

pip install gradio

```

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from analytics-python->gradio) (1.15.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->gradio) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->gradio) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->gradio) (1.25.11)
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-1.9.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.9 MB)
    |████████████████████████████████████████| 10.9 MB 50.2 MB/s
Collecting starlette==0.19.1
  Downloading starlette-0.19.1-py3-none-any.whl (63 kB)
    |████████████████████████████████████████| 63 kB 1.4 MB/s
Collecting anyio<5,>=3.4.0
  Downloading anyio-3.6.1-py3-none-any.whl (80 kB)
    |████████████████████████████████████████| 80 kB 10.6 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-packages (from Jinja2->gradio) (2.0.1)
Collecting mdurl~0.1
  Downloading mdurl-0.1.1-py3-none-any.whl (10 kB)
Collecting mdit-py-plugins
  Downloading mdit_py_plugins-0.3.0-py3-none-any.whl (43 kB)
    |████████████████████████████████████████| 43 kB 2.3 MB/s
Collecting linkify-it-py~1.0
  Downloading linkify_it_py-1.0.3-py3-none-any.whl (19 kB)
Collecting uc-micro-py
  Downloading uc_micro_py-1.0.1-py3-none-any.whl (6.2 kB)
Requirement already satisfied: cyclr>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->gradio) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->gradio) (1.4.2)
Requirement already satisfied: pyparsing!=2.0.4,!2.1.2,!2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->gradio) (3.0.9)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->gradio) (2022.1)
Collecting cryptography>=2.5
  Downloading cryptography-37.0.2-cp36-abi3-manylinux_2_24_x86_64.whl (4.0 MB)
    |████████████████████████████████████████| 4.0 MB 47.2 MB/s
Collecting pynacl>=1.0.1
  Downloading PyNaCl-1.5.0-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (856 kB)
    |████████████████████████████████████████| 856 kB 56.0 MB/s
Collecting bcrypt>=3.1.3
  Downloading bcrypt-3.2.2-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_24_x86_64.whl (62 kB)
    |████████████████████████████████████████| 62 kB 859 kB/s
Requirement already satisfied: cffi>=1.1 in /usr/local/lib/python3.7/dist-packages (from bcrypt>=3.1.3->paramiko->gradio) (1.15.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.1->bcrypt>=3.1.3->paramiko->gradio) (2.21)
Collecting asgiref>=3.4.0
  Downloading asgiref-3.5.1-py3-none-any.whl (22 kB)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.7/dist-packages (from uvicorn->gradio) (7.1.2)
Collecting h11>=0.8
  Downloading h11-0.13.0-py3-none-any.whl (58 kB)
    |████████████████████████████████████████| 58 kB 6.2 MB/s
Building wheels for collected packages: ffmpeg, python-multipart
  Building wheel for ffmpeg (setup.py) ... done
  Created wheel for ffmpeg: filename=ffmpeg-0.3.0-py3-none-any.whl size=4712 sha256=c97285cd8dad11945dc60ff235a3a723c51e286ef5
  Stored in directory: /root/.cache/pip/wheels/13/e4/6c/e8059816e86796a597c6e6b0d4c880630f51a1fcfa0befd5e6
  Building wheel for python-multipart (setup.py) ... done
  Created wheel for python-multipart: filename=python_multipart-0.0.5-py3-none-any.whl size=31678 sha256=de5aacfcc71d48aa4ca
  Stored in directory: /root/.cache/pip/wheels/2c/41/7c/bfd1c180534ffdc0972f78c5758f89881602175d48a8bcd2c
Successfully built ffmpeg python-multipart
Installing collected packages: sniffio, mdurl, uc-micro-py, multidict, markdown-it-py, frozenlist, anyio, yarl, starlette, p
Successfully installed aiohttp-3.8.1 aiosignal-1.2.0 analytics-python-1.4.0 anyio-3.6.1 asgiref-3.5.1 async-timeout-4.0.2 as

```

pip install timm

```

Collecting timm
  Downloading timm-0.5.4-py3-none-any.whl (431 kB)
    |████████████████████████████████████████| 431 kB 28.0 MB/s
Requirement already satisfied: torch>=1.4 in /usr/local/lib/python3.7/dist-packages (from timm) (1.11.0+cu113)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from timm) (0.12.0+cu113)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch>=1.4->timm) (4.2.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from torchvision->timm) (2.23.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision->timm) (7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision->timm) (1.21.6)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->timm) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->timm) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->timm) (1.25.11)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->torchvision->timm) (2021.10.8)
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)

    probabilitites = torch.nn.functional.softmax(out[0], dim=0)


    values, indices = torch.topk(probabilitites, 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()

/usr/local/lib/python3.7/dist-packages/paramiko/transport.py:236: CryptographyDeprecationWarning: Blowfish
  "class": algorithms.Blowfish,
Downloading: "https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-rsb-weights/resnet50-196c36e0.pth"
Colab notebook detected. To show errors in colab notebook, set `debug=True` in `launch()`
Running on public URL: https://25027.gradio.app
```

This share link expires in 72 hours. For free permanent hosting, check out Spaces (<https://huggingface.co/spaces/andrewyeh/face-anti-spoofing>)



lipstick, lip_rouge 32%

bow_tie, bow-tie, bowtie 4%

perfume, essence 3%

suit, suit_of_clothes 3%

red_wine 3%

Clear

Submit

Flag

view the api 🔑 • built with gradio 🏠

```
(<fastapi.applications.FastAPI at 0x7f5a35df5110>,
 'http://127.0.0.1:7860/',
 'https://25027.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/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)

<PIL.Image.Image image mode=RGB size=600x600 at 0x7F58ECEC9890>
<PIL.Image.Image image mode=RGB size=600x600 at 0x7F58ECECC610>

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 acco

            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 training 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 [00:15<00:00, 47.94it/s]
100%|██████████| 864/864 [00:17<00:00, 48.09it/s]
['fake', 'real']
100%|██████████| 191/191 [00:03<00:00, 48.94it/s]
100%|██████████| 215/215 [00:04<00:00, 47.07it/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': 0.4, 'epochs': 20, 'loss': 'binary_crossentropy', 'optimizer': 'adam'}
0.6149745623223104
```

▼ 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:
[[634 230]
 [252 515]]
```

	precision	recall	f1-score	support
0.0	0.72	0.73	0.72	864
1.0	0.69	0.67	0.68	767
accuracy			0.70	1631
macro avg	0.70	0.70	0.70	1631
weighted avg	0.70	0.70	0.70	1631

```
Confusion matrix for test set:
[[130 85]
 [ 86 105]]
```

	precision	recall	f1-score	support
0.0	0.60	0.60	0.60	215
1.0	0.55	0.55	0.55	191
accuracy			0.58	406
macro avg	0.58	0.58	0.58	406
weighted avg	0.58	0.58	0.58	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
```

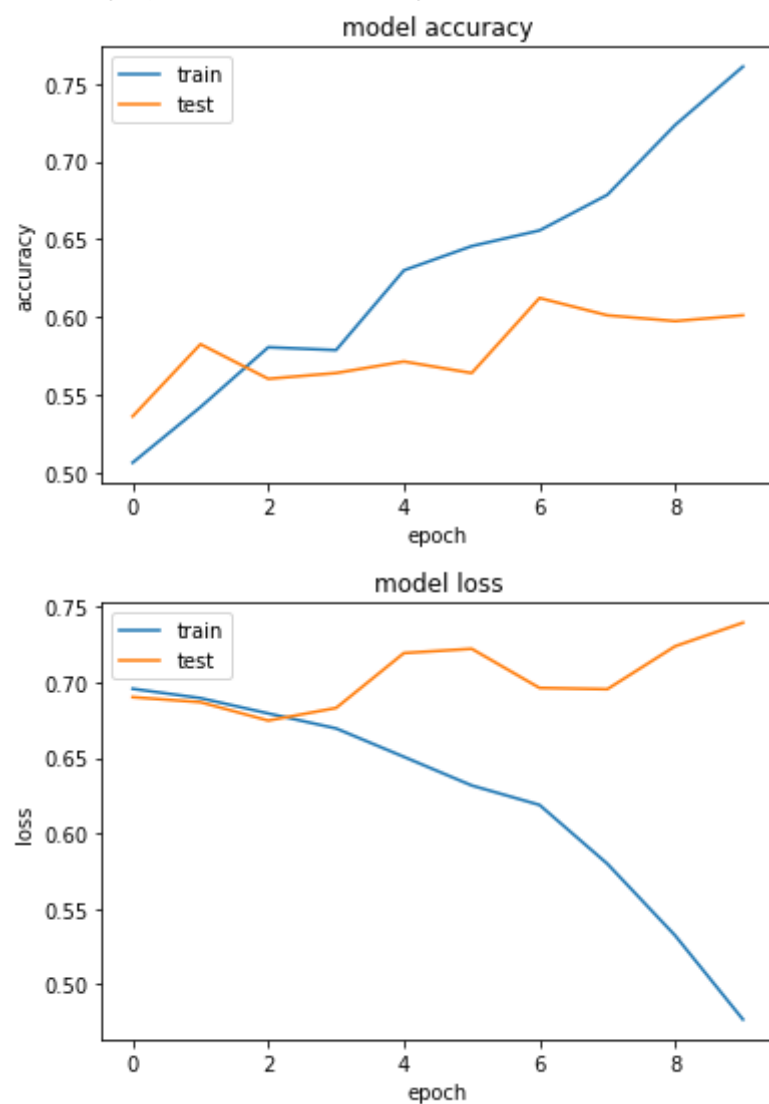
```
plot=history()
```

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

```
[0.6957947015762329, 0.6896182298660278, 0.6794621348381042, 0.6695437431335449, 0.650732696056366, 0.6317668557167053, 0.61888
```

```
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

