```
pip install gradio
```

```
Collecting backoff~=1.10.0
  Downloading backoff-1.10.0-py2.py3-none-any.whl (31 kB)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Collecting pydantic!=1.7,!=1.7.1,!=1.7.2,!=1.7.3,!=1.8,!=1.8.1,<2.0.0,>=1.6.2
  Downloading pydantic-1.9.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.wh
     |████████████████████████████████| 11.1 MB 37.5 MB/s
Collecting starlette==0.19.1
  Downloading starlette-0.19.1-py3-none-any.whl (63 kB)
     |████████████████████████████████| 63 kB 2.0 MB/s
Collecting anyio<5,>=3.4.0
  Downloading anyio-3.6.1-py3-none-any.whl (80 kB)
     |████████████████████████████████| 80 kB 7.0 MB/s
Collecting sniffio>=1.1
  Downloading sniffio-1.2.0-py3-none-any.whl (10 kB)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-pack
Collecting mdurl~=0.1
  Downloading mdurl-0.1.1-py3-none-any.whl (10 kB)
Collecting linkify-it-py~=1.0
  Downloading linkify_it_py-1.0.3-py3-none-any.whl (19 kB)
Collecting mdit-py-plugins
  Downloading mdit_py_plugins-0.3.0-py3-none-any.whl (43 kB)
     |████████████████████████████████| 43 kB 1.4 MB/s
Collecting uc-micro-py
  Downloading uc_micro_py-1.0.1-py3-none-any.whl (6.2 kB)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Collecting bcrypt>=3.1.3
  Downloading bcrypt-3.2.2-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.manyl
     |████████████████████████████████| 62 kB 937 kB/s
Collecting cryptography>=2.5
  Downloading cryptography-37.0.2-cp36-abi3-manylinux_2_24_x86_64.whl (4.0 MB)
     |████████████████████████████████| 4.0 MB 49.2 MB/s
Collecting pynacl>=1.0.1
  Downloading PyNaCl-1.5.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.manyl
     |████████████████████████████████| 856 kB 46.2 MB/s
Requirement already satisfied: cffi>=1.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (f
Collecting h11>=0.8
  Downloading h11-0.13.0-py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 3.3 MB/s
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.7/dist-packages (
Collecting asgiref>=3.4.0

  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Building wheels for collected packages: ffmpy, python-multipart
  Building wheel for ffmpy (setup.py) ... done
  Created wheel for ffmpy: filename=ffmpy-0.3.0-py3-none-any.whl size=4712 sha256=a3f
  Stored in directory: /root/.cache/pip/wheels/13/e4/6c/e8059816e86796a597c6e6b0d4c88
  Building wheel for python-multipart (setup.py) ... done
  Created wheel for python-multipart: filename=python_multipart-0.0.5-py3-none-any.wh
  Stored in directory: /root/.cache/pip/wheels/2c/41/7c/bfd1c180534ffdcc0972f78c5758f
Successfully built ffmpy python-multipart
Installing collected packages: sniffio, mdurl, uc-micro-py, multidict, markdown-it-py
Successfully installed aiohttp-3.8.1 aiosignal-1.2.0 analytics-python-1.4.0 anyio-3.6
```

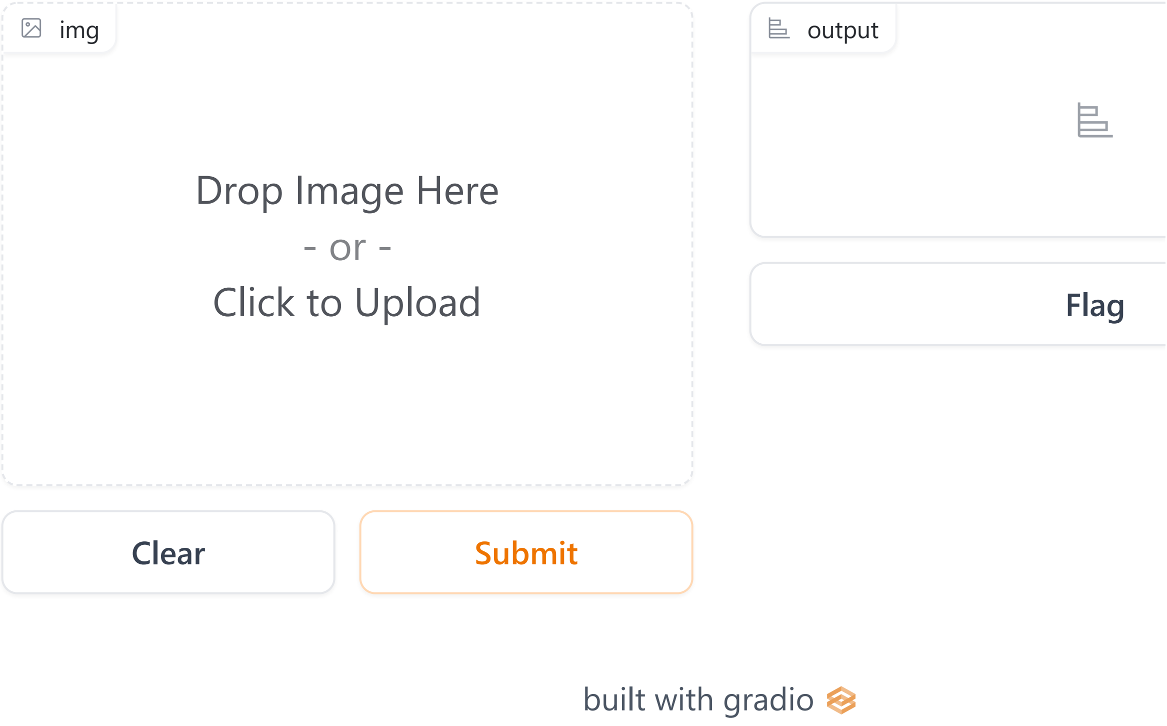
```
pip install timm
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
Collecting timm
  Downloading timm-0.5.4-py3-none-any.whl (431 kB)
     |████████████████████████████████| 431 kB 4.9 MB/s
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: torch>=1.4 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from tor
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Installing collected packages: timm
Successfully installed timm-0.5.4
```

Admin panel

```
import requests

import gradio as gr
import torch
from timm import create_model
from timm.data import resolve_data_config
from timm.data.transforms_factory import create_transform

IMAGENET_1k_URL = "https://storage.googleapis.com/bit_models/ilsvrc2012_wordnet_lemmas.txt"
LABELS = requests.get(IMAGENET_1k_URL).text.strip().split('\n')
model = create_model('resnet50', pretrained=True)

transform = create_transform(
    **resolve_data_config({}, model=model)
)
model.eval()
def predict_fn(img):
    img = img.convert('RGB')
    print(img)
    img = transform(img).unsqueeze(0)

    with torch.no_grad():
        out = model(img)
    probabilites = torch.nn.functional.softmax(out[0], dim=0)

    values, indices = torch.topk(probabilites, k=5)

    return {LABELS[i]: v.item() for i, v in zip(indices, values)}

gr.Interface(predict_fn, gr.inputs.Image(type='pil'), outputs='label').launch()
```

```
Downloading: "https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-r
/usr/local/lib/python3.7/dist-packages/gradio/deprecation.py:40: UserWarning: `optional`
  warnings.warn(value)
Colab notebook detected. To show errors in colab notebook, set `debug=True` in `launch()
Running on public URL: https://15907.gradio.app

This share link expires in 72 hours. For free permanent hosting, check out Spaces (https
```



```
(<gradio.routes.App at 0x7ffaa933fb90>,
 'http://127.0.0.1:7860/',
 'https://15907.gradio.app')
```

```
print(len(LABELS))
```

```
1000
```

```
# 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')
```

```
     Mounted at /content/gdrive
```

```
# Configuration

DATASET_DIR = '/content/gdrive/MyDrive/CelebA_Spoof'
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_leib
```

```
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)) # defaul
```

```
# 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

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

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)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-4-5ad04c617c33> in <module>()
      1 # Split image files into test and training set
      2 # OPTION 2 (copying files into newly created folders)
----> 3 files_real = os.listdir(f'{DATASET_DIR}/training_real')
      4 files_fake = os.listdir(f'{DATASET_DIR}/training_fake')
      5

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
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_
```

```
Number of training images: 3694
Number of test images: 923
```

```
# 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, n
    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)
            try:
              image_resized = cv2.resize(read_image, img_shape, interpolation=cv2.INTER_AREA)
            except:
              continue
            image = np.float32(image_resized)
            image = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dt

            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%|██████████| 2595/2595 [00:44<00:00, 58.08it/s]
100%|██████████| 1097/1097 [00:38<00:00, 28.19it/s]
['fake', 'real']
100%|██████████| 648/648 [00:19<00:00, 32.77it/s]
100%|██████████| 273/273 [00:09<00:00, 28.12it/s]
```

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

```
Training set (3690, 64, 64, 3)
Test set (921, 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: 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'}
0.9655826558265582
```

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:
[[1091    5]
 [   1 2593]]


              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1096
         1.0       1.00      1.00      1.00      2594

    accuracy                           1.00      3690
   macro avg       1.00      1.00      1.00      3690
weighted avg       1.00      1.00      1.00      3690

Confusion matrix  for test set:
[[259  14]
 [  5 643]]


              precision    recall  f1-score   support

         0.0       0.98      0.95      0.96       273
         1.0       0.98      0.99      0.99       648
```

```
        accuracy                           0.98       921
       macro avg       0.98      0.97      0.98       921
    weighted avg       0.98      0.98      0.98       921
```

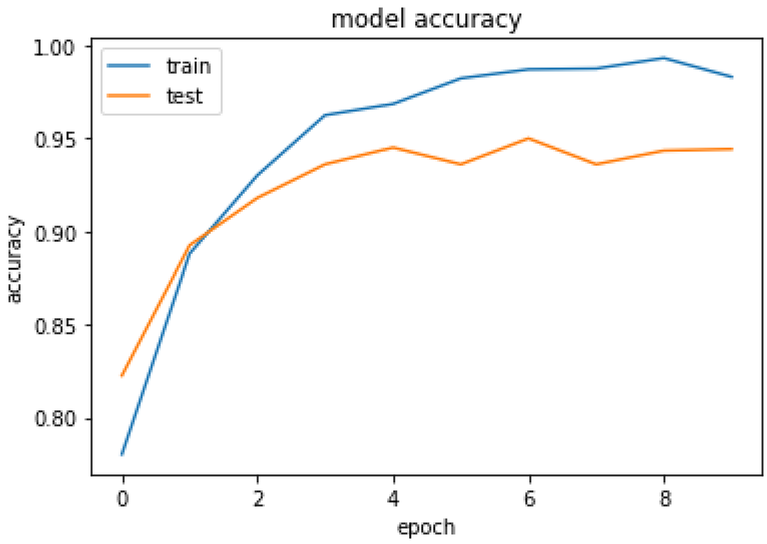
```
import matplotlib.pyplot as plt
import numpy
def history():
    classifier = Sequential() # Initialising the CNN
    classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    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=1

    return history

plot=history()
```

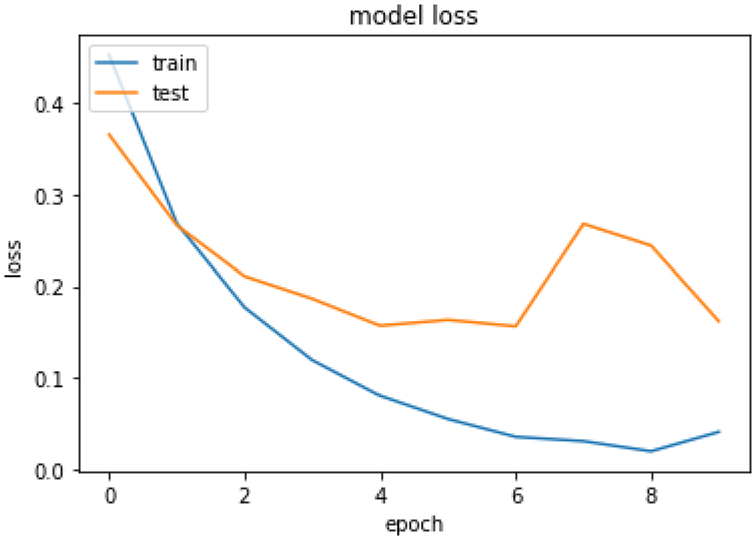
```
print(plot.history['loss'])
```

```
[0.45137330889701843, 0.268157035112381, 0.1771579533815384, 0.11977747082710266, 0.080
```

```
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)
```

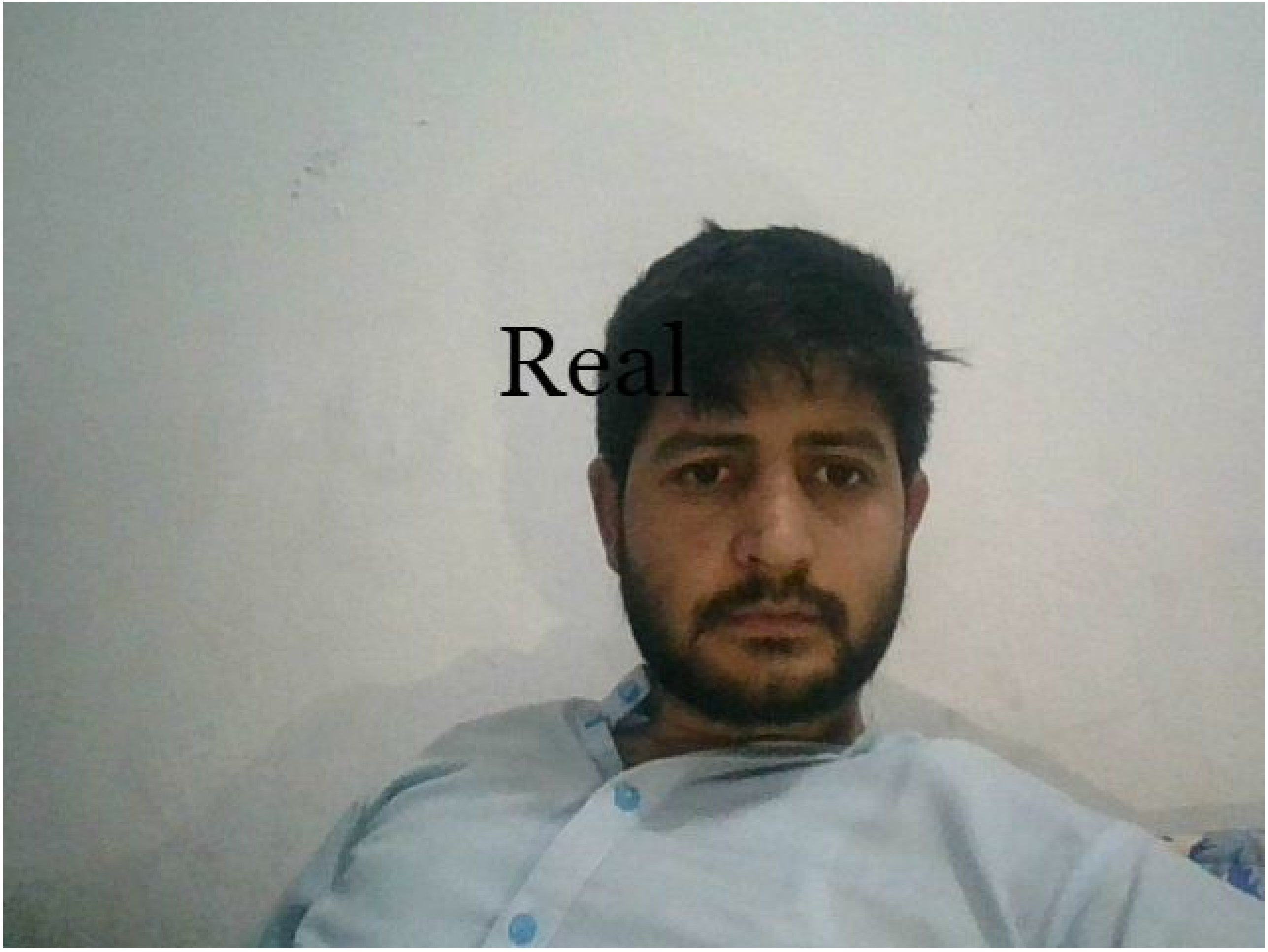
```
  frame = None
  while True:
    try:
      is_success, frame = cap.read()
    except cv2.error:
      continue

    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);
```

```
      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
```



✓ 20s completed at 10:48 PM