## **A Tutorial for the DAISEC Project**

## 

## 

## **Image Manipulation Detection**

## 

## 

## Juan Shan and Ian Carvalh

## 

## 

## 

## 

## 

## 

## 

## 

## 

## **Seidenberg School of Computer Science and Information Systems**

## **Pace University**

## 

## **October 2019**

## 

## 

## 

## 

## 

## 

## 

## 

## **Copyright for Material Reuse**

## 

## **This materials are developed under the support of the Department of Defense. Copyright© 2019 Juan Shan (jshan@pace.edu) and Ian Carvalho (ic34882n@pace.edu), Pace University. Please properly acknowledge the source for any reuse of the materials as below.**

## 

## **Li-Chiou Chen and Ian Carvalho, “Image Manipulation Detection,” A Tutorial for the DAISEC Project, Pace University, 2019.**

## 

## **Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation. A copy of the license is available at http://www.gnu.org/copyleft/fdl.html.**

## 

## 

## **Acknowledgment**

## 

## **The author(s) would like to acknowledge the support from the Department of Defense under CNAP Grant No. H98230-17-1-0335. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. government.**

## 

## 

## **Introduction: Use Deep Learning Models to Detect Image Manipulation**

With the increasing popularity of image manipulation from memes creation to image filters apps, there was a concern in the malicious use of such applications. Digitally forged images have been used to target opposing politicians, support or depreciate public manifestations, spread hoaxes, and et. cetera.

Studies like this article (https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-017-0067-2) have shown that people are not good at detecting manipulated images. In this tutorial, we will see how to train deep learning models to spot manipulated images.

The images contained in the dataset are separated into two categories: manipulated and non-manipulated images.

## **Dataset 1**

Dataset 1 is composed of face images gathered from Twitter and the Celebrity Face Dataset.

The Face Swap images were downloaded from Twitter searching from hashtags like #FaceSwap that indicates the image suffered some kind of manipulation. The Celebrity Face Dataset is a free public dataset that contains face images without any manipulation. There are 2546 images in total; 1039 images are non-manipulated and 1507 are manipulated. Below are some examples from each category.

**Images without manipulation (1039 images)**

**Faceswap images (1507 images)**

## **Dataset 2**

Dataset 2 consists of a series of more general images than face images in dataset 1. There are 566 original images collected from a varying number of situations, and they form category one: **Original** dataset. Each of these images was manipulated using some image editing tools like photoshop, and they form category two: **Doctored** dataset. The purpose of collecting dataset 2 is to test how deep learning models can detect manipulated images from more general situations.

Source of images:<https://www.reddit.com/r/photoshopbattles/>

**Original (566 images)**

**** 

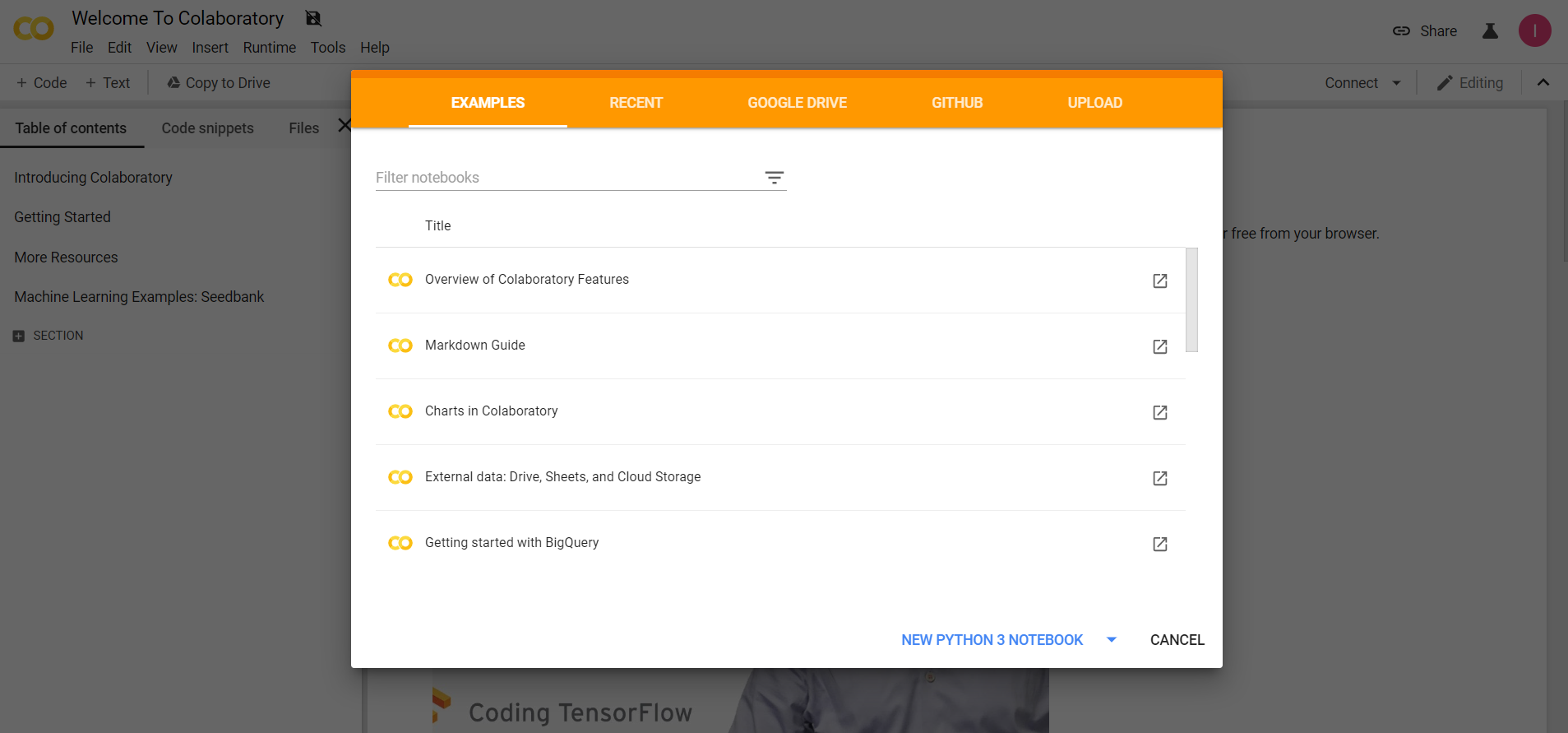
**Doctored (566 images)**

**** 

## **Google Collaboratory**

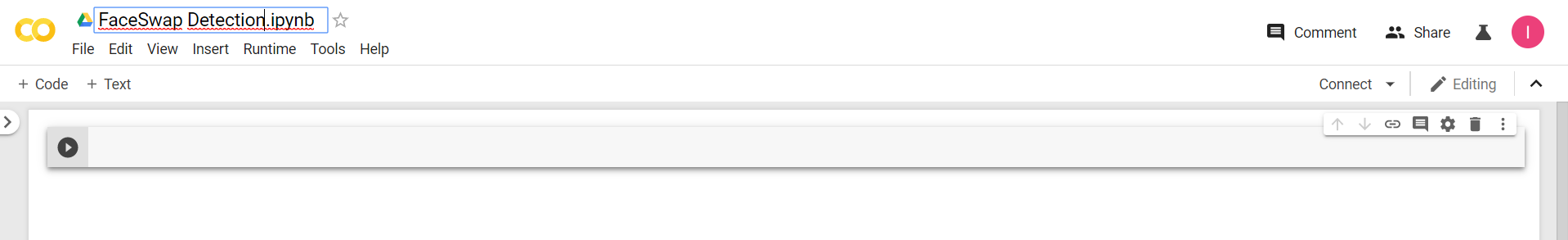
In order to train your model, you will be using Google Collaboratory, an online tool that allows running Python for research purposes and has support to train deep learning models on the GPU.

Access [https://colab.research.google.com](https://colab.research.google.com/) in your browser and you should see a screen like this.

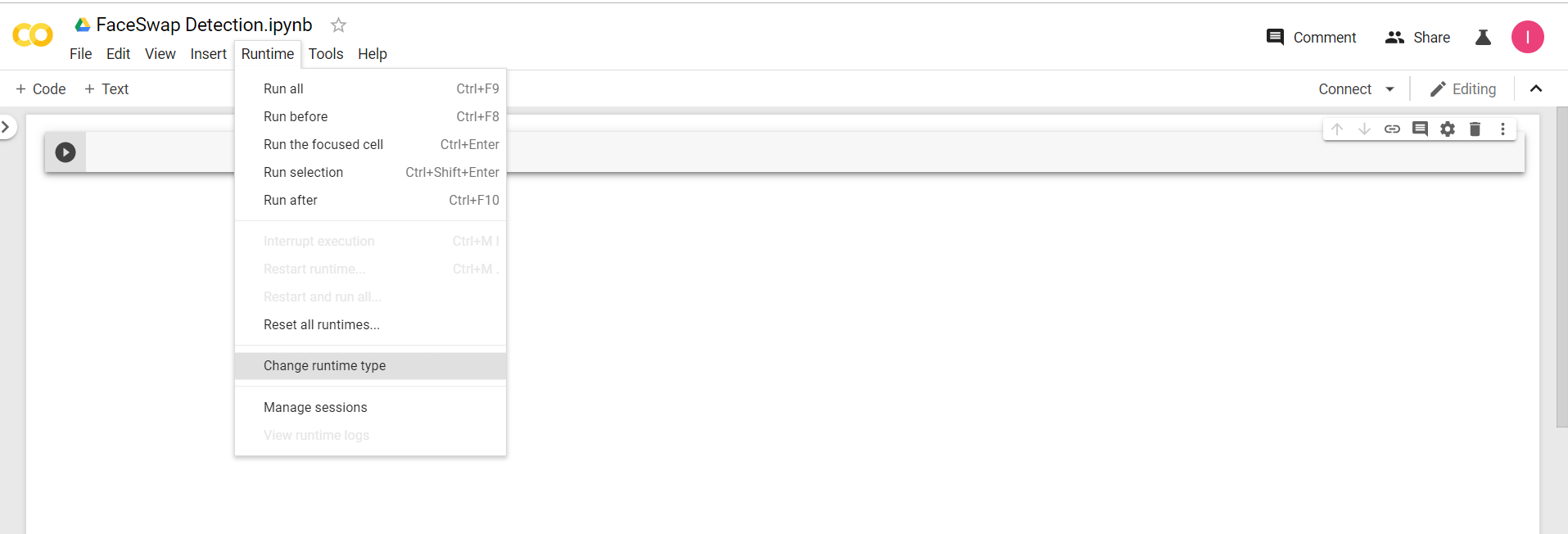


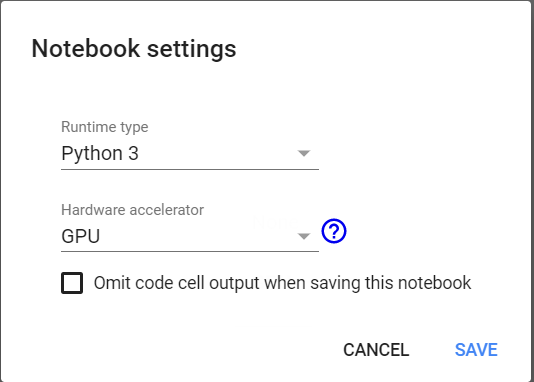
Jupyter Notebook is an environment that allows you to run python code on the cloud, without the need to install anything on your computer. Create a new Notebook by clicking “New Python 3 Notebook”.

Change the name of your Notebook by clicking on the title label and choosing a more appropriate title

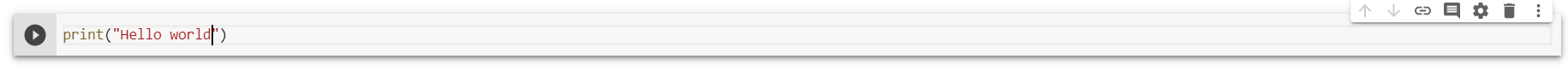


Remember to change your runtime time by going to the top menu and selection Runtime > Change Runtime Type and select “GPU” at the hardware accelerator box and clicking “Save”.

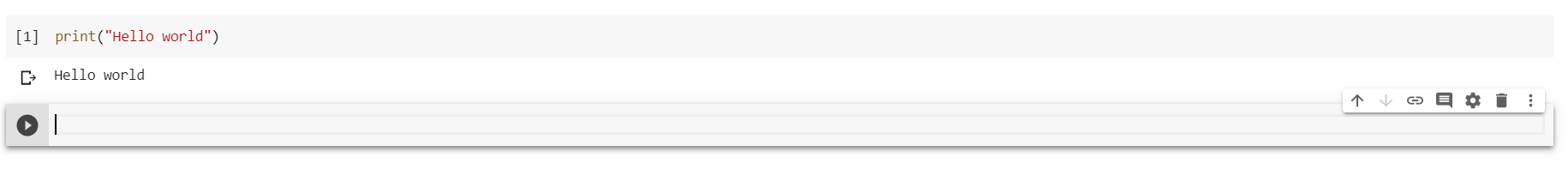




Google Colab allows you to run Python code and simply pressing Shift+Return.



You should see the result of your code evaluation right below your code.



## **Model Training**

**Step 1: Downloading data in Google Colab**

The following code paragraph is going to download the first dataset to your Colab environment. Copy and paste the following code to Google Colab notebook. Run it.

%%shell

ggID='11aYEGVfWneiy7Zb5tA5ZXuvxT5cwAT0R'

ggURL='https://drive.google.com/uc?export=download'

filename="$(curl -sc /tmp/gcokie "${ggURL}&id=${ggID}" | grep -o '="uc-name.\*</span>' | sed 's/.\*">//;s/<.a> .\*//')"

getcode="$(awk '/\_warning\_/ {print $NF}' /tmp/gcokie)"

curl -Lb /tmp/gcokie "${ggURL}&confirm=${getcode}&id=${ggID}" -o dataset1.zip

unzip dataset1.zip

Note: the above code download dataset1 to your Colab drive. In case you want to download the dataset to a local disk, use this link: <https://drive.google.com/uc?export=download&id=101o_pNtg99wHuioHhEqKMGNxLZkQPBh4>​

**Step 2: Creating the DataGenerator**

The following code paragraph defines how to generate the training and testing sets. The split parameter is 0.2, which means 80% of the data goes to training set and 20% of the data goes to testing set.

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D

from keras.layers import Activation, Dropout, Flatten, Dense

from keras import backend as K

dataset = 'dataset1'

# dimensions of our images.

img\_width, img\_height = 64, 64

datagen = ImageDataGenerator(validation\_split=0.2)

train\_generator = datagen.flow\_from\_directory(

dataset,

target\_size=(img\_width, img\_height),

subset='training',

class\_mode='binary',

)

validation\_generator = datagen.flow\_from\_directory(

dataset,

target\_size=(img\_width, img\_height),

subset='validation',

class\_mode='binary',

shuffle=False

)

**Step 3: Train the first model - No transfer learning**

The first convolutional neural network (CNN) model is constructed below. It has three convolutional layers and two fully connected layer. The parameters are randomly initialized so we will train the model from scratch.

nb\_train\_samples = len(train\_generator.classes)

nb\_train\_samples -= (nb\_train\_samples % batch\_size)

nb\_validation\_samples = len(validation\_generator.classes)

nb\_validation\_samples -= (nb\_validation\_samples % batch\_size)

epochs = 50

batch\_size = 32

if K.image\_data\_format() == 'channels\_first':

input\_shape = (3, img\_width, img\_height)

else:

input\_shape = (img\_width, img\_height, 3)

first\_model = Sequential()

first\_model.add(Conv2D(32, (3, 3), input\_shape=input\_shape))

first\_model.add(Activation('relu'))

first\_model.add(MaxPooling2D(pool\_size=(2, 2)))

first\_model.add(Conv2D(32, (3, 3)))

first\_model.add(Activation('relu'))

first\_model.add(MaxPooling2D(pool\_size=(2, 2)))

first\_model.add(Conv2D(64, (3, 3)))

first\_model.add(Activation('relu'))

first\_model.add(MaxPooling2D(pool\_size=(2, 2)))

first\_model.add(Flatten())

first\_model.add(Dense(64))

first\_model.add(Activation('relu'))

first\_model.add(Dropout(0.5))

first\_model.add(Dense(1))

first\_model.add(Activation('sigmoid'))

first\_model.compile(loss='binary\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

first\_model\_history = first\_model.fit\_generator(

train\_generator,

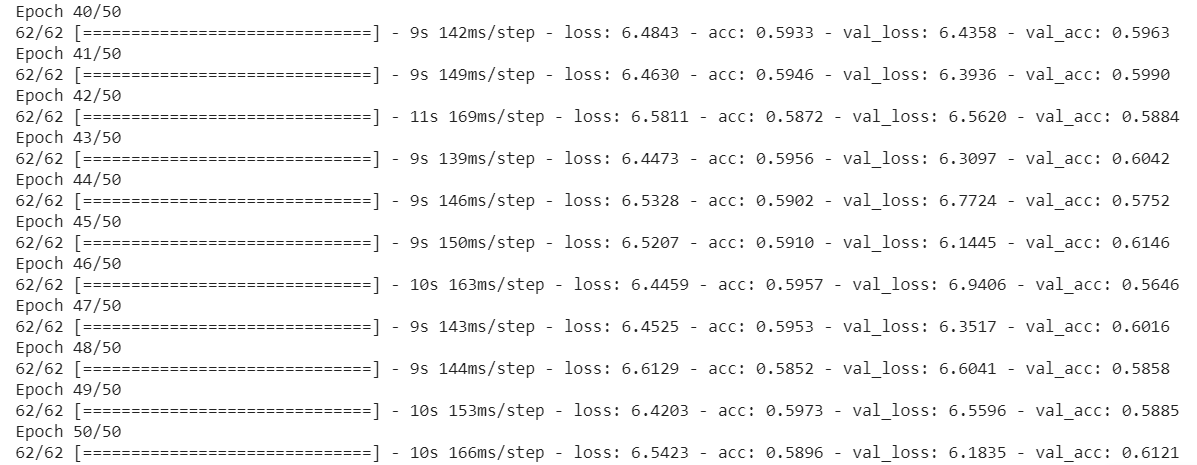
steps\_per\_epoch=nb\_train\_samples // batch\_size,

epochs=epochs,

validation\_data=validation\_generator,

validation\_steps=nb\_validation\_samples // batch\_size)

first\_model.save\_weights('first\_attempt.h5')



* In the above output, what does val\_acc mean? What is the val\_acc after epoch 1 and what is the val\_acc after epoch 50?
* If you increase the epoch to 100, will you get better accuracy?

**Step 4: Plotting performance and training history**

We will plot the training and testing performance curve during the 50 epochs training process.

def plot\_history(history):

print(history.keys())

# summarize history for accuracy

plt.plot(history['acc'])

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

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history['loss'])

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

plt.title('model loss')

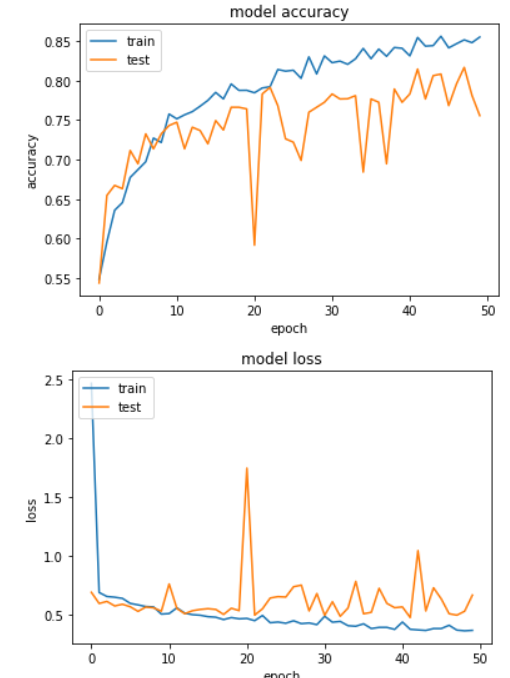
plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

plot\_history(first\_model\_history.history)



* Why the model accuracy increases generally while the model loss drops?
* The test accuracy is lower than the train. Why?
* According to the performance curve, do you think your model is well trained?
* Is there anything you can do to further improve the performance?

**Step 5: Second attempt - with transfer learning**

Now we will train a second model for the same task. This model is pre-trained on a large image dataset called ImageNet. The parameters are saved and copied. We will use our data to fine-tune this trained model, and update the parameters to tailor the model more suitable for our image manipulation detect task.

from keras.applications import VGG16

import numpy as np

vgg\_conv = VGG16(weights='imagenet',

include\_top=False,

input\_shape=(img\_width, img\_height, 3))

i = 0

print(nb\_train\_samples)

train\_features = np.zeros(shape=(nb\_train\_samples, 2, 2, 512))

train\_labels = np.zeros(shape=(nb\_train\_samples))

for inputs\_batch, labels\_batch in train\_generator:

features\_batch = vgg\_conv.predict(inputs\_batch)

if features\_batch.shape[0] < batch\_size:

print(features\_batch.shape)

continue

train\_features[i \* batch\_size : (i + 1) \* batch\_size] = features\_batch

train\_labels[i \* batch\_size : (i + 1) \* batch\_size] = labels\_batch

i += 1

if i \* batch\_size >= nb\_train\_samples:

break

train\_features = np.reshape(train\_features, (nb\_train\_samples, 2 \* 2 \* 512))

i = 0

validation\_features = np.zeros(shape=(nb\_validation\_samples, 2, 2, 512))

validation\_labels = np.zeros(shape=(nb\_validation\_samples))

print(validation\_features.shape)

for inputs\_batch, labels\_batch in validation\_generator:

features\_batch = vgg\_conv.predict(inputs\_batch)

if features\_batch.shape[0] < batch\_size:

continue

validation\_features[i \* batch\_size : (i + 1) \* batch\_size] = features\_batch

validation\_labels[i \* batch\_size : (i + 1) \* batch\_size] = labels\_batch

i += 1

if i \* batch\_size >= nb\_validation\_samples:

break

validation\_features = np.reshape(validation\_features, (nb\_validation\_samples, 2 \* 2 \* 512))

from keras import models

from keras import layers

from keras import optimizers

second\_model = models.Sequential()

second\_model.add(layers.Dense(256, activation='relu', input\_dim=2 \* 2 \* 512))

second\_model.add(layers.Dropout(0.5))

second\_model.add(layers.Dense(128, activation='relu'))

second\_model.add(layers.Dropout(0.5))

second\_model.add(layers.Dense(1, activation='sigmoid'))

second\_model.compile(optimizer=optimizers.RMSprop(lr=1e-4),

loss='binary\_crossentropy',

metrics=['acc'])

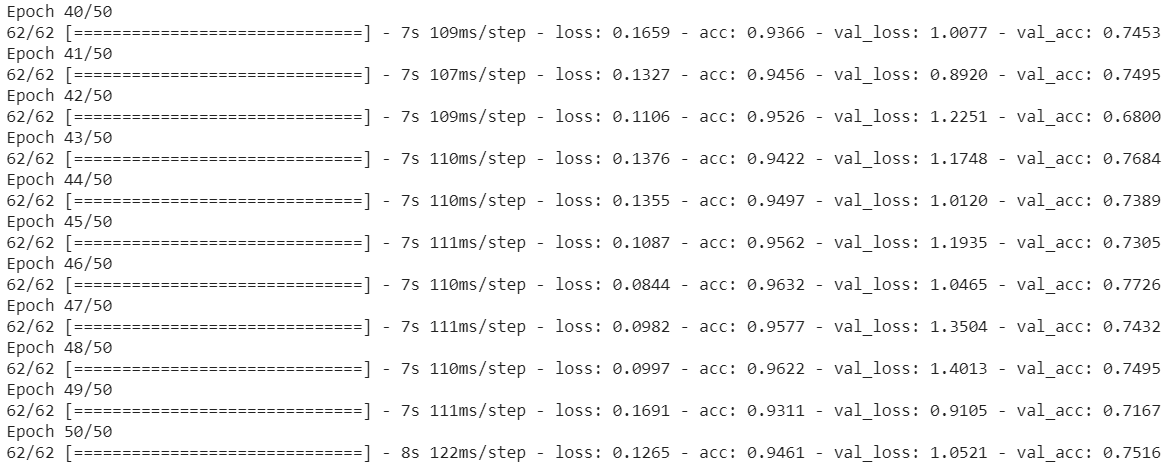
second\_model\_history = second\_model.fit(train\_features,

train\_labels,

epochs=epochs,

batch\_size=batch\_size,

validation\_data=(validation\_features,validation\_labels))



* After 50 epochs, what is the val\_acc of the second model? Is it improved compared to the first model?

**Step 6: Plotting performance and training history**

* Now, it’s your turn to plot the training history for the second model.

**Step 7: Further evaluation**

Precision identifies the frequency with which a model was correct when predicting the positive class. Recall is metric for classification models that answers the following question: Out of all the possible positive labels, how many did the model correctly identify? See the following link for further explanation of these two metrics.

<https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall>

The following code paragraph is going to evaluate the model using different evaluation metrics. The confusion matrix is created.

from sklearn.metrics import classification\_report, confusion\_matrix

target\_names = ['Original', 'Faceswap']

def model\_classification\_report(y, y\_pred, target\_names):

print('Model - Confusion Matrix')

print(confusion\_matrix(y, y\_pred))

print('Model - Classification Report')

print(classification\_report(y, y\_pred, target\_names=target\_names))

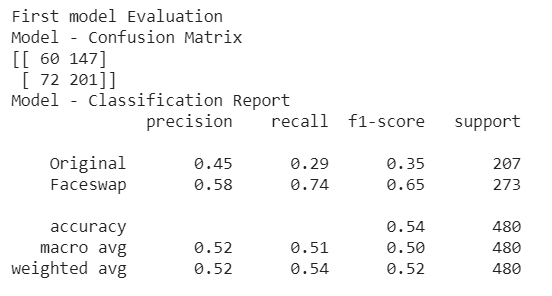
Y\_pred = first\_model.predict\_generator(validation\_generator, nb\_validation\_samples // batch\_size)

y\_pred = Y\_pred > 0.5

y = validation\_generator.classes[:len(y\_pred)]

print('First model Evaluation')

model\_classification\_report(y, y\_pred, target\_names)



* In the confusion matrix, what does f1-score mean?
* Now, write your own code to generate the Confusion Matrix and Classification Report for the second model.

This is the end of the program. The complete code of the model training, testing and evaluation is provided in the following link:<https://colab.research.google.com/drive/1eHHl1akyzZ5gx1OS7TQeYegyQ2NfVvuH>

The above notebook is provided for your reference. However, you are not suggested to run this notebook directly, instead, you want to create your own notebook and copy and paste each code paragraph to run and understand each subsection of the program.

## **More Experiments**

* For dataset 1, the current image size is scaled to 64x64 from the original 256x256. Try changing the size of the image and see how the results in the evaluation and processing time is affected by image size.
* Run the experiment on dataset 2. To do this, download dataset 2 using the following code:

%%shell

ggID='1qXnBd-cCfWKJB6MvWnGZgOitfIhWoEs9'

ggURL='https://drive.google.com/uc?export=download'

filename="$(curl -sc /tmp/gcokie "${ggURL}&id=${ggID}" | grep -o '="uc-name.\*</span>' | sed 's/.\*">//;s/<.a> .\*//')"

getcode="$(awk '/\_warning\_/ {print $NF}' /tmp/gcokie)"

curl -Lb /tmp/gcokie "${ggURL}&confirm=${getcode}&id=${ggID}" -o dataset2.zip

unzip dataset2.zip

* Update the variable *dataset* to “dataset2” in the code. Train a new model for dataset 2, with and without transfer learning. See how the detection accuracy on the second dataset. Do you expect it is higher or lower than the first dataset? Why?