DeepFake Face Detection

Photo by Javier Jaén, Svetikd on The New Yorker

We have seen after the development of GANs, Deepfakes came into existence.Though the development of these techniques were primarily to increase the amount of training data but many people were found misusing these techniques for criminal activities. So, it is the need of the hour to develop one such model which can differentiate between real and deepfake faces.

Import Dataset and Necessary Libraries

```
!wget -N "https://cainvas-static.s3.amazonaws.com/media/user_data/cainvas-admin/realVSfake.zip"
!unzip -qo realVSfake.zip
!rm realVSfake.zip
```

```
--2021-03-18 17:09:25--  https://cainvas-static.s3.amazonaws.com/media/user_data/cainvas-admin/realVSfake.zip
Resolving cainvas-static.s3.amazonaws.com (cainvas-static.s3.amazonaws.com)... 52.219.62.36
Connecting to cainvas-static.s3.amazonaws.com (cainvas-static.s3.amazonaws.com)|52.219.62.36|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 97144373 (93M) [application/zip]
Saving to: ‘realVSfake.zip’

realVSfake.zip      100%[===================>]  92.64M  47.3MB/s    in 2.0s

2021-03-18 17:09:27 (47.3 MB/s) - ‘realVSfake.zip’ saved [97144373/97144373]
```

```
import numpy as np
import pandas as pd
from keras.applications.mobilenet import preprocess_input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Dense,BatchNormalization, Flatten, MaxPool2D
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, Callback
from keras.layers import Conv2D, Reshape
from keras.utils import Sequence
from keras.backend import epsilon
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from keras.layers import Convolution2D, Conv2D, MaxPooling2D, GlobalAveragePooling2D
import cv2
import keras

from tqdm.notebook import tqdm_notebook as tqdm

import os
```

```
print(os.listdir("realVSfake/real_and_fake_face"))
```

```
['training_fake', 'training_real']
```

```
real = "realVSfake/real_and_fake_face/training_real/"
fake = "realVSfake/real_and_fake_face/training_fake/"

real_path = os.listdir(real)
fake_path = os.listdir(fake)
```

Visulaizing the real and fake faces

```
def load_img(path):
    image = cv2.imread(path)
    image = cv2.resize(image,(224, 224))
    return image[...,::-1]
```

```
fig = plt.figure(figsize=(10, 10))

for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(load_img(real + real_path[i]), cmap='gray')
    plt.suptitle("Real faces",fontsize=20)
    plt.axis('off')

plt.show()
```

```
fake_path.pop(5)
```

```
'.ipynb_checkpoints'
```

```
fig = plt.figure(figsize=(10,10))

for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(load_img(fake + fake_path[i]), cmap='gray')
    plt.suptitle("Fakes faces",fontsize=20)
    plt.title(fake_path[i][:4])
    plt.axis('off')

plt.show()
```

png

```
dataset_path = "realVSfake/real_and_fake_face"
```

Data augumentation and Data Loader

```
data_with_aug = ImageDataGenerator(horizontal_flip=True,
                                   vertical_flip=False,
                                   rescale=1./255,
                                  validation_split=0.2)
```

```
test = data_with_aug.flow_from_directory(dataset_path,
                                         class_mode="binary",
                                         target_size=(224, 224),
                                         batch_size=32,
                                         subset="validation"
                                         )
```

```
Found 167 images belonging to 2 classes.
```

```
train = data_with_aug.flow_from_directory(dataset_path,
                                         class_mode="binary",
                                         target_size=(224, 224),
                                         batch_size=32)
```

```
Found 835 images belonging to 2 classes.
```

Building VGG16 model from Scratch

```
 # The original model does not contain Dropout Layers
vgg_model = Sequential()
vgg_model.add(Conv2D(filters=64, kernel_size=1, input_shape=(224, 224, 3), activation='relu'))
vgg_model.add(Conv2D(filters=64, kernel_size=1, activation='relu'))
vgg_model.add(MaxPooling2D(pool_size=2))
vgg_model.add(Dropout(0.2))

vgg_model.add(Conv2D(filters=128, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=128, kernel_size=1, activation='relu'))
vgg_model.add(MaxPooling2D(pool_size=2))
vgg_model.add(Dropout(0.2))

vgg_model.add(Conv2D(filters=256, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=256, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=256, kernel_size=1, activation='relu'))
vgg_model.add(MaxPooling2D(pool_size=2))
vgg_model.add(Dropout(0.2))

vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(MaxPooling2D(pool_size=2))
vgg_model.add(Dropout(0.2))

vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(Conv2D(filters=512, kernel_size=1, activation='relu'))
vgg_model.add(MaxPooling2D(pool_size=2))
vgg_model.add(Dropout(0.2))

vgg_model.add(Flatten())
vgg_model.add(Dense(256, activation='relu'))
vgg_model.add(Dense(128, activation='relu'))
vgg_model.add(Dense(2, activation='softmax'))

vgg_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      4160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
dropout (Dropout)            (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 128)     8320
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     16512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 256)       33024
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       65792
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       65792
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 512)       131584
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       262656
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       262656
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 512)       262656
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       262656
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       262656
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 8,094,658
Trainable params: 8,094,658
Non-trainable params: 0
_________________________________________________________________
```

Model Training

```
vgg_model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics="accuracy")
history = vgg_model.fit(train,
                    epochs=10
                    )
```

```
Epoch 1/10
27/27 [==============================] - 9s 341ms/step - loss: 0.6934 - accuracy: 0.5629
Epoch 2/10
27/27 [==============================] - 9s 318ms/step - loss: 0.6882 - accuracy: 0.5749
Epoch 3/10
27/27 [==============================] - 8s 313ms/step - loss: 0.6828 - accuracy: 0.5749
Epoch 4/10
27/27 [==============================] - 8s 311ms/step - loss: 0.6873 - accuracy: 0.5749
Epoch 5/10
27/27 [==============================] - 9s 322ms/step - loss: 0.6824 - accuracy: 0.5749
Epoch 6/10
27/27 [==============================] - 8s 315ms/step - loss: 0.6821 - accuracy: 0.5749
Epoch 7/10
27/27 [==============================] - 8s 310ms/step - loss: 0.6826 - accuracy: 0.5749
Epoch 8/10
27/27 [==============================] - 8s 310ms/step - loss: 0.6845 - accuracy: 0.5749
Epoch 9/10
27/27 [==============================] - 8s 311ms/step - loss: 0.6823 - accuracy: 0.5749
Epoch 10/10
27/27 [==============================] - 8s 309ms/step - loss: 0.6831 - accuracy: 0.5749
```

```
#Creating an array of predicted test images

vgg_predictions = vgg_model.predict(test)
```

```
scores = vgg_model.evaluate(test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
6/6 [==============================] - 1s 207ms/step - loss: 0.6819 - accuracy: 0.5749
Test loss: 0.6819178462028503
Test accuracy: 0.57485032081604
```

We trained the model for only 10 epochs as we cans see that the model performance was not increasing and it managed to achieve only 57% accuracy.This happened due to less training data and originally VGG16 was trained on a very large dataset and for a very long time.Model performance can be increased on a larger dataset.

Save our model

```
# We are saving our model so that we can compile it later with DeepC Compiler
vgg_model.save("real_vs_fake.h5")
```

Loading original VGG16 pretrained Model from Keras

- VGG16 was originally trained on imagenet dataset but we will be using transfer learning to use VGG16 on our dataset

```
vgg16_model = tf.keras.applications.vgg16.VGG16(include_top=False, weights="imagenet", input_shape=(224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_nc
58892288/58889256 [==============================] - 1s 0us/step
```

```
# Viewing the last convolutional layer output shape
vgg16_model.output[-1]
```

```
<tf.Tensor 'strided_slice:0' shape=(7, 7, 512) dtype=float32>
```

```
model = Sequential([vgg16_model,
                    GlobalAveragePooling2D(),
                    Dense(512, activation = "relu"),
                    BatchNormalization(),
                    Dense(128, activation = "relu"),

                    Dense(2, activation = "softmax")])
# We will be training only our dense layers not the entire VGG16 model
model.layers[0].trainable = False

model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics="accuracy")

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 258
=================================================================
Total params: 15,045,314
Trainable params: 329,602
Non-trainable params: 14,715,712
_________________________________________________________________
```

Model Training

```
history =  model.fit(train,
                     epochs=15
                     )
```

```
Epoch 1/15
27/27 [==============================] - 9s 334ms/step - loss: 0.8334 - accuracy: 0.5605
Epoch 2/15
27/27 [==============================] - 8s 298ms/step - loss: 0.5703 - accuracy: 0.6910
Epoch 3/15
27/27 [==============================] - 8s 297ms/step - loss: 0.4913 - accuracy: 0.7581
Epoch 4/15
27/27 [==============================] - 8s 297ms/step - loss: 0.3976 - accuracy: 0.8395
Epoch 5/15
27/27 [==============================] - 8s 298ms/step - loss: 0.3572 - accuracy: 0.8503
Epoch 6/15
27/27 [==============================] - 8s 298ms/step - loss: 0.2969 - accuracy: 0.8946
Epoch 7/15
27/27 [==============================] - 8s 295ms/step - loss: 0.2719 - accuracy: 0.9042
Epoch 8/15
27/27 [==============================] - 8s 298ms/step - loss: 0.2413 - accuracy: 0.9162
Epoch 9/15
27/27 [==============================] - 8s 296ms/step - loss: 0.1952 - accuracy: 0.9437
Epoch 10/15
27/27 [==============================] - 8s 299ms/step - loss: 0.1998 - accuracy: 0.9377
Epoch 11/15
27/27 [==============================] - 8s 297ms/step - loss: 0.1560 - accuracy: 0.9581
Epoch 12/15
27/27 [==============================] - 8s 297ms/step - loss: 0.1536 - accuracy: 0.9557
Epoch 13/15
27/27 [==============================] - 8s 297ms/step - loss: 0.1493 - accuracy: 0.9569
Epoch 14/15
27/27 [==============================] - 8s 298ms/step - loss: 0.1681 - accuracy: 0.9449
Epoch 15/15
27/27 [==============================] - 8s 295ms/step - loss: 0.1293 - accuracy: 0.9665
```

```
model.save("real_vs_fake_vgg.h5")
```

Predictions

```
#Creating an array of predicted test images

predictions = model.predict(test)
```

```
temp = model.predict(x)
```

```
temp
```

```
array([[0.9154715 , 0.08452854],
       [0.8819129 , 0.11808711],
       [0.6422919 , 0.35770807]], dtype=float32)
```

```
pr = np.argmax(temp[[2]])
pr
```

```
0
```

As we can see by using transfer learning we obtained around 97% accuracy.

```
scores = model.evaluate(test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
6/6 [==============================] - 1s 202ms/step - loss: 0.2369 - accuracy: 0.9401
Test loss: 0.23689377307891846
Test accuracy: 0.940119743347168
```

Accessing Model Performance

```
test_path = "realVSfake/real_and_fake_face/"

plt.figure(figsize=(15,15))

start_index = 90

for i in range(16):
    plt.subplot(4,4, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    preds = np.argmax(predictions[[start_index+i]])

    gt1 = test.filenames[start_index+i][9:13]


    if gt == "fake":
        gt = 0
    else:
        gt = 1

    if preds != gt:
        col ="r"
        gt1 = "Wrongly Predicted"
    else:
        col = "g"
        gt1 = 'Correctly Predicted'

    plt.xlabel('i={}, pred={}, Status={}'.format(start_index+i,preds,gt1),color=col)
    plt.imshow(load_img(test_path+test.filenames[start_index+i]))
    #print(test_path+test.filenames[start_index+i])
    plt.tight_layout()

plt.show()
```

png

```
x = 'realVSfake/real_and_fake_face/training_fake/hard_90_1100.jpg'

x = tf.keras.preprocessing.image.load_img(
    x, grayscale=False, color_mode="rgb", interpolation="nearest", target_size=(224,224),
)

input_arr = keras.preprocessing.image.img_to_array(x)
input_arr = np.array([input_arr])  # Convert single image to a batch.
#input_arr = input_arr.reshape(224,224,3)
#predictions = model.predict(input_arr)
```

```
input_arr.shape
```

```
(1, 224, 224, 3)
```

```
demo = model.predict(input_arr)
demo
```

```
array([[0., 1.]], dtype=float32)
```

Compiling our model which we saved using DeepC Compiler

```
!deepCC real_vs_fake.h5
```

```
reading [keras model] from 'real_vs_fake.h5'
Saved 'real_vs_fake.onnx'
reading onnx model from file  real_vs_fake.onnx
Model info:
  ir_vesion :  4
  doc       :
WARN (ONNX): terminal (input/output) conv2d_input's shape is less than 1.
             changing it to 1.
WARN (ONNX): terminal (input/output) dense_2's shape is less than 1.
             changing it to 1.
WARN (GRAPH): found operator node with the same name (dense_2) as io node.
running DNNC graph sanity check ... passed.
Writing C++ file  real_vs_fake_deepC/real_vs_fake.cpp
INFO (ONNX): model files are ready in dir real_vs_fake_deepC
g++ -std=c++11 -O3 -I. -I/opt/tljh/user/lib/python3.7/site-packages/deepC-0.13-py3.7-linux-x86_64.egg/deepC/include -isystem /opt/tl
Model executable  real_vs_fake_deepC/real_vs_fake.exe
```