2)
    img = cv.rectangle(img, (int(x1), int(y1)-25), (int(x1)+w[0], int(y1)), (0, 0,
0), -1)
    img = cv.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 2)
    img = cv.putText(img, class_name, (int(x1), int(y1)-10),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    img = cv.putText(img, frame_count, (int(frame_width)-150,
int(frame_height)-500), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)
    pil_image = Image.fromarray(img)
    pil_image.show()
  out.write(img)

  if cv.waitKey(1) & 0xFF == ord('q'):
      break
cap.release()
out.release()
cv.destroyAllWindows()
print("success")



# *************** /test_function.ipynb ***********************


# -*- coding: utf-8 -*-
"""test_functions.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1cubsXJXGICTE9-kIPnTFkbnyr5IL3tWj

# Computer Vision Coursework Submission (INM460)

**Student name, ID and cohort:** Sam Clastine Jesumuthu (220038747) - PG

# Notebook Setup
In this section you should include all the code cells required to test your
coursework submission. Specifically:

### Mount Google Drive
"""

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

!pip install retina-face
```

"""### Define Local Path

In the next cell you should assign to the variable `GOOGLE_DRIVE_PATH_AFTER_MYDRIVE` the relative path of this folder in your Google Drive.

IMPORTANT: you have to make sure that **all the files required to test your functions are loaded using this variable** (as was the case for all lab tutorials). In other words, do not use in the notebook any absolute paths. This will ensure that the markers can run your functions. Also, **do not use** the magic command `%cd` to change directory.

"""

```
import os

# TODO: Fill in the Google Drive path where you uploaded the CW_folder_PG
# Example: GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Computer
Vision/CW_folder_PG'

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE =
'/content/drive/MyDrive/Computer-vision/CW_Folder_PG'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive',
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))
```

"""### Load packages

In the next cell you should load all the packages required to test your functions.
"""

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import cv2 as cv
from retinaface import RetinaFace
from matplotlib import animation, rc
from IPython.display import HTML
import imageio
from skimage.transform import resize
```

"""### Load models

In the next cell you should load all your trained models for easier testing of your functions. Avoid to load them within `MaskDetection` and `MaskDetectionVideo` to avoid having to reload them each time.
"""

```
hog_mlp_imbalanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_mlp_hyp_unbal.h5'))
hog_mlp_balanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_mlp_hyp_bal.h5'))
baseline_cnn_imbalanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_cnn.h5'))
baseline_cnn_balanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_cnn_bal.h5'))
Resnet_50_imbalanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_res.h5'))
Resnet_50_balanced = load_model(os.path.join(GOOGLE_DRIVE_PATH,
'Models','model_resv2.1.h5'))

"""# Load Test Dataset"""

x_test =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/or_images_t
est.npy')
x_test_hog =
np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/hog_images_
test.npy')
y_test
=np.load('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/CW_Dataset/y_test.npy
')

"""# Test MaskDetection

This section should allow a quick test of the `MaskDetection` function. First, add
cells with the code needed to load the necessary subroutines to make
`MaskDetection` work.
"""

def modelPlot(x_test_hog,x_test,y_test,model,modelName):
  if modelName == 'hog-mlp-imbalanced' or 'hog-mlp-balanced':
    prediction = model.predict(x_test_hog)
  else:
    prediction = model.predict(x_test)
  prediction = np.argmax(prediction, axis=1)
  class_names = {0: "without mask", 1: "with mask", 2: "improper mask"}
  fig, axs = plt.subplots(1, 4, figsize=(20, 10))
  predicted_class_name = class_names[int(prediction[8])]
  class_name = class_names[int(y_test[8])]
  axs[0].imshow(cv.cvtColor(x_test[8], cv.COLOR_BGR2RGB))
  axs[0].set_title("Original:"+str(class_name)+", predicted:"+
str(predicted_class_name))

  class_name = class_names[int(y_test[255])]
  predicted_class_name = class_names[int(prediction[255])]
  axs[1].imshow(cv.cvtColor(x_test[255], cv.COLOR_BGR2RGB))
```

```
  axs[1].set_title("Original:"+str(class_name)+", predicted:"+
str(predicted_class_name))

  class_name = class_names[int(y_test[166])]
  predicted_class_name = class_names[int(prediction[166])]
  axs[2].imshow(cv.cvtColor(x_test[166], cv.COLOR_BGR2RGB))
  axs[2].set_title("Original:"+str(class_name)+", predicted:"+
str(predicted_class_name))

  class_name = class_names[int(y_test[455])]
  predicted_class_name = class_names[int(prediction[455])]
  axs[3].imshow(cv.cvtColor(x_test[455], cv.COLOR_BGR2RGB))
  axs[3].set_title("Original:"+str(class_name)+", predicted:"+
str(predicted_class_name))
  plt.show()

def MaskDetection(x_test, x_test_hog, y_test, modelName):
  if modelName == 'hog-mlp-imbalanced':
    modelPlot(x_test_hog,x_test,y_test,hog_mlp_imbalanced,modelName)

  if modelName== 'hog-mlp-balanced':
    prediction = hog_mlp_balanced.predict(x_test_hog)
    modelPlot(x_test_hog,x_test,y_test,hog_mlp_balanced,modelName)

  if modelName== 'baseline-cnn-imbalanced':
    baseline_cnn_prediction = baseline_cnn_imbalanced.predict(x_test)
    modelPlot(x_test_hog,x_test,y_test,baseline_cnn_imbalanced,modelName)

  if modelName== 'baseline-cnn-balanced':
    baseline_cnn_prediction = baseline_cnn_balanced.predict(x_test)
    modelPlot(x_test_hog,x_test,y_test,baseline_cnn_balanced,modelName)

  if modelName== 'Resnet-50-imbalanced':
    resNet_50_prediction = Resnet_50_imbalanced.predict(x_test)
    modelPlot(x_test_hog,x_test,y_test,Resnet_50_imbalanced,modelName)

  if modelName== 'Resnet-50-balanced':
    resNet_50_prediction = Resnet_50_balanced.predict(x_test)
    modelPlot(x_test_hog,x_test,y_test,Resnet_50_balanced,modelName)

"""Then, make a call to the `MaskDetection` function to see what results it
produces. You must also indicate the syntax needed to test your different
models."""

# where model_type can be one of
  # 'hog-mlp-imbalanced'
  # 'hog-mlp-balanced'
  # 'baseline-cnn-balanced'
  # 'baseline-cnn-imbalanced'
  # 'Resnet-50-balanced'
```

```
  # 'Resnet-50-imbalanced'
MaskDetection(x_test, x_test_hog,y_test,'Resnet-50-balanced')

"""# Test MaskDetectionVideo

This section should allow a quick test of the `MaskDetectionVideo` function. First,
add cells with the code needed to load the necessary subroutines to make
`MaskDetectionVideo` work.
"""

def MaskDetectionVideo(file):
  # Loading the video
  cap = cv.VideoCapture(file)
  fps = int(cap.get(cv.CAP_PROP_FPS))
  # Getting the frames per second and size of the video
  frame_width = int(cap.get(3))
  frame_height = int(cap.get(4))
  size = (frame_width, frame_height)
  # Creating a dictionary of class names for the face mask classification
  class_names = {0: "without mask", 1: "with mask", 2: "improper mask"}
  print('FPS:', fps)
  print('size:',size)
  # Create a VideoWriter object for the output file
  fourcc = cv.VideoWriter_fourcc(*'XVID')
  out = cv.VideoWriter('output.mp4', fourcc, fps, size)
  # Initializing counters for frame and detected faces
  count = 0
  number_of_frames = int(cap.get(cv.CAP_PROP_FRAME_COUNT))
  # Starting the video processing loop
  while True:
    # Read the current frame
    s, img = cap.read()
    if not s:
      break
    # Detecting faces in the current frame using a RetinaFace model
    ret_faces = RetinaFace.detect_faces(img)
    # If faces are detected, classify them and draw boxes with labels on the image
    if ret_faces is not None:
      count += 1
      try:
        # Get the coordinates of the detected face
        x1,y1,x2,y2 = ret_faces['face_1']['facial_area'][0],
ret_faces['face_1']['facial_area'][1], ret_faces['face_1']['facial_area'][2],
ret_faces['face_1']['facial_area'][3]
        # Drawing a rectangle around the face on the image
        img = cv.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (128, 0,
0), 2)
        # Croping the detected face
        croped_faces = img[y1:y2, x1:x2]
        # Creating a frame count string
```

```
        frame_count = 'Frame: ' + str(count)
      except:
        continue
        # Resizing the cropped face to 64x64 pixels and normalize its values
      _face = cv.resize(croped_faces,(64,64))
      _face = cv.normalize(_face, None, 0, 1.0, cv.NORM_MINMAX, dtype=cv.CV_32F)
      _face = _face[tf.newaxis,:,:,:]
      # Classifying the face using a pre-trained model and get its class index
      pred = Resnet_50_imbalanced.predict(_face)
      class_idx = tf.argmax(pred, axis=-1)
      class_idx = class_idx.numpy()
      # Get the class name from the dictionary using the class index
      class_name = class_names[int(class_idx)]
      # Get the class name from the dictionary using the class index
      w,h= cv.getTextSize(class_name,cv.FONT_HERSHEY_SIMPLEX, 0.5, 2)
      img = cv.rectangle(img, (int(x1), int(y1)-25), (int(x1)+w[0], int(y1)), (0,
0, 0), -1)
      img = cv.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0),
2)
      img = cv.putText(img, class_name, (int(x1), int(y1)-10),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
      img = cv.putText(img, frame_count, (int(frame_width)-150,
int(frame_height)-500), cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)
    out.write(img)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break
  # Release the resources
  cap.release()
  out.release()
  cv.destroyAllWindows()
  video = imageio.mimread('/content/output.mp4',memtest=False)  #Loading video
  video = [resize(frame, (540,940))[..., :3] for frame in video]
  HTML(display_video(video).to_html5_video())
  print("success")

#https://github.com/TUIlmenauAMS/Videocoding/blob/main/seminars/vc_seminar01_suppor
t.ipynb
def display_video(video):
    fig = plt.figure(figsize=(7,7))  #Display size specification

    mov = []
    for i in range(len(video)):  #Append videos one by one to mov
        img = plt.imshow(video[i], animated=True)
        plt.axis('off')
        mov.append([img])

    #Animation creation
    anime = animation.ArtistAnimation(fig, mov, interval=50, repeat_delay=1000)
```

```
    plt.close()
    return anime

"""Then, make a call to the `MaskDetectionVideo` function to see what results it
produces."""

MaskDetectionVideo('/content/drive/MyDrive/Computer-vision/CW_Folder_PG/Video/TestV
ideo.mp4')

#https://github.com/TUIlmenauAMS/Videocoding/blob/main/seminars/vc_seminar01_suppor
t.ipynb

video = imageio.mimread('/content/output.mp4',memtest=False)  #Loading video
video = [resize(frame, (540,940))[..., :3] for frame in video]
HTML(display_video(video).to_html5_video())  #Inline video display in HTML5
```