```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

TO MOUNT WITH GOOGLE DRIVE

```
def prep(data_folder_path):
    dirs = os.listdir(data_folder_path)

    images = []
    labels = []

    for dir_name in dirs:
        print(dir_name)


        if dir_name=='0':

          label = 0

        elif dir_name=='1':
          label= 1


        subject_dir_path = data_folder_path + "/" + dir_name
```

```
        subject_images_names = os.listdir(subject_dir_path)

        for image_name in subject_images_names:

            if image_name.startswith("."):
                continue;


            image_path = subject_dir_path + "/" + image_name




            images.append( image_path)
            labels.append(label)



    return images,labels
```

APPENDING IMAGES

```
path = "/content/drive/My Drive/archive/Train"
train_img , train_lab = prep(path)
```

```
0
1
```

```
path = "/content/drive/My Drive/archive/test"
test_img , test_lab = prep(path)
```

```
0
1
```

```
tl = len(train_img)
train_data = []
c=0
for path in train_img:
  img = load_img(path,target_size=(224,224))
  img = img_to_array(img)
  img = preprocess_input(img)
  train_data.append(img)
  c+=1
  print(c)
```

```
t1 = len(test_img)
test_data = []
c=0
for path in test_img:
  img = load_img(path,target_size=(224,224))
  img = img_to_array(img)
  img = preprocess_input(img)
  test_data.append(img)
  c+=1
  print(c)


trainY ,testY = np.array(train_data) , np.array(test_data)
trainX , testX = np.array(train_lab) , np.array(test_lab)
```

CHANGING AS A ARRAY

```
trainY.shape
```

```
(536, 224, 224, 3)
```

```
import pickle
with open("fire.pickle","wb") as f:
  pickle.dump([trainY,trainX,testY,testX],f)
```

SAVING PRE PROCESSED IMAGES

```
aug = ImageDataGenerator(
  rotation_range=20,
  zoom_range=0.15,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.15,
  horizontal_flip=True,
  fill_mode="nearest")
```

DATA AUGUMANTATION

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
  input_tensor=Input(shape=(224, 224, 3)))
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

```
model = Model(inputs=baseModel.input, outputs=headModel)
for layer in baseModel.layers:
  layer.trainable = False
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 1
```

LAYER OF NEURAL NETWORK

```
INIT_LR = 1e-4
EPOCHS = 100
BS = 64

opt = Adam(lr=INIT_LR,decay=INIT_LR/EPOCHS)
model.compile(optimizer=opt,loss="sparse_categorical_crossentropy",metrics=["accuracy"])
```

```
from tensorflow import keras
class callbacks(keras.callbacks.Callback):
  def on_epoch_end(self,epochs,logs={}):
    if logs.get('accuracy') > 0.95 and logs.get('val_accuracy') > 0.86:
      print("\n Accuracy reached \n")
      self.model.stop_training = True
callbacks = callbacks()
```

```
H = model.fit(
    aug.flow(trainY,trainX),
    steps_per_epoch=len(trainX)//BS,
    validation_data=(testY,testX),
    validation_steps=len(testX)//BS,epochs=EPOCHS,shuffle=True,callbacks = [callbacks])
```

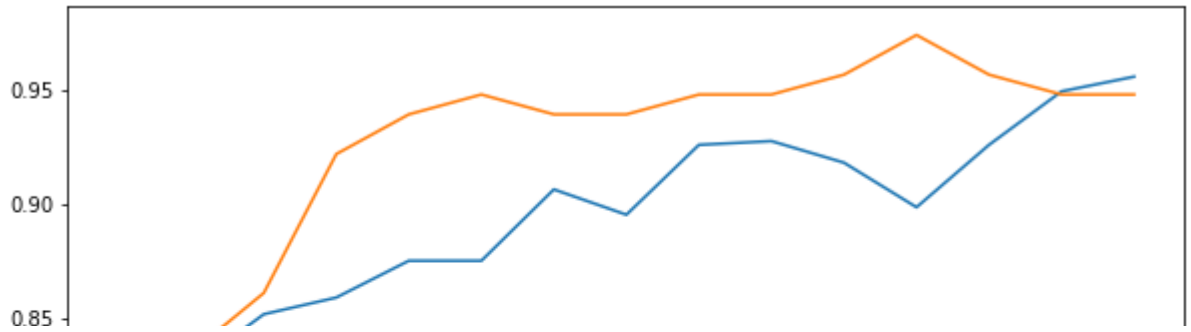
```
Epoch 1/100
8/8 [==============================] - 3s 411ms/step - loss: 0.6135 - accuracy: 0.7137 -
Epoch 2/100
8/8 [==============================] - 2s 296ms/step - loss: 0.4823 - accuracy: 0.8281 -
Epoch 3/100
8/8 [==============================] - 2s 300ms/step - loss: 0.4357 - accuracy: 0.8516 -
Epoch 4/100
8/8 [==============================] - 2s 289ms/step - loss: 0.3513 - accuracy: 0.8589 -
Epoch 5/100
8/8 [==============================] - 2s 287ms/step - loss: 0.3298 - accuracy: 0.8750 -
Epoch 6/100
8/8 [==============================] - 2s 296ms/step - loss: 0.3117 - accuracy: 0.8750 -
Epoch 7/100
```

TRAINNING

```
Epoch 9/100
```

```
acc = H.history['accuracy']
val_acc = H.history['val_accuracy']
loss = H.history['loss']
val_loss = H.history['val_loss']
epochs = range(len(acc))
plt.figure(figsize=(10,6))
plt.plot(epochs,acc)
plt.plot(epochs,val_acc)
plt.figure()
plt.plot(epochs,loss)
plt.plot(epochs,val_loss)
```

```
[<matplotlib.lines.Line2D at 0x7fb479622f98>]
```



PLOTING TRAINING AND TESTING ACCURACY AND LOSS FUCTION

```
from keras.models import model_from_json
model_json = model.to_json()
with open("model_fire_arch.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model_fire_weights.h5")
```

SAVING TRAINED MODEL

```
from tensorflow.keras.models import model_from_json
json_file = open('/content/drive/My Drive/archive/model_fire_arch.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
model.load_weights("/content/drive/My Drive/archive/model_fire_weights.h5")
```

LOADING TRAINED MODEL

```
import cv2
import time
clas = ["no fire","fire"]

#path = "/content/drive/My Drive/archive/test/1/100.jpg"
path = "/content/drive/My Drive/archive/test/0/450.jpg"
#path = "/content/drive/My Drive/archive/test/0/450.jpg"

img = load_img(path,target_size=(224,224))
img1=cv2.imread(path)
img1 =cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
#re = cv2.imread(path)
#img =cv2.cvtColor(re,cv2.COLOR_BGR2RGB)
#img = cv2.resize(img,(224,224))
img = img_to_array(img)
img = preprocess_input(img)
img = np.expand_dims(img,axis=0)
last = time.time()
```

```
pred = model.predict(img)
#print(time.time()-last)
op=clas[np.argmax(pred)]
e = np.argmax(pred)
x = round(pred[0][e]*100,2)
conf = str(x)+'%'
print(x)
print(op)

op = op + '-' + conf
if x>80.0:
    cv2.putText(img1,op,(25,50),cv2.FONT_HERSHEY_SIMPLEX,2,(0,200,0),3)
plt.imshow(img1)
```

```
92.03
no fire
<matplotlib.image.AxesImage at 0x7f302454efd0>
```

TESTING WITH THREE DIFFRENT IMAGES

```
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 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});
```

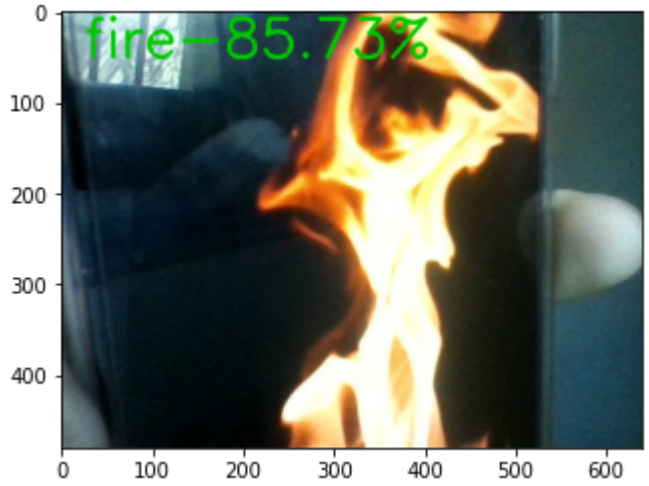
```
      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

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

FOR CAMERA

```
filename=take_photo()
import cv2
import time
clas = ["no fire","fire"]

path =filename

img = load_img(path,target_size=(224,224))
img1=cv2.imread(path)
img1 =cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
#re = cv2.imread(path)
#img =cv2.cvtColor(re,cv2.COLOR_BGR2RGB)
#img = cv2.resize(img,(224,224))
img = img_to_array(img)
img = preprocess_input(img)
img = np.expand_dims(img,axis=0)
last = time.time()
pred = model.predict(img)
#print(time.time()-last)
op=clas[np.argmax(pred)]
e = np.argmax(pred)
```

```
x = round(pred[0][e]*100,2)
conf = str(x)+'%'
print(x)
print(op)

op = op + '-' + conf
if x>80.0:
    cv2.putText(img1,op,(25,50),cv2.FONT_HERSHEY_SIMPLEX,2,(0,200,0),3)
plt.imshow(img1)
```

```
85.73
fire
<matplotlib.image.AxesImage at 0x7f30244cd748>
```



CAPTURING THE IMAGES