```
# Imports

import numpy as np  # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
import shutil # high-level operations on files
from tqdm import tqdm # Progress bar and status logging
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report,confusion_matrix

import cv2 # computer vision algorithms

# Importing the Keras libraries and packages
from keras import utils
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
```

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=Tru
```

```
# Configuration

DATASET_DIR = '/content/gdrive/MyDrive/real_and_fake_face'
TRAIN_DIR = '/content/gdrive/MyDrive/train_dataset'
TEST_DIR = '/content/gdrive/MyDrive/test_dataset'

RATE = 0.2 # splitting proportion for training and test datasets

# Parameters for Grid Search

N_EPOCHS = [20] #[20, 40, 100, 200]
OPTIMIZERS = ['adam'] #['adam', 'rmsprop', 'SGD']
DROPOUT_RATES =  [0.1, 0.2, 0.4]
LOSS_FUNCTIONS = ['binary_crossentropy']  #['sparse_categorical_crossentropy', 'kullback_leibler_divergence']
```

```
os.mkdir(TRAIN_DIR)
os.mkdir(TRAIN_DIR+'/fake')
os.mkdir(TRAIN_DIR+'/real')

os.mkdir(TEST_DIR)
os.mkdir(TEST_DIR+'/fake')
os.mkdir(TEST_DIR+'/real')
```

Updated folder structure:

```
# Split folders with images into training, validation and test folders.
# OPTION 1 (using split-folders)

#pip install split-folders

#import split_folders

# Split with a ratio.
# To only split into training and validation set, set a tuple to `ratio`, i.e, `(.8, .2)`.
#split_folders.ratio('input_folder', output="output", seed=1337, ratio=(.8, .1, .1)) # default values
```

```
# Split image files into test and training set
# OPTION 2 (copying files into newly created folders)
files_real = os.listdir(f'{DATASET_DIR}/training_real')
```

```
files_fake = os.listdir(f'{DATASET_DIR}/training_fake')


# sample from each class to create a test set
np.random.seed(0)
files_real_test = np.random.choice(
    files_real,
    size=round(len(files_real) * RATE),
    replace=False,
    p=None)

files_real_train = list(set(files_real) - set(files_real_test)) #[file for file in files_real if file not in files_real_test]

files_fake_test = np.random.choice(
    files_fake,
    size=round(len(files_fake) * RATE),
    replace=False,
    p=None)

files_fake_train = list(set(files_fake) - set(files_fake_test)) #[file for file in files_fake if file not in files_fake_test]

for file in files_real_train:
    shutil.copyfile(DATASET_DIR+'/training_real/'+file, TRAIN_DIR+'/real/'+file)

for file in files_fake_train:
    shutil.copyfile(DATASET_DIR+'/training_fake/'+file, TRAIN_DIR+'/fake/'+file)

for file in files_real_test:
    shutil.copyfile(DATASET_DIR+'/training_real/'+file, TEST_DIR+'/real/'+file)

for file in files_fake_test:
    shutil.copyfile(DATASET_DIR+'/training_fake/'+file, TEST_DIR+'/fake/'+file)
```

```
train_samples = sum([len(files) for r, d, files in os.walk(TRAIN_DIR)])
test_samples = sum([len(files) for r, d, files in os.walk(TEST_DIR)])
print('Number of training images: {} \nNumber of test images: {}'.format(train_samples, test_samples))
```

```
Number of training images: 1633
Number of test images: 408
```

```
# load and show an image with Pillow
# from PIL import Image
# image = Image.open('/kaggle/test_dataset/fake/hard_39_1111.jpg')
# # some details about the image
# print(image.format)
# print(image.mode)
# print(image.size)
```

```
def get_images(path, img_shape=(64, 64)):

    '''
    Returns a np array of images and labels from path
    Images must be stored in path/class1, path/class2
    '''
    main_path = path
    y = []
    list = [name for name in os.listdir(main_path) if os.path.isdir(os.path.join(main_path, name))]
    print(list)
    image_collection = []
    for idx,folder in enumerate(list):

        label = idx

        sub_list = sorted(os.listdir(os.path.join(main_path,folder)))

        for i in tqdm(range(1, len(sub_list))):
            image_path = os.path.join(main_path, folder, sub_list[i])
            read_image = cv2.imread(image_path)
            image_resized = cv2.resize(read_image, img_shape, interpolation=cv2.INTER_AREA)

            image = np.float32(image_resized)
            image = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F) #Change alpha, beta acc

            image_collection.append(image)
            y.append(label)

    y = np.array(y)
    y = tf.keras.utils.to_categorical(y,num_classes=len(list))

    return image_collection, y[:,0]
```

```
import tensorflow as tf
# Preparing test and trainng datasets
X_train,y_train = get_images(TRAIN_DIR,img_shape=(64,64))
X_test,y_test = get_images(TEST_DIR,img_shape=(64,64))
X_train = np.array(X_train)
X_test = np.array(X_test)
# print(X_train.shape)
# print(X_train[0])
# from PIL import Image
# im = Image.fromarray(X_train[0].astype('uint8'))
# im.save("img50.jpg")
```

```
['fake', 'real']
100%|██████████| 767/767 [00:17<00:00, 43.62it/s]
100%|██████████| 864/864 [00:18<00:00, 47.71it/s]
['fake', 'real']
100%|██████████| 191/191 [00:03<00:00, 48.06it/s]
100%|██████████| 215/215 [00:04<00:00, 48.18it/s]
```

```
print('Training set', X_train.shape)
print('Test set', X_test.shape)
```

```
Training set (1631, 64, 64, 3)
Test set (406, 64, 64, 3)
```

We don't have too much data to train the network. One of possible workarounds is to use ImageDataGenerator. On the one hand, it does allow us to generate additional examples. On the other hand, all of these examples are based on a too small dataset and the network still cannot generalize to data it was never trained on

```
#Shuffle training examples
X_train, y_train = shuffle(X_train, y_train)
```

```
def build_classifier(optimizer, dropout, loss):
    classifier = Sequential() # Initialising the CNN
    classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Dropout(dropout))
    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Dropout(dropout))
    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Dropout(dropout))
    classifier.add(Flatten())
    classifier.add(Dense(units = 128, activation = 'relu'))
    classifier.add(Dense(units = 1, activation = 'sigmoid')) #'tanh'))

    classifier.compile(optimizer = optimizer, loss = loss, metrics = ['accuracy'])

    return classifier

classifier = KerasClassifier(build_fn = build_classifier)

grid_parameters = {'epochs': N_EPOCHS,
                  'optimizer': OPTIMIZERS,
                  'dropout': DROPOUT_RATES,
                  'loss':LOSS_FUNCTIONS
                  }


grid_search = GridSearchCV(estimator = classifier,
                           param_grid = grid_parameters,
                           scoring = 'accuracy',
                           cv = 10)
```

```
grid_search = grid_search.fit(X_train, y_train, verbose=0)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: DeprecationWarning: KerasClassifier is deprecated, use Sci-Ker
```

```
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
print(best_parameters)
print(best_accuracy)
```

```
{'dropout': 0.2, 'epochs': 20, 'loss': 'binary_crossentropy', 'optimizer': 'adam'}
0.6210683824629658
```

New Section

```
predicted = grid_search.predict(X_test)
```

```
print('Confusion matrix for training set:')
print(confusion_matrix(y_train,grid_search.predict(X_train)))
print('\n')
print(classification_report(y_train,grid_search.predict(X_train)))

print('Confusion matrix  for test set:')
print(confusion_matrix(y_test,predicted))
print('\n')
print(classification_report(y_test,predicted))
```

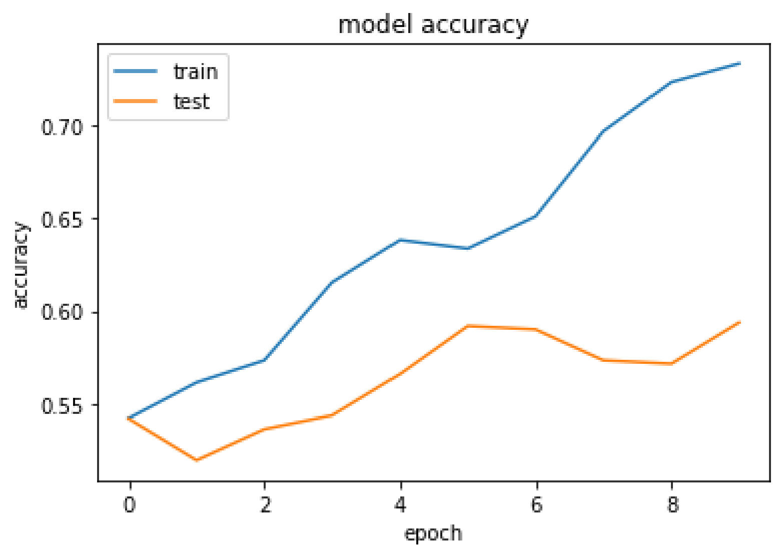
```
Confusion matrix for training set:
[[725 139]
 [266 501]]


              precision    recall  f1-score   support

         0.0       0.73      0.84      0.78       864
         1.0       0.78      0.65      0.71       767

    accuracy                           0.75      1631
   macro avg       0.76      0.75      0.75      1631
weighted avg       0.76      0.75      0.75      1631

Confusion matrix  for test set:
[[144  71]
 [ 89 102]]


              precision    recall  f1-score   support

         0.0       0.62      0.67      0.64       215
         1.0       0.59      0.53      0.56       191

    accuracy                           0.61       406
   macro avg       0.60      0.60      0.60       406
weighted avg       0.60      0.61      0.60       406
```

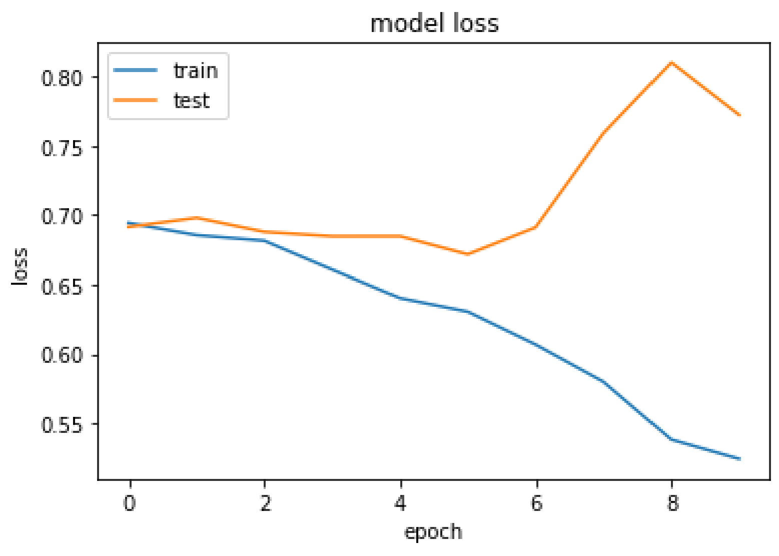
```
import matplotlib.pyplot as plt
import numpy
def history():
    classifier = Sequential() # Initialising the CNN
    classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    classifier.add(Flatten())
    classifier.add(Dense(units = 128, activation = 'relu'))
    classifier.add(Dense(units = 1, activation = 'sigmoid')) #'tanh'))
    classifier.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = classifier.fit(X_train, y_train, validation_split=0.33, epochs=10, batch_size=10, verbose=0)

    return history

plot=history()



print(plot.history['loss'])
```

```
[0.6942959427833557, 0.685585618019104, 0.681553065776825, 0.6608260869979858, 0.6402154564857483, 0.630456268787384, 0.606754€
```

```
print(plot.history.keys())
# summarize history for accuracy
plt.plot(plot.history['accuracy'])
plt.plot(plot.history['val_accuracy'])
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(plot.history['loss'])
plt.plot(plot.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





```
import cv2
from google.colab.patches import cv2_imshow



cap = cv2.VideoCapture(0)

if cap.isOpened():
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    res=(int(width), int(height))
    # this format fail to play in Chrome/Win10/Colab
    # fourcc = cv2.VideoWriter_fourcc(*'MP4V') #codec
    fourcc = cv2.VideoWriter_fourcc(*'H264') #codec
    out = cv2.VideoWriter('output.mp4', fourcc, 20.0, res)
```

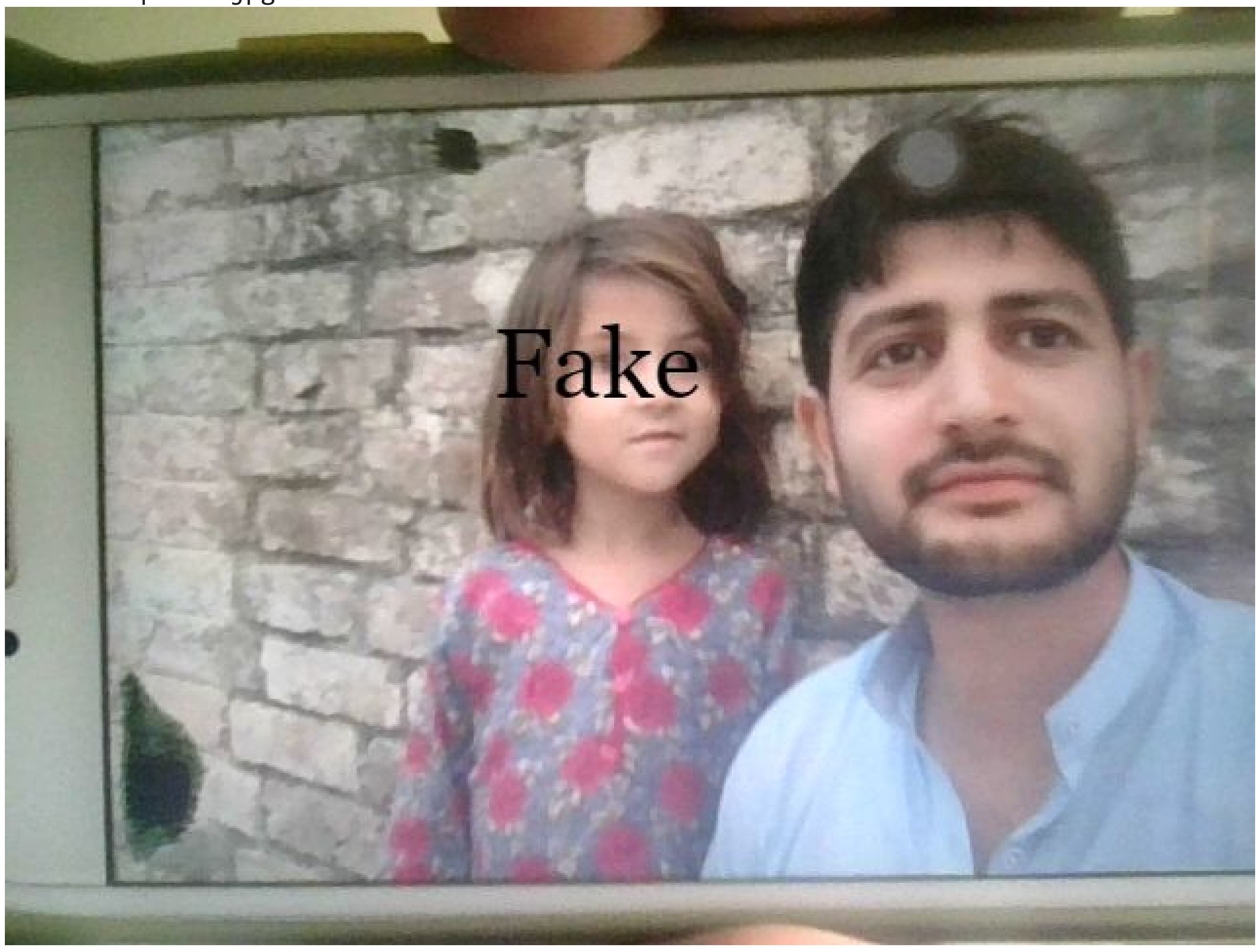
```
        frame = None
        while True:
            try:
                is_success, frame = cap.read()
            except cv2.error:
                continue

            if not is_success:
                break

            # OPTIONAL: do some processing

            # convert cv2 BGR format to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            out.write(image)

        out.release()

        # OPTIONAL: show last image
        if frame:
          cv2_imshow(frame)

    cap.release()


    from IPython.display import display, Javascript
    from google.colab.output import eval_js
    from base64 import b64decode


    def take_photo(filename='photo.jpg', quality=0.8):
      js = Javascript('''
        async function takePhoto(quality) {
          const mood = [0,1,2];
          const div = document.createElement('div');
          const capture = document.createElement('button');
          capture.textContent = 'Capture';
          div.appendChild(capture);

          const video = document.createElement('video');
          video.style.display = 'block';
          const stream = await navigator.mediaDevices.getUserMedia({video: true});

          document.body.appendChild(div);
          div.appendChild(video);
          video.srcObject = stream;
          await video.play();
          // Resize the output to fit the video element.
          google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);
          const pc = Math.floor(Math.random() * mood.length);
          // Wait for Capture to be clicked.
          await new Promise((resolve) => capture.onclick = resolve);
          const canvas = document.createElement('canvas');
          canvas.width = video.videoWidth;
          canvas.height = video.videoHeight;
          canvas.getContext('2d').drawImage(video, 0, 0);
          if (pc==0){
          z="Fake"
          }
          else if(pc==1){
          z="Real"
          }
          else{
          z="No face"
          }
          canvas.getContext('2d').font = "50px Georgia";;
          canvas.getContext('2d').fillText(z, 250, 200);

          stream.getVideoTracks()[0].stop();
          div.remove();
          return canvas.toDataURL('image/jpeg', quality);
        }
        ''')
      display(js)
      data = eval_js('takePhoto({})'.format(quality))
      binary = b64decode(data.split(',')[1])
      with open(filename, 'wb') as f:
```

```
      f.write(binary)
    return filename
```

Webcam test

```
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  print(str(err))
```

```
Saved to photo.jpg
```



✓ 11s completed at 5:29 PM