**Cấu trúc dự án**

Các data /thư mục chứa dữ liệu được mô tả trong phần *“Bộ dữ liệu phát hiện mặt nạ COVID-19 ”* .

Ba hình ảnh ví dụ /được cung cấp để bạn có thể kiểm tra trình phát hiện mặt nạ hình ảnh tĩnh.

Chúng tôi sẽ xem xét ba tập lệnh Python trong hướng dẫn này:

* train\_mask\_detector.py :
* Det\_mask\_video.py:

**Triển khai training dự án phát hiện đeo khẩu trang với Keras và TensorFlow**

Bây giờ chúng ta đã xem xét tập dữ liệu mặt nạ của mình, hãy tìm hiểu cách chúng ta có thể sử dụng Keras và TensorFlow để đào tạo bộ phân loại nhằm *tự động* phát hiện xem một người có đeo mặt nạ hay không.

Để hoàn thành nhiệm vụ này, chúng tôi sẽ tinh chỉnh [**kiến trúc MobileNet V2**](https://arxiv.org/abs/1801.04381) , một kiến ​​trúc hiệu quả cao có thể được áp dụng cho các thiết bị nhúng có khả năng tính toán hạn chế.

Việc triển khai hệ thống nhận diện người đeo khẩu trang cho các thiết bị nhúng có thể giảm chi phí sản xuất các hệ thống phát hiện mặt nạ như vậy, do đó lý do tại sao chúng tôi chọn sử dụng kiến ​​trúc này.

Mở ra train\_mask\_detector.py tệp trong cấu trúc thư mục của bạn và chèn mã sau:

# nhập các gói cần thiết

từ tensorflow.keras.preprocessing.image nhập ImageDataGenerator

từ tensorflow.keras.applications nhập khẩu MobileNetV2

từ tensorflow.keras.layers nhập AveragePooling2D

từ tensorflow.keras.layers nhập Dropout

từ tensorflow.keras.layers nhập Flatten

từ tensorflow.keras.layers nhập khẩu rậm

từ tensorflow.keras.layers nhập Đầu vào

từ tensorflow.keras.models nhập Mô hình

từ tensorflow.keras.optimizers nhập Adam

from tensorflow.keras.application.mobilenet\_v2 import preprocess\_input

từ tensorflow.keras.preprocessing.image nhập img\_to\_array

từ tensorflow.keras.preprocessing.image nhập load\_img

từ tensorflow.keras.utils nhập sang\_categorical

từ sklearn.preprocessing import LabelBinarizer

từ sklearn.model\_selection nhập train\_test\_split

từ sklearn.metrics nhập khẩu phân loại\_report

từ các đường dẫn nhập imutils

nhập matplotlib.pyplot dưới dạng plt

nhập numpy dưới dạng np

import argparse

nhập khẩu os

Bộ về tensorflow.keras nhập khẩu cho phép:

* Tăng dữ liệu
* Đang tải trình phân loại MobilNetV2 (chúng tôi sẽ tinh chỉnh mô hình này với các trọng số [**ImageNet**](http://www.image-net.org/) được đào tạo [**trước**](http://www.image-net.org/) )
* Xây dựng một đầu kết nối hoàn toàn (FC) mới
* Sơ chế
* Đang tải dữ liệu hình ảnh

Chúng tôi sẽ sử dụng scikit-learning (sklearn) để phân ngưỡng ảnh các class label, phân đoạn tập dữ liệu và in báo cáo phân loại.

[**Imutils**](https://github.com/jrosebr1/imutils/) của tôi paths việc triển khai sẽ giúp tìm và liệt kê các hình ảnh trong tập dữ liệu. Và chúng tôi sẽ sử dụng matplotlib để vẽ các đường cong đào tạo .

Tensorflow 2.0+ của tôi:

Tôi muốn xác định các siêu tham số học sâu của mình ở một nơi:

# khởi tạo tốc độ học ban đầu, số kỷ nguyên cần đào tạo,

# và kích thước lô

INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

Ở đây, tôi đã chỉ định các hằng số siêu tham số bao gồm tốc độ học ban đầu, số kỷ nguyên đào tạo và kích thước lô. Sau đó, chúng tôi sẽ áp dụng learning rate decay schedule để giảm tốc độ học(learning rate) ban đầu – INIT\_LR

Tải và xử lý trước dữ liệu đào tạo của mình:

# lấy danh sách hình ảnh trong thư mục tập dữ liệu , sau đó khởi tạo

# danh sách dữ liệu (tức là hình ảnh) và hình ảnh lớp

print ( "[INFO] đang tải hình ảnh ..." )

imagePaths = list ( path. list\_images ( args [ "dataset" ]))

dữ liệu = []

nhãn = []

# vòng qua các đường dẫn hình ảnh

cho imagePath trong imagePaths:

# trích xuất nhãn lớp từ tên tệp

nhãn = imagePath. tách ( os.path.sep ) [ -2 ]

# tải hình ảnh đầu vào (224x224) và xử lý trước nó

image = load\_img ( imagePath, target\_size = ( 224 , 224 ))

image = img\_to\_array ( hình ảnh )

image = preprocess\_input ( hình ảnh )

# cập nhật dữ liệu và danh sách nhãn, tương ứng

dữ liệu. nối thêm ( hình ảnh )

nhãn mác. chắp thêm ( nhãn )

# chuyển đổi dữ liệu và nhãn thành mảng NumPy

dữ liệu = np. mảng ( dữ liệu, dtype = "float32" )

nhãn = np. mảng ( nhãn )

Trong khối này, chúng tôi:

* Nắm bắt tất cả  imagePaths trong tập dữ liệu ( **Dòng 44** )
* Khởi tạo dữ liệu và nhãn mác danh sách ( **Dòng 45 và 46** )
* Vòng qua imagePaths và tải + tiền xử lý các hình ảnh ( **Dòng 49-60** ). Các bước tiền xử lý bao gồm thay đổi kích thước thành *224 × 224* pixel, chuyển đổi sang định dạng mảng và điều chỉnh cường độ pixel trong hình ảnh đầu vào thành phạm vi *[-1, 1]* (thông qua prerocess\_input )
* Thêm phần đã tiền xử lý hình ảnh và liên kết nhãn mác đến dữ liệu và nhãn mác danh sách, tương ứng ( **Dòng 59 và 60** )
* Đảm bảo dữ liệu đào tạo ở định dạng mảng NumPy ( **Dòng 63 và 64** )

Các dòng mã trên giả định rằng toàn bộ tập dữ liệu của bạn đủ nhỏ để vừa với bộ nhớ.

Tiếp theo, chúng tôi sẽ mã hóa nhãn mác , phân vùng tập dữ liệu và chuẩn bị cho việc [**tăng dữ liệu**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/) :

# thực hiện mã hóa một lần trên các nhãn

lb = LabelBinarizer ()

nhãn = lb. fit\_transform ( nhãn )

nhãn = to\_categorical ( nhãn )

# phân vùng dữ liệu thành các phần đào tạo và thử nghiệm bằng cách sử dụng 80%

# dữ liệu để đào tạo và 20% còn lại để thử nghiệm

( trainX, testX, trainY, testY ) = train\_test\_split ( dữ liệu, nhãn,

test\_size = 0,20 , phân tầng = nhãn, trạng thái ngẫu nhiên = 42 )

# construct trình tạo hình ảnh đào tạo để tăng dữ liệu

aug = ImageDataGenerator (

Rotation\_range = 20 ,

zoom\_range = 0,15 ,

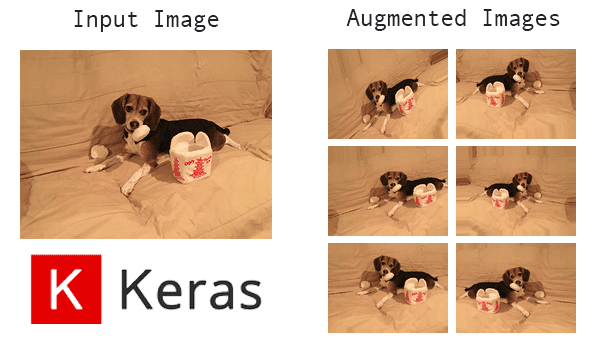
width\_shift\_range = 0,2 ,

height\_shift\_range = 0.2 ,

shear\_range = 0,15 ,

ngang\_flip = Đúng ,

fill\_mode = "gần nhất" )



**Các dòng 67-69** một nóng mã hóa các nhãn lớp , có nghĩa là dữ liệu sẽ ở định dạng sau:

$ python train\_mask\_detector.py - tập dữ liệu tập dữ liệu

[ INFO ] đang tải hình ảnh ...

- > ( trainX, testX, trainY, testY ) = train\_test\_split ( dữ liệu, nhãn,

( Pdb ) nhãn [ 500 : ]

mảng ([[ 1 , 0 ] ,

[ 1 , 0 ] ,

[ 1 , 0 ] ,

...,

[ 0 , 1 ] ,

[ 0 , 1 ] ,

[ 0. , 1. ]] , dtype = float32 )

( Pdb )

Như bạn có thể thấy, mỗi yếu tố của nhãn mác mảng bao gồm một mảng trong đó chỉ có một chỉ mục là "hot" (i.e., 1 ).

Sử dụng phương pháp tiện lợi của scikit-learning, **Dòng 73 và 74** phân đoạn dữ liệu thành 80% đào tạo và 20% còn lại để kiểm tra.

Trong quá trình đào tạo, chúng tôi sẽ áp dụng các đột biến nhanh chóng cho hình ảnh nhằm nỗ lực cải thiện khả năng tổng quát hóa. Điều này được gọi là [**tăng dữ liệu**](https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/) , trong đó các tham số xoay ngẫu nhiên, thu phóng, cắt, dịch chuyển và lật được thiết lập trên các **Dòng 77-84** . Chúng tôi sẽ sử dụngtháng tám đối tượng tại thời điểm đào tạo.

Nhưng trước tiên, chúng ta cần chuẩn bị MobileNetV2 để [**tinh chỉnh**](https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/) :

# tải mạng MobileNetV2, đảm bảo các tập hợp lớp FC đầu là

# rời khỏi

baseModel = MobileNetV2 ( trọng lượng = "imagenet" , include\_top = False ,

input\_tensor = Đầu vào ( shape = ( 224 , 224 , 3 )))

# cấu tạo phần đầu của mô hình sẽ được đặt trên đầu

# mô hình cơ sở

headModel = baseModel.output

headModel = AveragePooling2D ( pool\_size = ( 7 , 7 )) ( headModel )

headModel = Làm phẳng ( name = "làm phẳng" ) ( headModel )

headModel = Dense ( 128 , kích hoạt = "relu" ) ( headModel )

headModel = Bỏ học ( 0.5 ) ( headModel )

headModel = Dense ( 2 , kích hoạt = "softmax" ) ( headModel )

# đặt mô hình FC đầu lên trên mô hình cơ sở (điều này sẽ trở thành

# mô hình thực tế mà chúng tôi sẽ đào tạo)

model = Model ( input = baseModel.input, outputs = headModel )

# lặp lại trên tất cả các lớp trong mô hình cơ sở và đóng băng chúng để chúng sẽ

# \* không \* được cập nhật trong quá trình đào tạo đầu tiên

cho lớp trong baseModel.layers:

layer.trainable = Sai

Thiết lập tinh chỉnh là một quy trình gồm ba bước:

1. Tải MobileNet với các trọng số [**ImageNet**](http://www.image-net.org/) được đào tạo [**trước**](http://www.image-net.org/) , không bỏ đầu mạng ( **Dòng 88 và 89** )
2. Tạo một đầu FC mới và gắn nó vào đế thay cho đầu cũ ( **Dòng 93-102** )
3. Cố định các lớp cơ sở của mạng ( **Dòng 106 và 107** ). Trọng số của các lớp cơ sở này sẽ không được cập nhật trong quá trình nhân giống ngược, trong khi trọng số của lớp đầu *sẽ* được điều chỉnh.

Tinh chỉnh là một chiến lược mà tôi gần như luôn khuyên dùng để thiết lập mô hình cơ sở trong khi tiết kiệm đáng kể thời gian.

Với dữ liệu đã chuẩn bị và kiến ​​trúc mô hình sẵn sàng để tinh chỉnh, chúng tôi hiện đã sẵn sàng biên dịch và đào tạo mạng phát hiện mặt nạ của mình:

# biên dịch mô hình

print ( "[INFO] mô hình biên dịch ..." )

opt = Adam ( lr = INIT\_LR, phân rã = INIT\_LR / EPOCHS )

người mẫu. biên dịch ( loss = "binary\_crossentropy" , Optimizer = opt,

metrics = [ "precision" ])

# đào tạo người đứng đầu mạng lưới

print ( "[INFO] đầu đào tạo ..." )

H = mô hình. phù hợp (

tháng tám. luồng ( trainX, trainY, batch\_size = BS ) ,

step\_per\_epoch = len ( trainX ) // BS,

validation\_data = ( testX, testY ) ,

validation\_steps = len ( testX ) // BS,

epochs = EPOCHS )

**Dòng 111-113** biên dịch mô hình với Adam

trình tối ưu hóa, [**lịch trình giảm tốc độ học tập**](https://www.pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/) và entropy chéo nhị phân. Nếu bạn đang xây dựng từ tập lệnh đào tạo này với *> 2* lớp, hãy đảm bảo sử dụng entropy chéo phân loại.

lr = init\_lr * \frac{1.0}{1.0 + decay * iterations}

Đào tạo về mặt nạ được triển khai qua **dòng 117-122** . Lưu ý cách đối tượng tăng dữ liệu (tháng tám) sẽ cung cấp hàng loạt dữ liệu hình ảnh bị đột biến.

Khi quá trình đào tạo hoàn tất, chúng tôi sẽ đánh giá mô hình kết quả trên bộ thử nghiệm:

# đưa ra dự đoán trên bộ thử nghiệm

print ( "[INFO] đánh giá mạng ..." )

predIdxs = mô hình. dự đoán ( testX, batch\_size = BS )

# đối với mỗi hình ảnh trong tập hợp thử nghiệm, chúng tôi cần tìm chỉ mục của

# nhãn có xác suất dự đoán lớn nhất tương ứng

predIdxs = np. argmax ( predIdxs, axis = 1 )

# hiển thị một báo cáo phân loại được định dạng độc đáo

print ( phân loại\_report ( testY. argmax ( axis = 1 ) , predIdxs,

target\_names = lb.classes\_ ))

# tuần tự hóa mô hình vào đĩa

print ( "[INFO] lưu kiểu máy dò mặt nạ ..." )

người mẫu. save ( args [ "model" ] , save\_format = "h5" )

Tại đây, **Dòng 126-130** đưa ra các dự đoán trên tập kiểm tra, lấy các chỉ số nhãn của lớp có xác suất cao nhất. Sau đó chúng ta in một báo cáo phân loại trong thiết bị đầu cuối để kiểm tra.

**Dòng 138** xếp thứ tự phân loại mặt nạ người mẫu vào đĩa.

Bước cuối cùng là vẽ các đường cong chính xác và mất mát :

# vẽ biểu đồ về độ chính xác và mất mát trong quá trình đào tạo

N = EPOCHS

plt.style. sử dụng ( "ggplot" )

làm ơn. hình ()

làm ơn. plot ( np. arange ( 0 , N ) , H.history [ "loss" ] , label = "train\_loss" )

làm ơn. plot ( np. arange ( 0 , N ) , H.history [ "val\_loss" ] , label = "val\_loss" )

làm ơn. plot ( np. arange ( 0 , N ) , H.history [ " precision " ] , label = "train\_acc" )

làm ơn. plot ( np. arange ( 0 , N ) , H.history [ "val\_accuracy" ] , label = "val\_acc" )

làm ơn. title ( "Mất mát trong đào tạo và độ chính xác" )

làm ơn. xlabel ( "Kỷ nguyên #" )

làm ơn. ylabel ( "Mất mát / Độ chính xác" )

làm ơn. chú giải ( loc = "phía dưới bên trái" )

làm ơn. savefig ( args [ "plot" ])

Khi cốt truyện của chúng ta đã sẵn sàng, **Dòng 152** lưu con số vào đĩa bằng cách sử dụng

--âm mưu

 đường dẫn tập tin.

**Đào tạo máy dò mặt nạ COVID-19 với Keras / TensorFlow**

Hiện chúng tôi đã sẵn sàng đào tạo máy dò mặt nạ của mình bằng Keras, TensorFlow và Deep Learning.

Đảm bảo rằng bạn đã sử dụng phần ***“Tải xuống”*** của hướng dẫn này để tải xuống mã nguồn và bộ dữ liệu mặt nạ.

Từ đó, mở một thiết bị đầu cuối và thực hiện lệnh sau:

**Hình 10: Các** đường cong đào tạo về loss/accurancy của model chứng tỏ độ chính xác cao và ít có dấu hiệu model này bị overfitting trên dữ liệu. chúng ta có thể thấy có rất ít dấu hiệu của overfitting, với validation\_loss luôn thấp hơn train\_loss,

Giờ đây, chúng tôi đã sẵn sàng áp dụng kiến ​​thức của mình về thị giác máy tính và học sâu bằng Python, OpenCV và TensorFlow / Keras để thực hiện bài toán.

Như bạn có thể thấy, chúng tôi đang đạt được **độ chính xác ~ 99%** trên bộ thử nghiệm của mình.

**Triển khai máy dò mặt nạ COVID-19 trong các luồng video thời gian thực với OpenCV**

# nhập các gói cần thiết

from tensorflow.keras.application.mobilenet\_v2 import preprocess\_input

từ tensorflow.keras.preprocessing.image nhập img\_to\_array

từ tensorflow.keras.models nhập load\_model

từ imutils.video nhập khẩu dòng video

nhập numpy dưới dạng np

import argparse

nhập imutils

thời gian nhập khẩu

nhập cv2

nhập khẩu os

Thuật toán cho tập lệnh này giống nhau, nhưng nó được ghép lại với nhau theo cách để cho phép xử lý mọi khung hình trong luồng webcam của bạn.

Do đó, sự khác biệt duy nhất khi nói đến nhập khẩu là chúng ta cần một Dòng video lớp học và thời gian. Cả hai điều này sẽ giúp chúng tôi làm việc với luồng. Chúng tôi cũng sẽ tận dụng

imutils cho phương pháp thay đổi kích thước nhận biết khía cạnh của nó.

Logic dự đoán mặt nạ / phát hiện khuôn mặt cho tập lệnh này nằm trong

phát hiện\_and\_p dự đoán\_ mặt nạ

 hàm số:

def detector\_and\_posystem\_mask ( frame, faceNet, maskNet ) :

# lấy các kích thước của khung và sau đó tạo một đốm màu

# từ nó

( h, w ) = frame.shape [ : 2 ]

blob = cv2.dnn. blobFromImage ( khung, 1.0 , ( 300 , 300 ) ,

( 104,0 , 177,0 , 123,0 ))

# vượt qua đốm màu qua mạng và nhận được tính năng nhận diện khuôn mặt

faceNet. setInput ( blob )

dò tìm = faceNet. chuyển tiếp ()

# khởi tạo danh sách các khuôn mặt, vị trí tương ứng của chúng,

# và danh sách các dự đoán từ mạng khẩu trang

khuôn mặt = []

locs = []

preds = []

Bằng cách xác định chức năng tiện lợi này ở đây, vòng xử lý khung của chúng ta sẽ dễ đọc hơn một chút về sau.

**Chức năng này phát hiện khuôn mặt và sau đó áp dụng công cụ phân loại mặt nạ cho ROI của từng khuôn mặt.**Một hàm như vậy hợp nhất mã - nó thậm chí có thể được chuyển sang một tệp Python riêng biệt nếu bạn muốn.

phát hiện\_and\_p dự đoán\_ mặt nạ

 hàm chấp nhận ba tham số:

* frame

: Một khung từ luồng

* faceNet

: Mô hình được sử dụng để phát hiện vị trí của các khuôn mặt trong ảnh

* maskNet

: Mô hình phân loại khẩu trang COVID-19

Bên trong, chúng tôi xây dựng một

bãi

, phát hiện khuôn mặt và khởi tạo danh sách, hai trong số đó hàm được đặt để trả về. Những danh sách này bao gồm

những khuôn mặt

 (tức là, ROI),

locs

 (các vị trí khuôn mặt), và

preds

 (danh sách dự đoán có mặt nạ / không có mặt nạ).

Từ đây, chúng ta sẽ lặp lại khuôn mặt

sự phát hiện

:

# vòng lặp qua các phát hiện

đối với tôi trong phạm vi ( 0 , detection.shape [ 2 ]) :

# trích xuất độ tin cậy (tức là xác suất) được liên kết với

# phát hiện

độ tin cậy = sự phát hiện [ 0 , 0 , i, 2 ]

# lọc ra các phát hiện yếu bằng cách đảm bảo độ tin cậy là

# lớn hơn độ tin cậy tối thiểu

nếu niềm tin > args [ "niềm tin" ] :

# tính tọa độ (x, y) của hộp giới hạn cho

# đối tượng

hộp = phát hiện [ 0 , 0 , i, 3 : 7 ] \* np. mảng ([ w, h, w, h ])

( startX, startY, endX, endY ) = hộp. astype ( "int" )

# đảm bảo các hộp giới hạn nằm trong kích thước của

# khung

( startX, startY ) = ( max ( 0 , startX ) , max ( 0 , startY ))

( endX, endY ) = ( min ( w - 1 , endX ) , min ( h - 1 , endY ))

Bên trong vòng lặp, chúng tôi lọc ra các phát hiện yếu ( **Dòng 34-38** ) và trích xuất các hộp giới hạn trong khi đảm bảo tọa độ hộp giới hạn không nằm ngoài giới hạn của hình ảnh ( **Dòng 41-47** ).

Tiếp theo, chúng tôi sẽ thêm ROI theo khuôn mặt vào hai trong số các danh sách tương ứng :

# trích xuất ROI của khuôn mặt, chuyển đổi nó từ BGR sang kênh RGB

# đặt hàng, thay đổi kích thước thành 224x224 và xử lý trước

face = frame [ startY: endY, startX: endX ]

face = cv2. cvtColor ( khuôn mặt, cv2.COLOR\_BGR2RGB )

face = cv2. thay đổi kích thước ( khuôn mặt, ( 224 , 224 ))

face = img\_to\_array ( face )

face = preprocess\_input ( face )

# thêm các hộp khuôn mặt và viền vào tương ứng của chúng

# danh sách

những khuôn mặt. chắp thêm ( mặt )

locs. append (( startX, startY, endX, endY ))

Sau khi trích xuất ROI của khuôn mặt và xử lý trước ( **Dòng 51-56** ), chúng tôi nối

đối mặt

 ROI và các hộp giới hạn trong danh sách tương ứng của chúng.

Bây giờ chúng tôi đã sẵn sàng để chạy

những khuôn mặt

 thông qua công cụ dự đoán mặt nạ :

# chỉ đưa ra dự đoán nếu ít nhất một khuôn mặt được phát hiện

nếu len ( mặt ) > 0 :

# để suy luận nhanh hơn, chúng tôi sẽ đưa ra dự đoán hàng loạt trên \* tất cả \*

# khuôn mặt cùng một lúc thay vì dự đoán từng khuôn mặt

# trong vòng lặp `for` ở trên

mặt = np. mảng ( mặt, dtype = "float32" )

preds = maskNet. dự đoán ( khuôn mặt, batch\_size = 32 )

# trả về 2 bộ vị trí khuôn mặt và vị trí tương ứng của chúng

# địa điểm

return ( locs, preds )

Logic ở đây được xây dựng cho tốc độ. Đầu tiên, chúng tôi đảm bảo ít nhất một khuôn mặt được phát hiện ( **Dòng 63** ) - nếu không, chúng tôi sẽ

trở lại

 trống

preds

.

***2020-06-10 Cập nhật:****Bài đăng trên blog này hiện đã được cập nhật với****Dòng 67****để chuyển đổi*

*những khuôn mặt*

*thành mảng NumPy dấu phẩy động 32 bit. Ngoài ra,****Dòng 61****từ khối trước đã bị xóa (trước đây, nó đã thêm một thứ nguyên lô không cần thiết). Sự kết hợp của hai thay đổi này hiện****sửa một lỗi****đang ngăn chặn nhiều*

*preds*

*được trả về từ suy luận. Với bản sửa lỗi, nhiều khuôn mặt trong một hình ảnh được nhận dạng đúng là có mặt nạ hoặc không có mặt nạ.*

Thứ hai, chúng tôi đang thực hiện kết luận trên  *toàn bộ lô* của

những khuôn mặt

trong khung để đường dẫn nhanh hơn ( **Dòng 68** ). Sẽ không hợp lý nếu bạn viết một vòng lặp khác để đưa ra dự đoán trên từng mặt riêng lẻ do chi phí cao (đặc biệt nếu bạn đang sử dụng GPU yêu cầu nhiều giao tiếp trên xe buýt hệ thống của bạn). **Sẽ hiệu quả hơn nếu thực hiện các dự đoán hàng loạt.**

**Dòng 72** trả về vị trí hộp giới hạn khuôn mặt và các dự đoán mặt nạ / không mặt nạ tương ứng cho người gọi.

Tiếp theo, chúng tôi sẽ xác định các [**đối số dòng lệnh**](https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/)  :

# xây dựng trình phân tích cú pháp đối số và phân tích cú pháp đối số

ap = argparse. ArgumentParser ()

ap. add\_argument ( "-f" , "--face" , type = str,

default = "face\_detector" ,

help = "đường dẫn đến thư mục mô hình máy dò khuôn mặt" )

ap. add\_argument ( "-m" , "--model" , type = str,

default = "mask\_detector.model" ,

help = "đường dẫn đến mô hình phát hiện mặt nạ được đào tạo" )

ap. add\_argument ( "-c" , "--confidence" , type = float, default = 0.5 ,

help = "xác suất tối thiểu để lọc các phát hiện yếu" )

args = vars ( ap. parse\_args ())

Các đối số dòng lệnh bao gồm:

* --đối mặt

: Đường dẫn đến thư mục dò tìm khuôn mặt

* --model

: Đường dẫn đến trình phân loại mặt nạ được đào tạo

* --sự tự tin

: Ngưỡng xác suất tối thiểu để lọc các phát hiện khuôn mặt yếu

Với việc nhập khẩu, chức năng tiện lợi và dòng lệnh

args

 sẵn sàng hoạt động, chúng tôi chỉ có một số khởi tạo cần xử lý trước khi chúng tôi lặp qua các khung:

# tải mô hình máy dò khuôn mặt được tuần tự hóa từ đĩa

print ( "[INFO] đang tải kiểu máy dò khuôn mặt ..." )

prototxtPath = os.path.sep. tham gia ([ args [ "face" ] , "deploy.prototxt" ])

weightsPath = os.path.sep. tham gia ([ args [ "face" ] ,

"res10\_300x300\_ssd\_iter\_140000.caffemodel" ])

faceNet = cv2.dnn. readNet ( prototxtPath, weightsPath )

# tải mô hình phát hiện mặt nạ từ đĩa

print ( "[INFO] tải kiểu máy dò mặt nạ ..." )

maskNet = load\_model ( args [ "model" ])

# khởi chạy luồng video và cho phép cảm biến máy ảnh nóng lên

print ( "[INFO] bắt đầu luồng video ..." )

vs = VideoStream ( src = 0 ) . start ()

thời gian. ngủ ( 2.0 )

Ở đây chúng tôi đã khởi tạo:

* Máy dò khuôn mặt
* Máy dò mặt nạ COVID-19
* Luồng video webcam

Hãy tiếp tục lặp lại các khung trong luồng:

# vòng lặp qua các khung từ luồng video

trong khi Đúng :

# lấy khung hình từ luồng video phân luồng và thay đổi kích thước

# để có chiều rộng tối đa là 400 pixel

frame = so với read ()

khung = imutils. thay đổi kích thước ( khung, chiều rộng = 400 )

# phát hiện các khuôn mặt trong khung và xác định xem họ có đang đeo

# mặt nạ hay không

( locs, preds ) = Discovery\_and\_posystem\_mask ( frame, faceNet, maskNet )

Chúng tôi bắt đầu lặp lại các khung trên **Dòng 103** . Bên trong, chúng tôi lấy một

khung

 từ dòng suối và

thay đổi kích thước

nó ( **Dòng 106 và 107** ).

Từ đó, chúng tôi đưa tiện ích thuận tiện của mình vào sử dụng; **Đường dây 111** phát hiện và dự đoán xem mọi người có đeo mặt nạ hay không.

Hãy xử lý hậu kỳ (tức là chú thích) kết quả phát hiện mặt nạ COVID-19:

# vòng qua các vị trí khuôn mặt được phát hiện và vị trí tương ứng của chúng

# địa điểm

for ( box, pred ) trong zip ( locs, preds ) :

# giải nén hộp giới hạn và dự đoán

( startX, startY, endX, endY ) = hộp

( mask, withoutMask ) = pred

# xác định nhãn lớp và màu mà chúng ta sẽ sử dụng để vẽ

# hộp giới hạn và văn bản

label = "Mask" if mask > withoutMask else "No Mask"

color = ( 0 , 255 , 0 ) if label == "Mask" else ( 0 , 0 , 255 )

# bao gồm xác suất trong nhãn

label = "{}: {: .2f}%" . định dạng ( nhãn, tối đa ( mặt nạ, không có nhiệm vụ ) \* 100 )

# hiển thị nhãn và hình chữ nhật hộp giới hạn trên đầu ra

# khung

cv2. putText ( khung, nhãn, ( startX, startY - 10 ) ,

cv2.FONT\_HERSHEY\_SIMPLEX, 0,45 , màu, 2 )

cv2. hình chữ nhật ( khung, ( startX, startY ) , ( endX, endY ) , color, 2 )

Bên trong vòng lặp về kết quả dự đoán (bắt đầu từ **Dòng 115** ), chúng tôi:

* Mở gói hộp giới hạn khuôn mặt và dự đoán mặt nạ / không mặt nạ ( **Dòng 117 và 118** )
* Xác định

nhãn mác

 và

màu sắc

( **Dòng 122-126** )

* Chú thích

nhãn mác

và hộp giới hạn mặt ( **Dòng 130-132** )

Cuối cùng, chúng tôi hiển thị kết quả và thực hiện dọn dẹp:

# hiển thị khung đầu ra

cv2. imshow ( "Khung" , khung )

phím = cv2. waitKey ( 1 ) & 0xFF

# nếu phím `q` được nhấn, ngắt khỏi vòng lặp

if key == ord ( "q" ) :

nghỉ

# dọn dẹp một chút

cv2. killAllWindows ()

so với dừng ()

Sau

khung

 được hiển thị, chúng tôi nắm bắt

Chìa khóa

máy ép. Nếu người dùng nhấn

NS

 (bỏ), chúng tôi

nghỉ

 ra khỏi vòng lặp và thực hiện công việc dọn phòng.

Thật tuyệt khi triển khai máy dò mặt nạ thời gian thực của bạn với Python, OpenCV và học sâu với TensorFlow / Keras!

**Phát hiện mặt nạ COVID-19 bằng OpenCV trong thời gian thực**

Để xem máy dò mặt nạ COVID-19 thời gian thực đang hoạt động, hãy đảm bảo bạn sử dụng phần ***“Tải xuống”*** của hướng dẫn này để tải xuống mã nguồn và mô hình máy dò mặt nạ được đào tạo trước.

Sau đó, bạn có thể khởi chạy trình phát hiện mặt nạ trong các luồng video thời gian thực bằng lệnh sau:

$ python explore\_mask\_video.py

[INFO] tải mô hình máy dò khuôn mặt ...

[INFO] đang tải mô hình máy dò mặt nạ ...

[INFO] bắt đầu luồng video ...0:07

Ở đây, bạn có thể thấy rằng máy dò mặt nạ có khả năng chạy trong thời gian thực (và cũng *chính xác* trong dự đoán của nó).

**Góp ý để phát triển**

Như bạn có thể thấy từ các phần kết quả ở trên, máy dò mặt nạ đang hoạt động khá tốt mặc dù:

1. Có dữ liệu đào tạo hạn chế
2. Các

with\_mask

lớp được tạo nhân tạo (xem phần *“Tập dữ liệu mặt nạ được tạo như thế nào?”* ở trên).

Để cải thiện hơn nữa mô hình phát hiện mặt nạ , **bạn nên thu thập *hình ảnh thực tế* (thay vì hình ảnh được tạo nhân tạo) về những người đeo mặt nạ.**

Mặc dù tập dữ liệu nhân tạo hoạt động tốt trong trường hợp này, nhưng không có gì thay thế được cho dữ liệu thực.

**Thứ hai, bạn cũng nên thu thập hình ảnh của những khuôn mặt có thể “nhầm lẫn” bộ phân loại nghĩ rằng người đó đang đeo mặt nạ trong khi thực tế không phải vậy** - các ví dụ tiềm năng bao gồm áo sơ mi quấn quanh mặt, quấn khăn trên miệng, v.v.

Tất cả những điều này là ví dụ về một thứ *có thể* bị nhầm lẫn là mặt nạ bằng máy dò mặt nạ .

**Cuối cùng, bạn nên xem xét đào tạo một bộ *phát hiện đối tượng* hai lớp chuyên dụng hơn là một bộ phân loại hình ảnh đơn giản.**

Phương pháp hiện tại để phát hiện một người có đeo mặt nạ hay không là một quy trình gồm hai bước:

1. **Bước # 1:** Thực hiện nhận diện khuôn mặt
2. **Bước # 2:** Áp dụng máy dò mặt nạ cho từng khuôn mặt

Các *vấn đề* với phương pháp này là một mặt nạ, theo định nghĩa, che khuất một phần của khuôn mặt. Nếu đủ khuôn mặt bị che khuất, khuôn mặt sẽ không thể được phát hiện *và do đó, thiết bị dò tìm mặt nạ sẽ không được áp dụng.*

Để giải quyết vấn đề đó, bạn nên đào tạo một bộ phát hiện đối tượng hai lớp bao gồm

with\_mask

 lớp học và

without\_mask

 lớp.

Kết hợp máy dò vật thể với máy chuyên dụng

with\_mask

 lớp sẽ cho phép cải tiến mô hình theo hai khía cạnh.

Đầu tiên, máy dò đối tượng sẽ có thể phát hiện tự nhiên những người đeo mặt nạ mà nếu không máy dò khuôn mặt sẽ không thể phát hiện được do khuôn mặt bị che khuất quá nhiều.

Thứ hai, cách tiếp cận này giảm phạm vi thị giác máy tính xuống một bước duy nhất - thay vì áp dụng nhận diện khuôn mặt và *sau đó là* mô hình phát hiện mặt nạ , tất cả những gì chúng tôi cần làm là áp dụng công cụ phát hiện đối tượng để cung cấp cho chúng tôi các hộp giới hạn cho mọi người

with\_mask

 và

without\_mask

 trong một lần chuyển tiếp của mạng.