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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947

Enter your authorization code:
··········
Mounted at /content/drive/
```

```
import cv2,os

data_path='/content/drive/My Drive/Xử lý đa chiều/dataset/'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]

label_dict=dict(zip(categories,labels))

print(label_dict)
print(categories)
print(labels)
```

```
{'with_mask': 0, 'without_mask': 1}
['with_mask', 'without_mask']
[0, 1]
```

```
data=[]
target=[]


for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)
    print('CATEGORY = ', category)
    for img_name in img_names:
        print('image_name = ', img_name)
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #Coverting the image into gray scale
            resized=cv2.resize(gray,(64,64))
            #resizing the gray scale into 64x64, since we need a fixed common size for all
            data.append(resized)
            target.append(label_dict[category])
            #appending the image and the label(categorized) into the list (dataset)

        except Exception as e:
```

```
            print('Exception:',e)
            #if any exception rasied, the exception will be printed here. And pass to the
```

```
import numpy as np

data=np.array(data)/255.0
data=np.reshape(data,(data.shape[0],64,64,1))
target=np.array(target)

from keras.utils import np_utils

new_target=np_utils.to_categorical(target)
```

```
np.save('data',data)
np.save('target',new_target)
```

```
import numpy as np

data=np.load('data.npy')
target=np.load('target.npy')

#loading the save numpy arrays in the previous code
```

```
from keras.models import Sequential
from keras.layers import Dense,Activation,Flatten,Dropout
from keras.layers import Conv2D,MaxPooling2D
from keras.callbacks import ModelCheckpoint

model=Sequential()

model.add(Conv2D(16,(3,3),input_shape=data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The first CNN layer followed by Relu and MaxPooling layers

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The second convolution layer followed by Relu and MaxPooling layers

model.add(Flatten())
model.add(Dropout(0.5))
#Flatten layer to stack the output convolutions from second convolution layer
model.add(Dense(128,activation='relu'))
#Dense layer of 128 neurons
model.add(Dense(2,activation='softmax'))
#The Final layer with two outputs for two categories
```

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

```
from sklearn.model_selection import train_test_split

train_data,test_data,train_target,test_target=train_test_split(data,target,test_size=0.1)
```

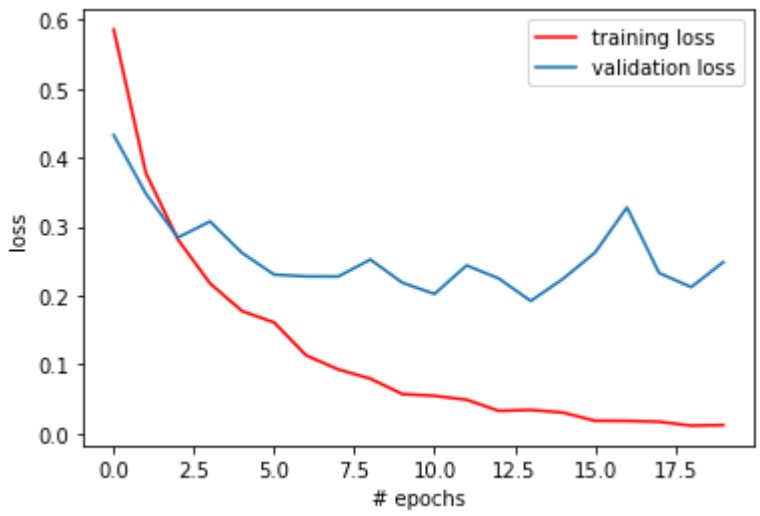
```
print(train_data.shape)
print(test_data.shape)
```

```
(3461, 64, 64, 1)
(385, 64, 64, 1)
```

```
checkpoint = ModelCheckpoint('model-{epoch:03d}.model',monitor='val_loss',verbose=0,save_b
history=model.fit(train_data,train_target,epochs=20,callbacks=[checkpoint],validation_spli
```

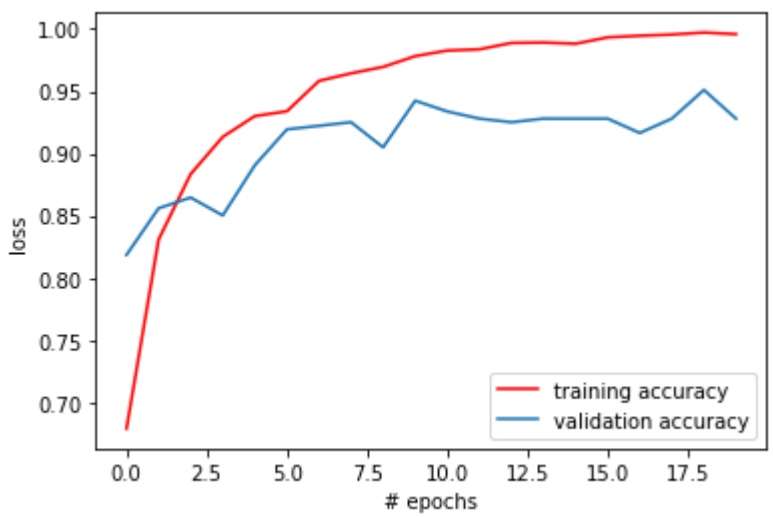
```
Epoch 1/20
97/98 [============================>.] - ETA: 0s - loss: 0.5860 - accuracy: 0.6788WA
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatic
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/tra
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatic
INFO:tensorflow:Assets written to: model-001.model/assets
98/98 [==============================] - 12s 121ms/step - loss: 0.5858 - accuracy: 0
Epoch 2/20
97/98 [============================>.] - ETA: 0s - loss: 0.3784 - accuracy: 0.8305IN
98/98 [==============================] - 12s 123ms/step - loss: 0.3779 - accuracy: 0
Epoch 3/20
97/98 [============================>.] - ETA: 0s - loss: 0.2814 - accuracy: 0.8837IN
98/98 [==============================] - 12s 120ms/step - loss: 0.2812 - accuracy: 0
Epoch 4/20
98/98 [==============================] - 11s 112ms/step - loss: 0.2177 - accuracy: 0
Epoch 5/20
97/98 [============================>.] - ETA: 0s - loss: 0.1772 - accuracy: 0.9301IN
98/98 [==============================] - 12s 120ms/step - loss: 0.1774 - accuracy: 0
Epoch 6/20
```

```
from matplotlib import pyplot as plt

plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```




```
98/98 [==============================] - 11s 111ms/step - loss: 0.0184 - accuracy: 0
```

```
plt.plot(history.history['accuracy'],'r',label='training accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```




```
print(model.evaluate(test_data,test_target))
```

```
13/13 [==============================] - 0s 27ms/step - loss: 0.2254 - accuracy: 0.9
[0.22535236179828644, 0.9402597546577454]
```

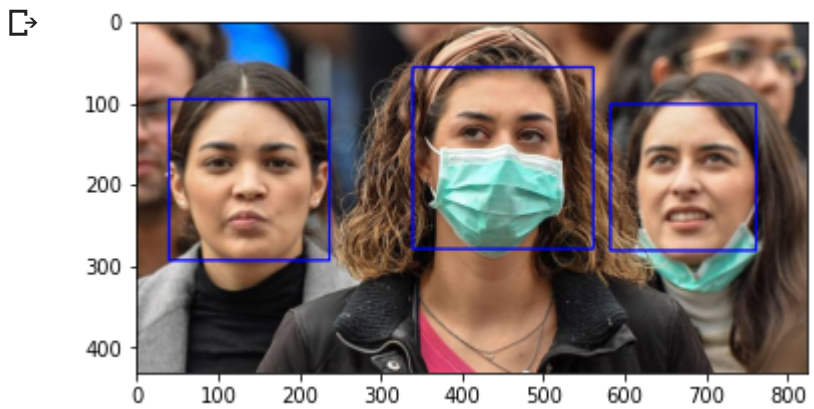
```
from keras.models import load_model
import cv2
import numpy as np
```

```
!pip install git+git://github.com/PnS2019/pnslib.git
```

```
Collecting git+git://github.com/PnS2019/pnslib.git
  Cloning git://github.com/PnS2019/pnslib.git to /tmp/pip-req-build-2thsc29v
  Running command git clone -q git://github.com/PnS2019/pnslib.git /tmp/pip-req-buil
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (fro
Building wheels for collected packages: pnslib
  Building wheel for pnslib (setup.py) ... done
  Created wheel for pnslib: filename=pnslib-0.1.0a1-cp36-none-any.whl size=1489619 s
  Stored in directory: /tmp/pip-ephem-wheel-cache-hh68urj1/wheels/a2/8e/a9/c62e72840
Successfully built pnslib
Installing collected packages: pnslib
Successfully installed pnslib-0.1.0a1
```

```
from pnslib import utils
face_clsfr = cv2.CascadeClassifier(
    utils.get_haarcascade_path('haarcascade_frontalface_default.xml'))

#face_clsfr=cv2.CascadeClassifier('/content/drive/My Drive/Xử lý đa chiều/DIVYA TANWAR/haa

labels_dict={0:'No mask',1:'Mask'}
color_dict={0:(0,0,255),1:(0,255,0)}
```

```
import cv2
from pnslib import utils
import matplotlib.pyplot as plt

# read image
img = cv2.imread("/content/drive/My Drive/Xử lý đa chiều/images/pic2.jpg")

# load face cascade and eye cascade
face_cascade = cv2.CascadeClassifier(
    utils.get_haarcascade_path('haarcascade_frontalface_default.xml'))

eye_cascade = cv2.CascadeClassifier(
    utils.get_haarcascade_path('haarcascade_eye.xml'))

# search face
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    #roi_gray = gray[y:y+h, x:x+w]
    #roi_color = img[y:y+h, x:x+w]
    #eyes = eye_cascade.detectMultiScale(roi_gray)
    #for (ex, ey, ew, eh) in eyes:
        #cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)


img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img)
plt.show()
```




```
# Thử ném 1 ảnh vào
import cv2
from google.colab.patches import cv2_imshow
```

```
img=cv2.imread('/content/drive/My Drive/Xử lý đa chiều/images/pic2.jpg')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
faces=face_clsfr.detectMultiScale(gray,1.3,5)

for x,y,w,h in faces:
    face_img=gray[y:y+w,x:x+w]
    resized=cv2.resize(face_img,(64,64))
    normalized=resized/255.0
    reshaped=np.reshape(normalized,(1,64,64,1))
    result=model.predict(reshaped)
    num=np.argmax(result,axis=1)[0]

    (withoutMask,mask)=result[0]
    label = "{}: {:.2f}%".format(labels_dict[num], max(mask, withoutMask) * 100)

    cv2.putText(img, label, (x, y - 10),cv2.FONT_HERSHEY_SIMPLEX, 0.45, color_dict[num], 1
    cv2.rectangle(img, (x, y), (x+w, y+h), color_dict[num], 1)

img = cv2.resize(img, (320,320))
cv2_imshow(img)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img)
plt.show()
```

```
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
```

```
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))
```

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

img=cv2.imread('photo.jpg')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
faces=face_clsfr.detectMultiScale(gray,1.3,5)

for x,y,w,h in faces:
    face_img=gray[y:y+w,x:x+w]
    resized=cv2.resize(face_img,(64,64))
    normalized=resized/255.0
    reshaped=np.reshape(normalized,(1,64,64,1))
    result=model.predict(reshaped)
    num=np.argmax(result,axis=1)[0]

    (withoutMask,mask)=result[0]
    label = "{}: {:.2f}%".format(labels_dict[num], max(mask, withoutMask) * 100)

    cv2.putText(img, label, (x, y - 10),cv2.FONT_HERSHEY_SIMPLEX, 0.45, color_dict[num], 1
    cv2.rectangle(img, (x, y), (x+w, y+h), color_dict[num], 1)

img = cv2.resize(img, (320,320))
cv2_imshow(img)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img)
plt.show()
```

```
source=cv2.VideoCapture(0)
while(True):
    ret,img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=face_clsfr.detectMultiScale(gray,1.3,5)

    for x,y,w,h in faces:
        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(64,64))
```

```
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,64,64,1))
        result=model.predict(reshaped)
        num=np.argmax(result,axis=1)[0]

        (withoutMask,mask)=result[0]
        label = "{}: {:.2f}%".format(labels_dict[num], max(mask, withoutMask) * 100)

        cv2.putText(img, label, (x, y - 10),cv2.FONT_HERSHEY_SIMPLEX, 0.45, color_dict[num
        cv2.rectangle(img, (x, y), (x+w, y+h), color_dict[num], 1)

    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break
cv2.destroyAllWindows()
source.release()
```