Importing the libraries

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import os
%matplotlib inline
```

```
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
!git clone https://github.com/laxmimerit/male-female-face-dataset.git
```

```
fatal: destination path 'male-female-face-dataset' already exists and is not an empty d
```

```
epochs = 50
lr = 1e-3
batch_size = 128
data =[]
labels = []
```



```
train_datagen = ImageDataGenerator(horizontal_flip=True,width_shift_range=0.4,height_shift_ra
                                   zoom_range=0.3,
                                   rotation_range = 20,
                                   rescale = 1/255,
                                   )
```

```
test_datagen = ImageDataGenerator(rescale = 1/255)
```

```
target_size = (size,size)
```

```
train_generator = train_datagen.flow_from_directory(
    directory='/content/male-female-face-dataset/Training',
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 47009 images belonging to 2 classes.
```

```
validation_generator = test_gen.flow_from_directory(
    directory='/content/male-female-face-dataset/Validation',
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 11649 images belonging to 2 classes.
```

```
x,y = train_generator.next()
x.shape
```

```
(128, 224, 224, 3)
```

Build ML model

```
model = Sequential()
model.add(InceptionV3(include_top=False, pooling='avg', weights ='imagenet'))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(2048, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
model.layers[0].trainable = False
model.summary()
```



```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0
_________________________________________________________________
batch_normalization_570 (Bat (None, 2048)              8192
_________________________________________________________________
dense_4 (Dense)              (None, 2048)              4196352
_________________________________________________________________
batch_normalization_571 (Bat (None, 2048)              8192
```

```
dense_5 (Dense)              (None, 1)                 2049
=================================================================
Total params: 26,017,569
Trainable params: 4,206,593
Non-trainable params: 21,810,976
_________________________________________________________________
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
len((train_generator.filenames)), batch_size, len((train_generator.filenames))//batch_size
```

```
(47009, 128, 367)
```

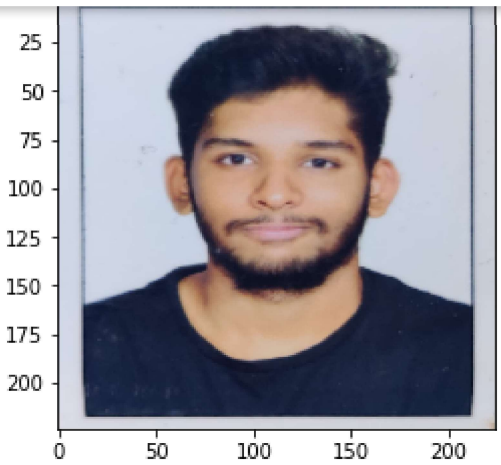
```
model.fit(train_generator,steps_per_epoch=len(train_generator.filenames)//batch_size,
          epochs=1, validation_data=validation_generator,validation_steps=len(validation_gener
```

```
367/367 [==============================] - 674s 2s/step - loss: 0.4224 - accuracy: 0.85
<keras.callbacks.History at 0x7f150ea7aa90>
```

Testing the Model

```
img_path = '/content/lmao.jpg'
```

```
img = load_img(img_path, target_size=(size, size, 3))
plt.imshow(img)
```

```
img = img_to_array(img)
```

```
img = img/255.0
```

```
img = img.reshape(1, size, size, 3)
```

```
model.predict(img)
```

```
     array([[0.99669635]], dtype=float32)
```

```
train_generator.class_indices
```

```
     {'female': 0, 'male': 1}
```

```
def get_classes(data):
  prob = model.predict(img)[0][0]

  if prob<=0.5:
    return 'female',(1-prob)
  else:
    return 'male', prob
```

Predicted whether male or female with it's corresponding accuracy

```
get_classes(img)
```

```
     ('male', 0.99669635)
```


Real-Time Prediction using Webcam

- List item
- List item

Importing the libraries for using webcam

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

Creating the working webcam functionality

```
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      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);

      // 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);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
```



```
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
```

Saving the captured image to directory

```
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:
# Errors will be thrown if the user does not have a webcam or if they do not
# grant the page permission to access it.
  print(str(err))
```

```
Saved to photo.jpg
```


```
def get_prediction(img_path):
  img = load_img(img_path, target_size=(size, size, 3))
  plt.imshow(img)
  img = img_to_array(img)
  img = img/255.0
  img = img.reshape(1, size, size, 3)
  pred, prob = get_classes (img)
  return pred, prob


get_prediction(img_path)
```

```
('male', 0.99669635)
```