ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN HỌC: CS114

ĐỀ TÀI: PHÁT HIỆN ĐỐI TƯỢNG KHÔNG ĐEO KHẨU TRANG VÀ ĐEO KHẨU TRANG SAI CÁCH THÔNG QUA CAMERA GIÁM SÁT

Giảng viên hướng dẫn:
ThS. Phạm Ngyễn Trường An
PGS . TS Lê Đình Duy

Sinh viên thực hiện:
Vũ Nguyễn Nhật Thanh - 19522246
Trương Thế Tấn - 19522180
Nông Thanh Hồng - 19521551

TP. HỒ CHÍ MINH - 07/2021

Nội dung

CHƯƠNG 1. TỔNG QUAN

1.1. Mô tả bài toán

1.1.1. Ngữ cảnh ứng dụng

Dịch covid – 19 đang hoành hành trên khắp thế giới trong đó có Việt Nam. Với tốc độ lây lan chóng mặt như hiện nay, người dân cần tăng cường ý thức phòng chống đối với loại dịch bệnh này. Việc đeo khẩu trang đúng cách chính là biện pháp khả thi và dễ dàng cho việc phòng chống dịch toàn dân này. Khẩu trang chính là vật dụng phổ biến đối với người dân, có giá thành rẻ, dễ tiếp cận và có thể tìm thấy ở bất kỳ hiệu thuốc nào. Tuy nhiên, vẫn có rất nhiều người không đeo khẩu trang hoặc đeo không đúng cách ở những nơi đông người và quen thuộc như phía trước các cửa hàng hoặc các tòa nhà… Gây ảnh hưởng đến người khác đồng thời tăng gánh nặng nhân lực khi phải tăng cường sự giám sát đối với những người này. Làm giảm hiệu quả của việc phòng chống dịch. Vậy nên việc áp dụng công nghệ thông tin để phát hiện sớm các đối tượng này thông qua các camera được gắn tại các cửa ra vào sẽ giúp giảm bớt gánh nặng về nhân sự và nguy cơ lây nhiễm chéo.

1.1.2. Input và Output của bài toán

Input của bài toán là các frame mà các camera outdoor hoặc các camera hồng ngoại ghi lại được trong lúc ghi hình tại các cửa ra vào hoặc bên trong các toà nhà. Output của bài toán là một con số đại diện cho nhãn được mô hình dự đoán ra kèm theo một vector chứa tọa độ của các bbox bao quanh khuôn mặt tương ứng.

1.2. Mô tả dữ liệu

Dữ liệu được sử dụng cho bài toán được thu thập trên trang web greetyimage.com, bằng các từ khóa "face mask, daily in life in a country, crown, covid-19 and people, …".

Các khó khăn khi thu thập dữ liệu:

- Dữ liệu được tìm kiếm thủ công và phải đươc resize nhỏ lại trước khi huấn luyện để tránh tràn VRAM.
- Phải chọn lọc trên từng bức ảnh sao cho khớp với mô tả của bài toán và ngữ cảnh thực tế có thể xảy ra trong thực tế.
- Tốn nhiều thời gian và công sức để thu thập.

Dữ liệu nhóm em sử dụng là do nhóm em tự thu thập, và chưa có ai thu thập dữ liệu này, nhóm em xin đưa ra những dẫn chứng sau:

- Các dataset mà nhóm tìm được trước đây đều có một điểm chung là giải quyết bài toán phát hiện đeo khẩu trang nhưng tất cả chúng đều không khớp hoàn toàn với ngữ cảnh mà nhóm đang giải quyết.
- Các dataset mà nhóm tìm được đều chỉ dùng cho bài toán phân loại và chúng chỉ có 2 nhãn là đeo khẩu trang và không đeo khẩu trang.
- Các dataset có sẵn đều sử dụng cách ghép khẩu trang vào những khuôn mặt không đeo khẩu trang để tạo ra nhãn đeo khẩu trang sai, dataset có sẵn mà nhóm từng tìm được đều không có nhãn đeo khẩu trang sai được lấy từ thực tế.
- Các dataset có sẵn đều đã được huấn luyện với mô hình của người công bố dataset, các mô hình đó đều có kết quả tốt với độ chính xác cao, nhóm không muốn tốn thời gian để giải quyết một bài toán đã được giải rất tốt trước đó.

CHƯƠNG 2. CÁC NGHIÊN CỨU TRƯỚC ĐÓ

2.1. Convolutional Neural Network(CNN) trong xử lý ảnh

CNN là một mô hình học sâu phổ biến hiện nay, hầu hết các hệ thống nhận diện và xử lý ảnh hiện nay đều sử dụng CNN vì tốc độ xử lý nhanh và có độ chính xác cao.

Cấu trúc một CNN bao gồm:

- **Convolutional layer**
- **Pooling layer**
- **Fully - connected layer**

Convolutional layer thực hiện phép toán cross - correlation hoặc convolution, **pooling layer** làm giảm kích thước mẫu bằng các khối 2x2 của tầng trước đó, **fully - connected layer** khá giống với các neural network thông thường khác, nó thực hiện công việc cuối cùng là phân lớp dữ liệu.

Ưu điểm trong xử lý ảnh

- Thách thức trong xử lý ảnh là khi làm việc với ảnh màu chính là lượng dữ liệu khổng lồ cần phải tính toán. Ví dụ điển hình là các bức ảnh sử dụng trong Computer Vision, chúng thường có kích thước là 224x224 hoặc lớn hơn, màu sắc của mỗi một pixel ảnh là sự trộn lẫn 3 loại màu là Red, Green, Blue nếu các bức hình đó sử dụng kênh màu RGB, việc này dẫn đến một điều là số input features sẽ rất lớn, cụ thể trong ví dụ này là:

$$(224\times224)\times3=150528\ (input\ features)$$

  Trong đó: (224×224) là số pixel của bức ảnh; 3 là số bộ lọc, ứng với 3 màu Red, Green, Blue.

- Nếu sử dụng kiến trúc neural network cũ thì số lượng neuron lớn như vậy sẽ làm cho toàn bộ quá trình học rất chậm và dẫn đến quá tải cho hệ thống.

- Thêm nữa với các kiến trúc xử lý ảnh cũ thì : nếu huấn luyện một network mà hoạt động tốt trên một hình ảnh vật thể nhất định nhưng khi cho network thử với cùng một bức hình đó nhưng vị trí của vật thể bị lệch sang một vị trí khác, khi đó network sẽ phản hồi hoàn toàn khác.

Nhược điểm trong phát hiện đối tượng trong ảnh.

- Tuy hoạt động rất tốt trong việc phân loại các vật thể thông qua các đặc trưng. Nhưng với nhiệm vụ phát hiện vật thể, số lượng vật thể trong mỗi bức ảnh là không giống nhau, hai vật thể giống nhau có thể có trong cùng một bức ảnh. Vì vậy số lượng đầu ra của một mô hình CNN đơn thuần sẽ thay đổi chứ không giữ nguyên.

2.2. Phát hiện đối tượng - Object Detection

Phát hiện đối tượng là một trong những vấn đề phổ biến nhất trong lĩnh vực thị giác máy tính. Yêu cầu chính của vấn đề này chính là tìm ra đối tượng trong ảnh và phân loại nó giữa rất nhiều đối tượng khác trong ảnh. Với sự phát triển của khoa học máy tính, rất nhiều kỹ thuật hiện đại với tốc độ và độ chính xác cao đã được ra đời.

Các phương pháp phát hiện đối tượng có thể được chia ra thành hai loại:

- **Phát hiện hai giai đoạn** - Bao gồm 1 giai đoạn để tạo vùng đề xuất đối tượng và 1 giai đoạn phân loại đối tượng. Tiêu biểu hiện giờ là **Faster – RCNN.**
- **Phát hiện một giai đoạn hay đề xuất vùng tự do** – quy trình được thực hiện từ đầu đến cuối không tách rời nhau. Tiêu biểu hiện giờ là **Single Shot MultiBox Detector - SSD.**
-

2.3. Một vài nghiên cứu trước

1. Link trích dẫn: Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment.

- Dữ liệu được sử dụng trong bài báo được các tác giả tạo thủ công một phần và có thêm dữ liệu được thu thập trên mạng với tổng cộng gần 7500 bức hình. Dữ liệu gồm 2 nhãn là **Mask** và **No mask** . Ví dụ:

Hình 1: Những người đeo khẩu trang

- Các tác giả sử dụng 2 phương pháp trong bài báo đó là **Yolo v3** và **Faster CNN.** Với mỗi phương pháp, hiệu xuất lần lượt là:

+ Yolo v3:

Training loss - 0.1

Validation loss- 0.25

+ Faster-RCNN:

Training loss - 0.04

Validation loss- 0.15

2. Link trích dẫn: Face Mask Detection using SSD

- Dữ liệu được tác giả sử dụng là dữ liệu gồm 5749 bức ảnh chứa nhiều mặt người thuộc hai nhãn là **Mask** và **No mask**.
- Hiệu xuất của mô hình **SSD**:

+ Training loss - 0.1723

+ Validation loss- 0.1322

CHƯƠNG 3. XÂY DỰNG BỘ DỮ LIỆU

3.1. Tiêu chí để chọn lọc dữ liệu.

Những bức ảnh được nhóm thu thập phải là ảnh chụp ngoài trời hoặc trong các và phải thỏa mãn những yếu tố về ánh sáng, backgound, góc chụp, và điều kiện thời tiết đối với ảnh chụp ngoài trời:

- Ánh sáng phải là các ánh sáng trắng từ các bóng đèn hoặc là ánh sáng mặt trời vào ban ngày trong khoảng thời gian từ 6h sáng đến 17h30 chiều, hoặc ánh sáng trắng từ các bóng đèn trên trần của các tòa nhà, tòa chung, ánh sáng từ các cột đèn điện hoặc từ các bóng đèn được treo để thắp sáng phần sân phía trước cửa ra vào tại các toà nhà.

Hình 2: Hình minh họa yếu tố ánh sáng

- Các đối tượng chính cần phát hiện trong ảnh (khuôn mặt) phải rõ nét hoặc hơi mờ nhưng vẫn có thể nhìn ra được, các khuôn mặt đeo các loại khẩu trang phổ biến như khẩu trang y tế, khẩu trang vải, khẩu trang kháng bụi chất lượng cao, không tính những trường hợp đeo mặt nạ chống khí độc, các khuôn mặt không bị che khuất hoặc bị che khuất một phần nhỏ để vẫn nhìn thấy khẩu trang và phần lớn khuôn mặt, những khuôn mặt sử dụng khăn

choàng thay cho khẩu trang vẫn được chấp nhận, các khuôn mặt có thể quay trái, phải, cúi xuống hoặc trực diện (nhìn thẳng vào camera) và không có ngẩng đầu lên quá cao vì không có nhiều trường hợp những người ra vào các tòa nhà lại ngước nhìn lên quá cao.

Hình 3: HÌnh minh họa đối tượng trong ảnh

- Điều kiện thời tiết đối với ảnh chụp ngoài trời là những điều kiện thời tiết có thể gặp trong thực tế như mưa rào, mưa phùn, nắng gắt, nhiều bóng râm, hoặc có sương mù nhẹ (sương sớm), không lấy những bức hình có sương mù dày đặc do hiện tượng này hiếm gặp tại các khu vực thị trấn, thành phố và các vùng đồng bằng.

Hình 4: Hình minh họa thời tiết

- Background phía sau các đối tượng cần detect (khuôn mặt) là những hình ảnh về các đối tượng mà tại các cửa ra vào của các cửa hàng và các tòa nhà, tòa chung cư thường gặp như xe hơi, đường phố, cây xanh, thảm cỏ, làn đường, hàng rào, tòa nhà, bức tường, thú nuôi, con người, cột điện, các biển báo giao thông, và các tờ quảng cáo trên các bức tường hoặc trên cột điện.

Hình 5: Hình minh họa Background

3.2. Các yếu tố khác

Để kiểm tra được tính đúng đắn của dữ liệu, nhóm tiến hành chọn lọc ngay từ lúc tìm kiếm ảnh, sau khi kiếm ảnh xong thì nhóm tiến hành quan sát lai và đánh giá để loại bỏ những bức ảnh không thỏa với tiêu chí nhóm đặt ra.

Các trường hợp khó cho mô hình:

- Kích thước khuôn mặt nhỏ (32x32) làm mất đi các chi tiết nhỏ
- Khuôn mặt bị mờ khiến các chi tiết không được biểu hiện rõ ràng
- Khuôn mặt bị che khuất với các đối tượng khác

Hình 6: Hình ảnh cho trường hợp khó

- Khuôn mặt cúi quá sâu
- Phần cổ được che kín bởi áo khoác hoặc khăn choàng dẫn đến mô hình có thể khó phân biệt là không đeo khẩu trang hay đeo khẩu trang sai cách.

Hình 7: Hình ảnh cho trường hợp khó

3.3. Độ đa dạng

Số lượng ảnh tập train và tập test lần lượt là 1572 ảnh và 362 ảnh.

Số lượng labels trong các tập train và test được thể hiện bảng dưới đây:

labels	**Training set** *(labels)*	**Test set** *(labels)*
Correct	4864	1127
Incorrect	1146	215
Face	1510	632

Số lượng ảnh của mỗi label trong tập train và test được thể hiện bảng dưới đây:

labels	**Training set** *(ảnh)*	**Test set** *(ảnh)*
Correct	1278	264
Incorrect	677	141
Face	632	148

CHƯƠNG 4. TRAINING VÀ ĐÁNH GIÁ MODEL

4.1. Mô hình SSD_Resnet50

4.1.1. Nguồn tham khảo

Nguồn tham khảo:

- Training Custom Object Detector
- TensorFlow 2 Detection Model Zoo

4.1.2. Các khó khăn trong quá trình huấn luyện

Do do Resnet50 có số lượng conv_layer nhiều nền việc huấn luyện mô hình bằng CPU mất rất nhiều thời gian, CPU hoạt động hết công xuất trong thời gian dài có thể gây ra nhiều hư hại cho các linh kiện xung quanh và cả bo mạch chủ nằm bên dưới CPU, đối với CPU dòng U thì điều này lại sẽ là thảm họa do nó không được thiết kế để chạy tác vụ nặng trong thời gian dài. Vì vậy để huấn luyện mô hình ssd sử dụng resnet50, cần sử dụng GPU để tăng tốc độ xử lý và giảm thời gian huấn luyện, sử dụng colab để không tốn chi phí cho phần cứng.

Do huấn luyện mô hình bằng GPU nên cần đảm bảo môi trường huấn luyện có cuda và phiên bản của các gói được sử dụng như numpy, tensorflow, libgcc, python,…

4.1.3. Thiết lập môi trường

Vì object detection API và COCO API đều sử dụng tensorflow 2.5.0, tensorflow 2.5.0 được build trên nền cuda 11, cuda 11 trên colab đang bị lỗi thiếu thư viện liên kết khiến cho tensorflow không thể sử dụng GPU được. Vì vậy cần sử dụng tensorflow 2.3.0 để sử dụng cuda 10, một môi trường không thể cùng cài song song 2 phiên bản tensorflow nên cần tạo môt môi trường ảo để cài đặt vả sử dụng được cả 2 phiên bản trên.

4.1.3.1. Mini Conda

Nguồn tham khảo: Conda + Google Colab

Cài đặt mini conda và nâng cấp version của toàn bộ gói trong môi trường lên phiên bản mới nhất đồng thời cài thêm một số gói cần thiết khác.

1.1 Tải và cài đặt conda

```
[ ] %%bash
    MINICONDA_INSTALLER_SCRIPT=Miniconda3-4.5.4-Linux-x86_64.sh
    MINICONDA_PREFIX=/usr/local
    wget https://repo.continuum.io/miniconda/$MINICONDA_INSTALLER_SCRIPT
    chmod +x $MINICONDA_INSTALLER_SCRIPT
    ./$MINICONDA_INSTALLER_SCRIPT -b -f -p $MINICONDA_PREFIX
```

1.2. cập nhật các gói và cài đặt tensorflow

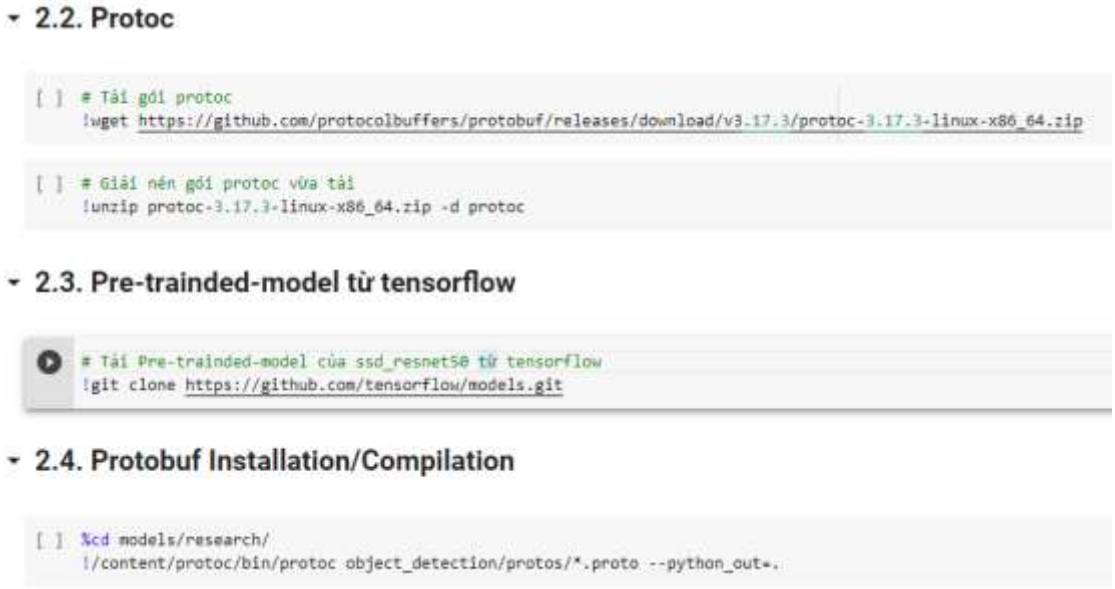
```
!rm Miniconda3-4.5.4-Linux-x86_64.sh
!conda install --channel defaults conda python=3.9.5 --yes
!conda update --channel defaults --all --yes
!conda update --channel conda-forge --all --yes
!conda install ipykernel --yes
!conda install -c fastchan numpy=1.19.5 --yes
!pip install Cython
```

Hình 8: Script cái đặt Mini Conda

4.1.3.2. Protobuf

Tensorflow Object detection API sử dụng Protobuf để cấu hình model và huấn luyện các tham số. Sử dụng Protobuf để tạo ra các file python và một số file mã nguồn khác, các file mã nguồn này được sử dụng để huấn luyện model.

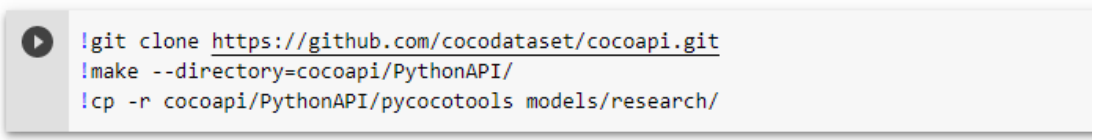
Tải trước pre-trainded-model để tạo ra file python phù hợp với model đó, đồng thời sử dụng công cụ protoc để build và compile Protobuf.



Hình 9: Script cài đặt Protobuf

4.1.3.3. COCO API

Từ các phiên bản tensorflow 2.x, gói pycocotools là một trong các gói cần thiết cho Object Detection API, đồng thời gói này cũng sẽ hỗ trợ cho việc đánh giá độ tin cậy của model sau khi huấn luyện.



Hình 10: Script cài đặt COCO API

4.1.3.4. Object Detection API

Bản chất của việc cài Object Detection API là cài đặt gói object_detection, gói này được gọi trong các file python dùng cho quá trình huấn luyện và đánh giá model.

```
!cp models/research/object_detection/packages/tf2/setup.py models/research/.
!python -m pip install --use-feature=2020-resolver models/research/.
```

Hình 11: Script cài đặt Object Detetion API

4.1.3.5. Cài song song Tensorflow 2.3.0 và Tensorflow 2.5.0

Cả COCO API và Object Detection API đều sử dụng Tensorflow 2.5.0, phiên bản này sử dụng nền cuda 11, phiên bản cuda 11 này đang gặp phải lỗi liên kết thư viện libcusolver11.so, mã nguồn trong các file python vẫn có thể chạy dù không có thư viện này, nhưng thời gian huấn luyện sẽ cực kì lâu do phải chạy toàn bộ quá trình huấn luyện bằng CPU.

Colab có cài đặt sẵn cuda 10.0 nên nhóm em quyết định cài đặt thêm một phiên bản Tensorflow thấp hơn là 2.3.0 để chạy trên nền cuda 10.

```
[ ] !conda install -c fastchan python=3.8.5 --yes
    !pip install tensorflow==2.3.0 tensorflow-gpu==2.3.0
```

```
!python -m pip install --use-feature=2020-resolver models/research/.
```

```
!conda install -c fastchan python=3.8.5 --yes
!pip install tensorflow==2.3.0 tensorflow-gpu==2.3.0
```

Hình 12: Script cài đặt Tensorflow song song

4.1.4. Chuẩn bị các file cần thiết cho quá trình huấn luyện

4.1.4.1. train.record và test.record

File train.record được build từ dataset dùng cho quá trình huấn luyện, thay vì đọc trực tiếp dataset thì model sẽ đọc từ file train.record trong suốt quá trình huấn luyện.

File test.record được build từ dataset dùng cho việc đánh giá độ tin cậy của model sau khi huấn luyện.

Cả train.record và test.record đều được tạo bằng file generate_tfrecord.py.

```
# Tạo file train.record
!python generate_tfrecord.py \
        -x images/train \
        -l annotations/label_map.pbtxt \
        -o annotations/train.record

# Tạo file test.record
!python generate_tfrecord.py \
        -x images/test \
        -l annotations/label_map.pbtxt \
        -o annotations/test.record
```

Hình 13: Script tạo file train.record và test.record

4.1.4.2. pipeline.config

File pipeline.config là file cấu hình của model, nó chứa mọi thiết lập về các hyperparameter như learning rate, match thresold, batch size, step. File này cũng được sử dụng để đánh giá model.

Các thiết lập Hyperparameter trong pipeline.config đều được giữ ở mức mặc định và chỉ thay đổi các tham số sau để model có thể chạy được trên colab mà không đạt tới giới hạn sử dụng GPU và tránh việc tràn VRAM:

- num_class: 3
- batch_size: 4

4.1.4.3. label_map.pbtxt

File label_map.pbtxt chứa thông tin về nhãn mà mô hình sẽ detect ra, đồng thời là thông tin hiển thị cho từng label và số nguyên tương ứng của label đó.

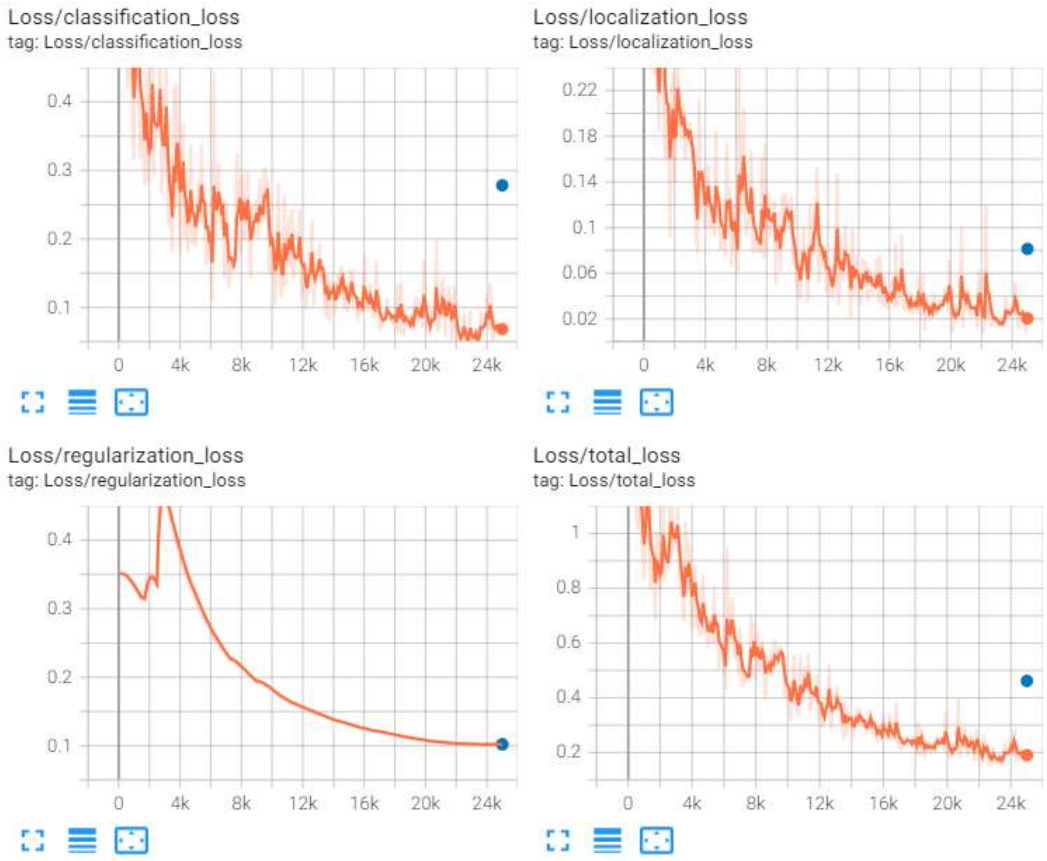
```
[ ]  %%writefile annotations/label_map.pbtxt
     item {
         id: 1
         name: 'correct'
         display_name: 'true'
     }

     item {
         id: 2
         name: 'incorrect'
         display_name: 'wrong'
     }

     item {
         id: 3
         name: 'face'
         display_name: 'face'
     }
```

Hình 14 Nội dung file label_map.pbtxt

4.1.5. Huấn luyện và kiểm thử

4.1.5.1. Huấn luyện

- Thời gian huấn luyện: 4 giờ 28 phút
- Dung lượng VRAM tối đa sử dụng: 5.84 GB
- Loss/classification_loss: 0.06068
- Loss/localization_loss: 0.02738
- Loss/regularization_loss: 0.1021
- Loss/total_loss: 0.2003



Hình 15: Biểu đồ các mất mát trong lúc huấn luyện

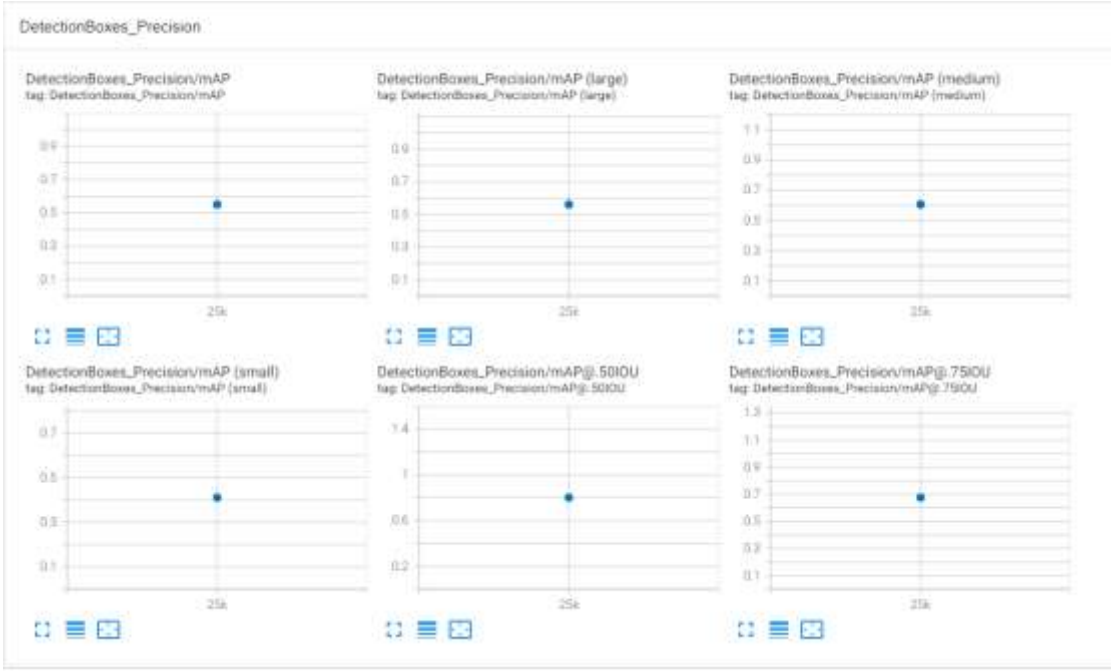
4.1.5.2. Kiểm thử

Đánh giá model ssd_resnet50 dựa vào 3 yếu tố: loss, precision, recall.

- Đánh giá dựa trên precision

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.550
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.802
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.676
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.411
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.606
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.562
```

Hình 16: Avarage Precision



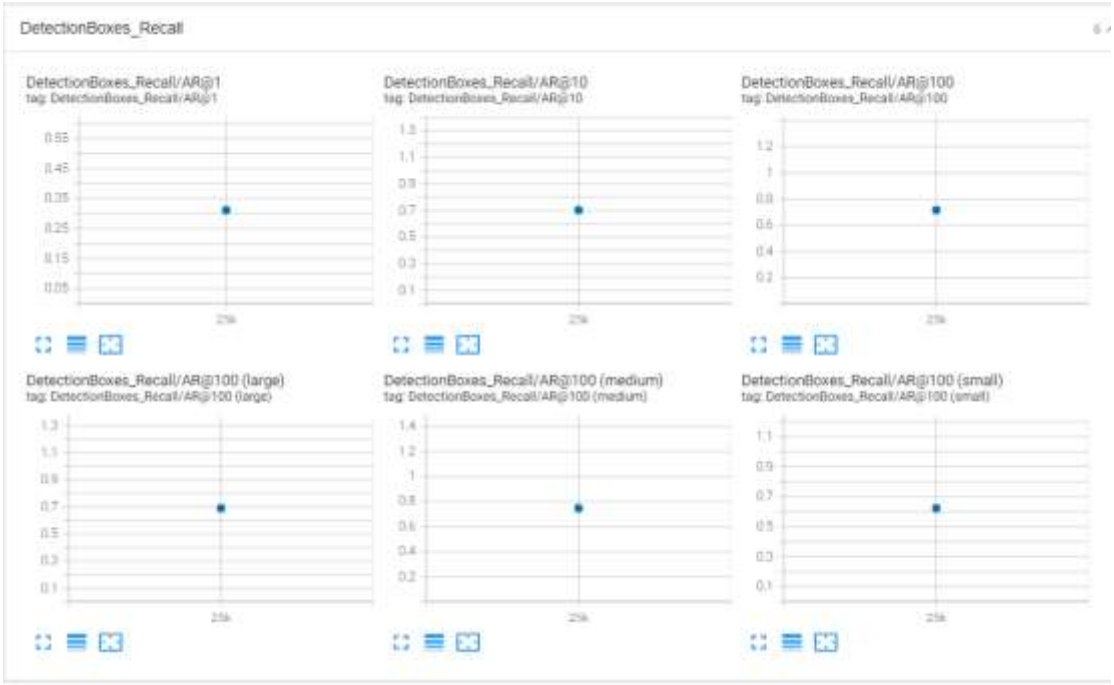
Hình 17: Biểu diễn Avarage Precision

Nhận xét: mô hình cho một giá trị precision ở mức khá (0.550) khi chạy trên toàn bộ dữ liệu của tập kiểm thử, đối với các bbox chỉ đat được 0,5% so với tổng của ground-truth và overlap thì model lai cho một giá trị precision tốt (0.802). Khi model chạy với các đối tượng có kích thước nhỏ (area < 32x32), thì model cho giá trị khá thấp (0.411), tuy nhiên model lai cho giá trị precision tốt với các đối tượng có kích thước vừa (32x32 < area < 96x96) và lớn (area > 96x96) với giá trị lần lượt cụ thể là 0.606 và 0.562.

- Đánh giá dựa trên recall

```
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.311
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.704
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.715
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.623
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.746
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.690
```

Hình 18: Biểu đồ Recall



Hình 19: Biểu diễn Recall

Nhận xét: mô hình cho một giá trị recall ở mức tệ (0.311) khi chạy trên toàn bộ dữ liệu của tập kiểm thử với chỉ 1 lần dự đoán, khi số lần dự đoán được tăng lên thì model lai cho một giá trị recall tốt hơn (0.704 và 0.715). Khi model chạy với các đối tượng có kích thước nhỏ (area < 32x32), vừa (32x32 < area < 96x96) và lớn (area > 96x96) thì model đều cho giá trị recall tốt, cụ thể lần lượt là 0.623, 0.746 và 0.690

- Loss/localization_loss: 0.081277
- Loss/classification_loss: 0.278385
- Loss/regularization_loss: 0.102060
- Loss/total_loss: 0.461722

Nhận xét: Cả localization_loss, regularization_loss đều có giá trị tốt, đối với classification_loss thì giá trị 0.278385 là một giá trị có thể chấp nhận được,

nhưng vẫn khá cao (~0.3), nguyên nhân có thể là bởi sự chênh lệch số lượng nhãn của tập dữ liệu huấn luyện.

Kết luận: Mô hình có khả năng phát hiện tốt (localization_loss: 0.081277), nhưng vẫn khó phát hiện khi chạy với các đối tượng có kích thước nhỏ, khả năng phân loại nhãn của mô hình ở mức tốt, tuy nhiên vẫn còn dễ bị phân biệt sai đối với một số trường hợp khó

Một số kết quả demo:

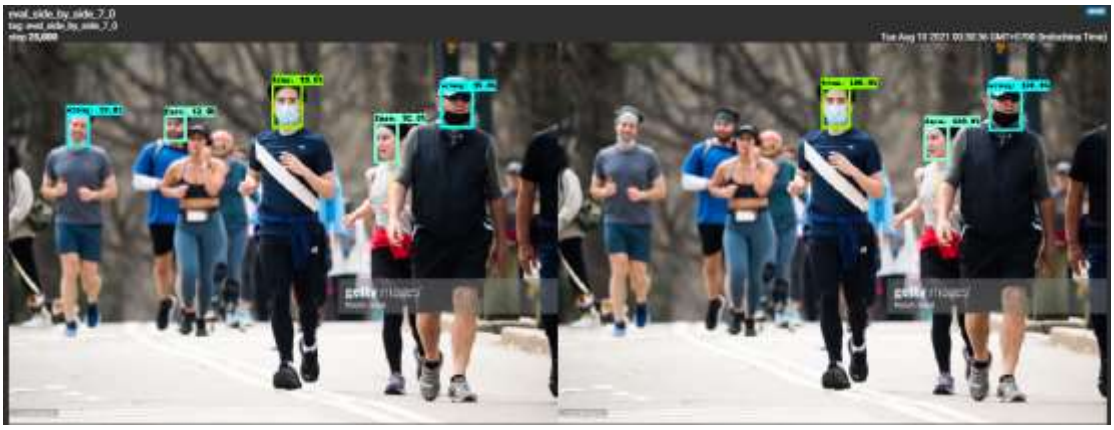
Video 1: https://youtu.be/_6c6eugOQDg

Video 2: https://youtu.be/i3LFg3ZFQ1M

Những ảnh detect sai:

Hình 20: Ảnh phát hiện và phân lớp sai

Hình 21: Ảnh phát hiện và phân lớp sai



Hình 22: Ảnh phát hiện và phân lớp sai

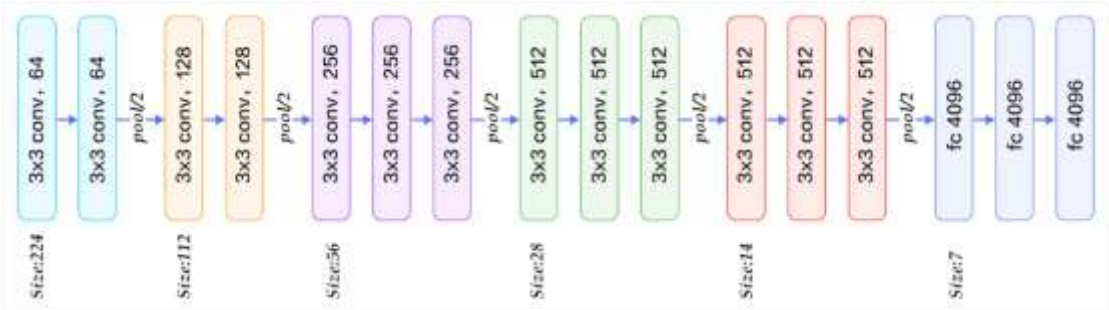
Link colab huấn luyện mô hình: Training_facemask_detector_using_resnet50
Link colab đánh giá mô hình: Evalute_facemask_detector_using_resnet50
Link colab demo kết quả: facemask_detection_using_ssd_resnet50
Link drive chứa mô hình và dataset: ssd_resnet50

4.2. Mô hình Faster RCNN

4.2.1. Tham khảo

Về bài toán phát hiện người đeo khẩu trang đã có nhiều với các mô hình khác nhau. Tuy nhiên, phần lớn bài toán đặt ra là việc phát hiện mang khẩu trang và không mang khẩu trang (2 nhãn).

Với mô hình Faster RCNN cũng đã được sử dụng bài toán phát hiện mang khẩu trang (với 2 nhãn là mang và không mang). Nhóm có tham khảo kết quả trong bài báo Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment. Dữ liệu được sử dụng trong bài báo được các tác giả tạo thủ công một phần và có thêm dữ liệu được thu thập trên mạng với tổng cộng gần 7500 bức hình. Dữ liệu gồm 2 nhãn là **Mask** và **No mask** .Với mô hình Faster-RCNN, các tác giả đã sử dụng kiến trúc ResNet-101-FPN làm xương sống cho mô hình cơ sở. Kết quả thu được của các tác giả là với IoU = 0.5, mAP = 0.62.



(nguồn ảnh www.quora.com/What-is-the-VGG-neural-network)

Nhóm sử dụng kiến trúc VGG16 cho việc phân loại trong mô hình với trọng số *vgg16_weights_tf_dim_ordering_tf_kernels.h5* được tải về từ Releases · fchollet/deep-learning-models (github.com).

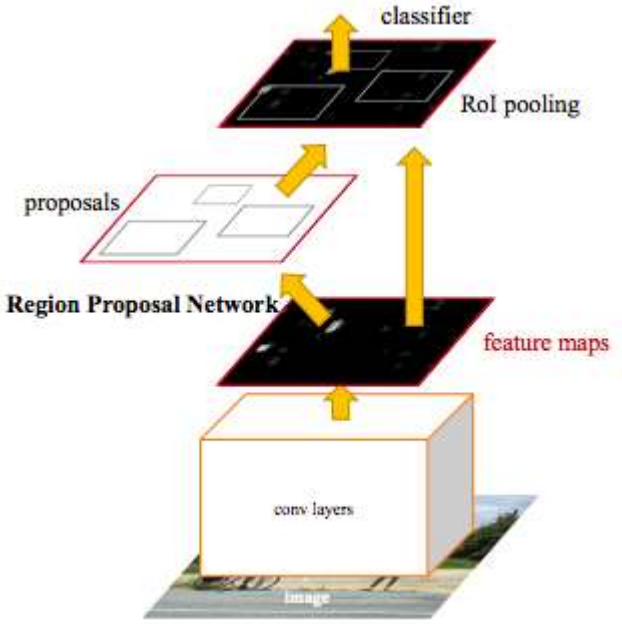
Nhóm thao khảo một số khái niệm và tham số trong bài báo gốc Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks của nhóm tác giả Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun.

Soure code tham khảo của nhóm: Faster_RCNN_for_Open_Images_Dataset_Keras (github.com)

4.2.2. Các khái niệm quan trọng

Faster-RCNN sử dụng 1 mạng con gọi là **RPN (Region Proposal Network)** với mục đích trích xuất ra các vùng có khả năng chứa đối tượng từ ảnh (hay còn gọi là RoI - Region of Interest), khác hoàn toàn với cách xử lý của 2 mô hình anh em trước đó là RCNN và Fast-RCNN

Luồng xử lý của Faster-RCNN có thể tóm gọn lại như sau:



(nguồn: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks - Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun)

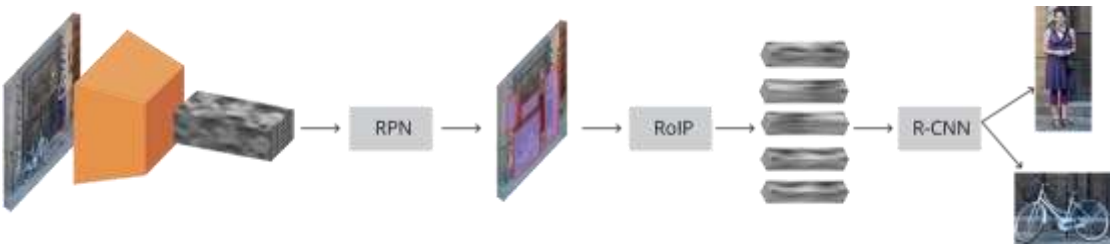
- Input image được đưa qua 1 backbone CNN (cụ thể nhóm sử dụng VGG16), thu được feature map.
- 1 mạng con RPN (gồm các conv layer) dùng để trích rút ra các vùng gọi là *RoI (Region of Interest)*, hay các vùng có khả năng chứa đối tượng từ feature map của ảnh. Input của RPN là feature map, output của RPN bao gồm 2 phần:
    1. Binary object classification để phân biệt đối tượng với background, không quan tâm đối tượng là gì.
    2. Bounding box regression để xác định vùng ảnh có khả năng chứa đối tượng.

    vậy nên RPN bao gồm 2 hàm loss và hoàn toàn có thể được huấn luyện riêng so với cả model.
- Sau khi đi qua RPN, ta sẽ thu được vùng RoI với kích thước khác nhau (W & H) nên sử dụng RoI Pooling (kế thừa từ Fast-RCNN) để thu về cùng 1 kích thước cố định

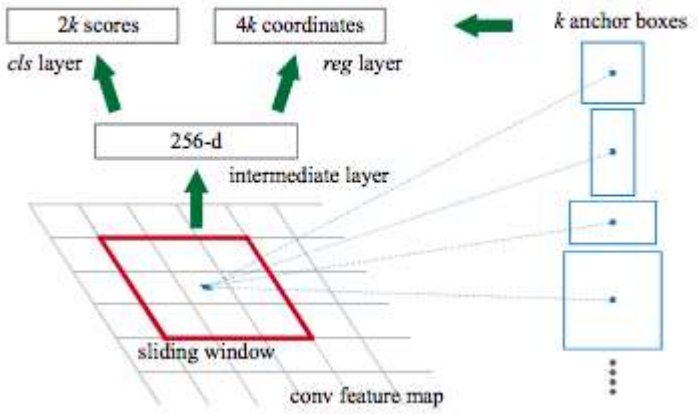
- Thực hiện tách thành 2 nhánh ở phần cuối của model, 1 cho object classification với N + 1 (N là tổng số class, +1 là background) với bounding box regression. Vậy nên sẽ định nghĩa thêm 2 thành phần loss nữa (khá tương tự như RPN) và loss function của Faster-RCNN sẽ gồm 4 thành phần: ***2 loss của RPN, 2 loss của Fast-RCNN***

4.2.2.1. Region Proposal Network (RPN)

Với Faster-RCNN, thay vì việc sử dụng Selective Search (RCNN, Fast RCNN), mô hình được thiết kế thêm 1 mạng con gọi là **RPN (Region Proposal Network)** để trích rút các vùng có khả năng chứa đối tượng của ảnh. Điều này giúp Faster RCNN xử lý nhanh hơn người tiền nhiệm.



(nguồn https://viblo.asia)



(nguồn: [Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks] - Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun)

Về cơ bản, đầu tiên RPN sử dụng 1 conv layer với 512-d với VGG16, kernel_size = (3, 3) lên feature map. Sau đó được chia 2 nhánh, 1 cho binary

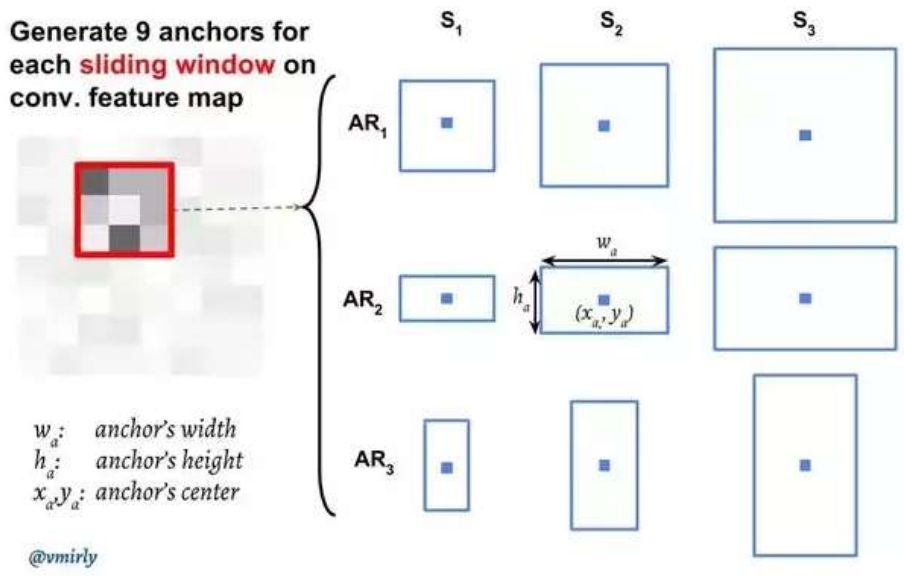
object classification, 1 cho bounding box regression. Cả 2 sử dụng 1 conv layer với kernel_size = (1, 1) nhưng với số channel đầu ra khác nhau.

- Với *binary object classification* thì có 2k channel output, với k là tổng số lượng anchors nhằm xác định rằng 1 anchor có khả năng chứa đối tượng hay là background (không quan tâm đối tượng là gì).
- Với *bounding box regression* thì có 4k channel output, với 4 thể hiện cho 4 tọa độ (x, y, w, h).

```
Args:
    base_layers: vgg in here
    num_anchors: 9 in here
Returns:
    [x_class, x_regr, base_layers]
    x_class: classification for whether it's an object
    x_regr: bboxes regression
    base_layers: vgg in here
"""
x = Conv2D(512, (3, 3), padding='same', activation='relu',
           kernel_initializer='normal', name='rpn_conv1')(base_layers)

x_class = Conv2D(num_anchors, (1, 1), activation='sigmoid',
                 kernel_initializer='uniform', name='rpn_out_class')(x)

x_regr = Conv2D(num_anchors * 4, (1, 1), activation='linear',
                kernel_initializer='zero', name='rpn_out_regress')(x)

return [x_class, x_regr, base_layers]
```

4.2.2.2. Anchor

Anchor là điểm trung tâm của cửa sổ trượt. Các nhà phát triển đã chọn 3 tỷ lệ và 3 tỷ lệ khung hình. Vì vậy, có thể có tổng cộng 9 đề xuất cho mỗi pixel, đây là cách giá trị của k được quyết định, K = 9 cho trường hợp này, k là số anchor. Đối với toàn bộ hình ảnh, số lượng neo là W * H * K.



(nguồn https://viblo.asia)

Các anchor này được gán là positive / negative (object / background) dựa vào diện tích trùng lặp hay IoU overlap với ground truth bounding box theo nguyên tắc sau:

- Các anchor có tỉ lệ IoU lớn nhất với ground truth box sẽ được coi là positive
- Các anchor có tỉ lệ IoU >= 0.7 sẽ được coi là positive
- Các anchor có tỉ lệ IoU < 0.3 sẽ được coi là negative (background)
- Các anchor nằm trong khoảng 0.3 <= x < 0.7 sẽ được coi là neutral (trung tính) là sẽ không được sử dụng trong quá trình huấn luyện mô hình.

4.2.2.3. Roi Pooling

RoI Pooling với mục đích cố định kích thước đầu ra của feature map sau khi thực hiện RoI Pooling. Việc thực hiện RoI Pooling là bắt buộc vì phần cuối của mạng gồm 2 nhánh fully connected layer nên yêu cầu input size phải cố định

4.3.2.4. Loss function (Faster-RCNN model).

loss function của model Faster-RCNN gồm 4 thành phần:

1. RPN classification (binary classification, object or background).
2. RPN regression (anchor -> region proposal).

3. Fast-RCNN classification (over N+1 classes).
4. Fast-RCNN bounding box regression (region proposal -> ground truth bounding box).

Loss function sẽ được định nghĩa với công thức như bên dưới

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

- Trong đó, i là là index của anchor trong 1 mini-batch và p_i là xác suất dự đoán cho anchor tại index thứ i là 1 đối tượng. Chú ý, loss cho bounding box regression chỉ được tính khi anchor là positive (cách xác định anchor là positive / negative / trung tính được đề cập bên trên).

- L_{cls} với RPN là binary_crossentropy để xác định anchor có chứa đối tượng (object) hay không. L_{cls} với Fast-RCNN tại phần cuối của mô hình là categorial_crossentropy loss với (N + 1) lớp

```
def rpn_loss_cls_fixed_num(y_true, y_pred):
    return lambda_rpn_class * float(K.sum(
        float(y_true[:, :, :, :num_anchors]) * float(K.binary_crossentropy(float(y_pred[:, :, :, :]),
                                                                           float(y_true[:, :, :, num_anchors:])))) / K.sum(
                                                                                   epsilon + float(y_true[:, :, :, :num_anchors])))
```

```
def class_loss_cls(y_true, y_pred):
    return lambda_cls_class * float(K.mean(categorical_crossentropy(float(y_true[0, :, :]), float(y_pred[0, :, :]))))
```

- L_{reg} là loss tính cho bounding box regression. Trong bài báo gốc tác giả đề cập đến Loss Smooth L1 Loss. Công thức của Smooth L1 loss dạng Huber-loss được định nghĩa như sau:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

```
def rpn_loss_regr_fixed_num(y_true, y_pred):

    # x is the difference between true value and predicted vaue
    x = float(float(y_true[:, :, :, 4 * num_anchors:]) - float(y_pred))

    # absolute value of x
    x_abs = K.abs(x)

    # If x_abs <= 1.0, x_bool = 1
    x_bool = K.cast(K.less_equal(x_abs, 1.0), tf.float32)

    return float(lambda_rpn_regr) * float(K.sum(
        float(y_true[:, :, :, :4 * num_anchors]) * float((x_bool * (0.5 * x * x) + (1 - x_bool) * (x_abs - 0.5)))) / K.sum(
                                                                                    epsilon + float(y_true[:, :, :, :4 * num_anchors])))
```

```
def class_loss_regr_fixed_num(y_true, y_pred):
    x = float(float(y_true[:, :, 4*num_classes:]) - float(y_pred))
    x_abs = K.abs(x)
    x_bool = K.cast(K.less_equal(x_abs, 1.0), 'float32')
    return lambda_cls_regr * float(K.sum(
            float(y_true[:, :, :4*num_classes]) * float((x_bool * (0.5 * x * x) + (1 - x_bool)) * (x_abs - 0.5))) / K.sum(
                epsilon + float(y_true[:, :, :4*num_classes])))
```

4.2.3. Thiết lập training

4.2.3.1. Môi trường

Môi trường: Google's Colab với Tesla K80 GPU.

4.2.3.2. Chuẩn bị training labels

Như nhóm em đã trình bày ở phần trên, dữ liệu của nhóm gồm 3 nhãn là

+ face : không mang khẩu trang

+ correct mang khẩu trang đúng cách

+ incorrect: mang khẩu trang không đúng cách.

Dữ liệu sau khi gán nhãn được xuất sang một file .csv bao gồm các cột như sau:

	filename	width	height	class	xmin	ymin	xmax	ymax
0	0.png	716	477	face	232	273	257	308
1	0.png	716	477	face	164	270	195	322

Để phù hợp với mô hình faster R-CNN mà nhóm em sử dụng, nhóm em chuyển đổi sang file annotation.txt với các cột dữ liệu là *filename, xmin, ymin, xmax, ymax, class,* các giá trị cách nhau bởi dấu phẩy ",".

Các ảnh trong tập train có kích thước giống nhau là 716x477, tuy nhiên, để quá trình đào tạo nhanh hơn và cũng như là tiết kiệm được tài nguyên trong Google Colab nên phần nào ảnh sẽ được resize thành kích thước 300x300.

4.2.3.3. Training

4.2.3.3.1. Lưu và tải lại trọng số mô hình

Với việc sử dụng GPU trên Google Colab miễn phí, chỉ có thể sử dụng trong 3, 4 giờ/ngày sẽ bị ngắt, tuy nhiên quá trình train model diễn ra lâu hơn như vậy. Ngoài ra, một vấn đề khác nữa là khi train quá quá 30 epoch (khoảng 3.5 giờ) thì sẽ bị tràn RAM dẫn đến restart runtime.

Cho nên không thể đặt một số epoch cố định ngay từ đầu và đào tạo một mạch liền được. Thay vào đó, nhóm em sẽ chạy 30 epoch mỗi lần train và lưu lại trọng số mô hình vào một file .hdf5 và lưu lại các giá trị loss, accuracy ở mỗi epoch vào một file .csv thông qua Google Drive.

```
if s.path.isfile(C.model_path):
    # If this is a continued training, load the trained model from before
    print('Continue training based on previous trained model')
    print('Loading weights from {}'.format(C.model_path))
    model_rpn.load_weights(C.model_path, by_name=True)
    model_classifier.load_weights(C.model_path, by_name=True)
    # Load the records
    record_df = pd.read_csv(record_path)
```

Tổng số epoch mà nhóm đã đào tạo là 218 epoch, ***thời gian đào tạo khoảng 14 giờ***. Một vấn đề về việc lưu trọng số mô hình, ở đây, nhóm em không lưu trọng số mô hình ở mỗi epoch mà chỉ lưu trọng số ở epoch nào có giá trị loss thấp hơn giá trị loss trước đó được lưu. Việc này đảm bảo được các giá trị tốt nhất cho mô hình.

```
if len(record_df)==0:   #Khi train từ đầu
    best_loss = np.Inf
else:                   #Khi tiếp tục train
    best_loss = np.min(r_curr_loss)

if curr_loss < best_loss:
    print('Total loss decreased from {} to {}, saving weights'.format(best_loss,curr_loss))
    best_loss = curr_loss
    model_all.save_weights(C.model_path)
```

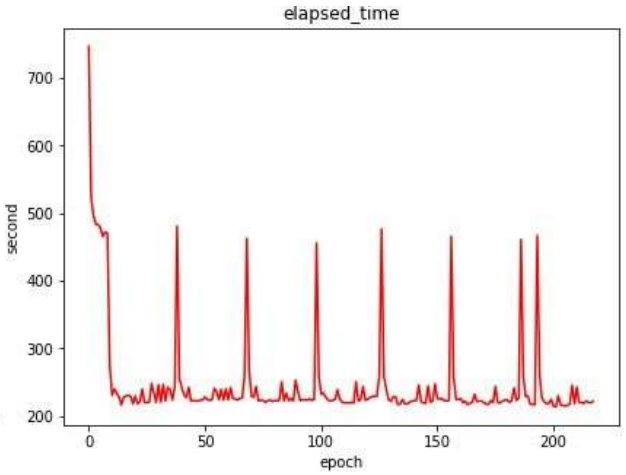
Trọng số được lưu nhiều ở 100 epoch đổ lại, trên 100 epoch, thì càng ít thỏa điều kiền để lưu trọng số.

4.2.3.3.2. Thời gian train và test

Như đã trình bày ở *4.3.3.3.1. Lưu và tải lại trọng số mô hình*, tổng số epoch đã đào tạo là 218 với thời gian khoảng 14 giờ. Nhóm em sẽ trình bày sâu hơn tại sao lại có con số 14 giờ đó.

	epoch	elapsed_time
0	1	746.76
1	2	523.26
2	3	496.62
3	4	484.14
4	5	483.00
..	...	...
213	214	218.76
214	215	222.36
215	216	220.44
216	217	219.72
217	218	221.76

[218 rows x 2 columns]
sum elapsed time: 53952
average elapsed time a epoch: 247.4879



Do đã lưu thời gian mỗi epoch vào file record.csv nên không khó để nhóm em vẽ lại biểu đồ và tính thời gian trung bình hoàn thành một epoch.

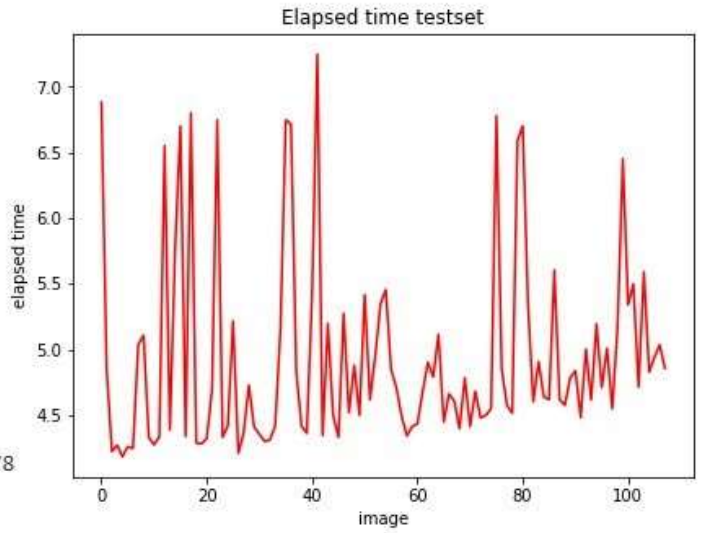
Tổng thời gian thực hiện 218 epoch là 53952 giây, tương đương với 14.9866 ~ 15 giờ. 15 giờ là con số chính xác hơn khoảng dự đoán 14 giờ dựa vào thời gian thực hiện 30 epoch mỗi lần đào tạo. Trung bình một epoch thực hiện trong 247.4879 giây.

Mỗi batch mô hình xử lý một ảnh, do tập train có 705 ảnh nên nhóm chọn độ dài cho mỗi epoch là 705.

Thử nghiệm trên tập test gồm 108 ảnh của nhóm được thời gian sau:

	image	time
0	0	6.880546
1	1	4.841651
2	2	4.222525
3	3	4.269569
4	4	4.182409
..	...	...
103	103	5.590183
104	104	4.827693
105	105	4.939421
106	106	5.034708
107	107	4.855589

[108 rows x 2 columns]
average Elapsed time: 4.93378



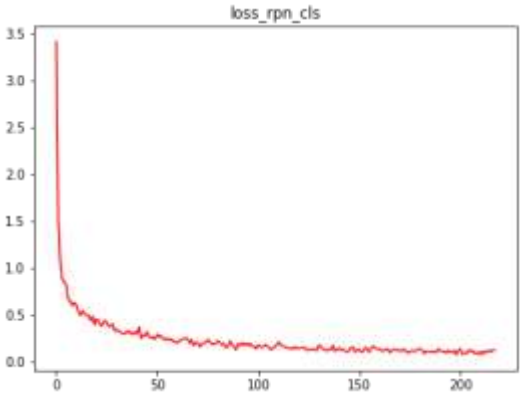
Thời gian trung bình trong dự đoán mỗi ảnh trong tập test set là 4.93378 giây. Đó là thời gian trung bình khi kiểm tra từng ảnh riêng biệt, khi thử nghiệm trên 2 video (nguồn www.gettyimages.dk)

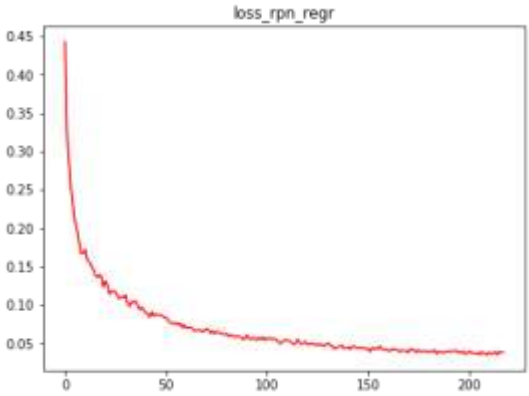
1. Video 1 (dài 13 giây) :Kết quả demo_1, thời gian dự đoán là 1489.871 giây (~24 phút) .
2. Video_2: (dài 20 giây): Kết quả của demo_2, thời gian dự đoán là 2165.228 giây.

4.2.3.4. Đánh giá mô hình Faster RCNN

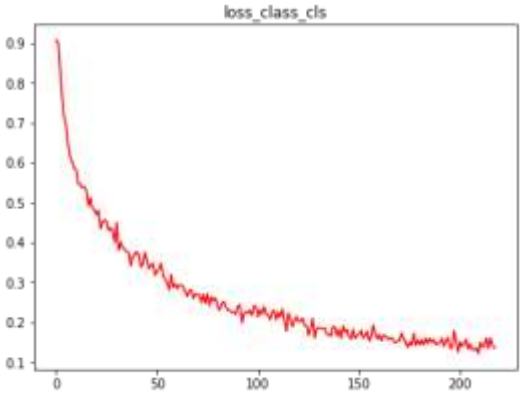
Như nhóm đã đề cập ở phần lý thuyết trên, mô hình RPN có hai đầu ra. Một là *binary object classification* để phân loại xem đó có phải là một đối tượng hay không và hai là *bounding box regression* để trả về tọa độ của các hộp giới hạn. Từ hình bên dưới, nhóm em nhận thấy rằng nó học rất nhanh trong 25 epoch đầu tiên. Sau đó, nó trở nên chậm hơn đối với binary object classification trong khi bounding box regression vẫn tiếp tục đi xuống.

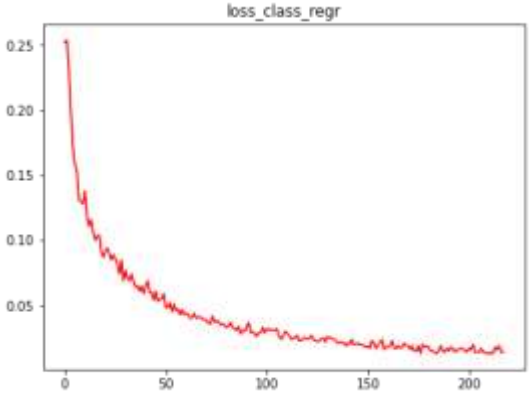
Lý do cho điều này có thể là độ chính xác của đối tượng đã cao trong giai đoạn đầu của quá trình đào tạo của chúng tôi, nhưng đồng thời, độ chính xác của tọa độ các hộp giới hạn vẫn còn thấp và cần thêm thời gian để tìm hiểu.



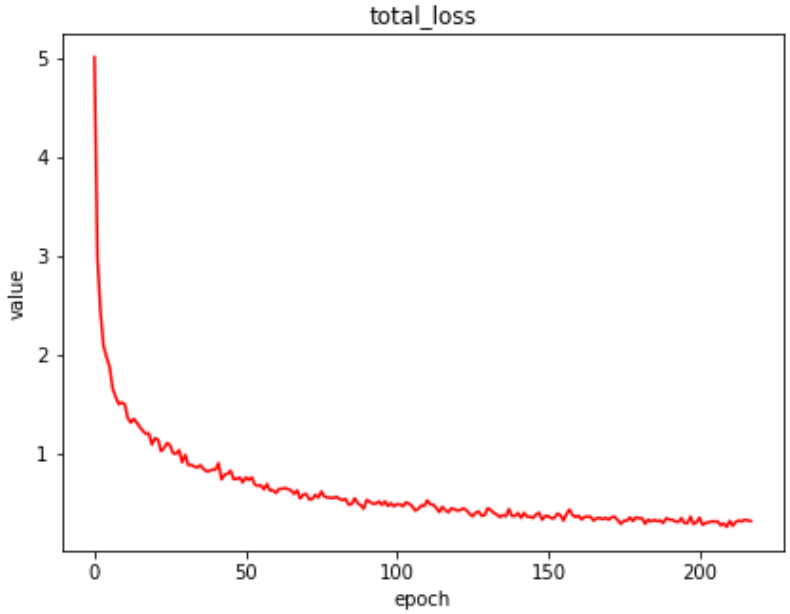


Quá trình học tập tương tự được thể hiện trong mô hình Classifier. So với hai biểu đồ của mô hình RPN trên, nhóm em thấy nó có xu hướng tương tự. So sánh với hai biểu đồ dưới, nhóm em thấy rằng dự đoán đối tượng dễ dàng hơn dự đoán tên lớp của một bbox.





Total_loss này là tổng của 4 loss trên. Nó có xu hướng giảm dần. Tuy nhiên một trong những lý do mà nhóm em dừng ở epoch 218 là do nhận thấy giá trị total_loss ít giảm từ epoch 200 trở đi. Bên cạnh đó là lý do tiết kiệm thời gian nên nhóm em dừng ở epoch 218.



Các giá trị Loss:

1. loss_rpn_cls (binary object classification): **0.078**
2. loss_rpn_regr (bounding box regression): **0.035**
3. loss_class_cls (Fast-RCNN classification): **0.122**
4. loss_class_regr (Fast-RCNN bounding box regression): **0.013**
5. total_loss: **0.255**

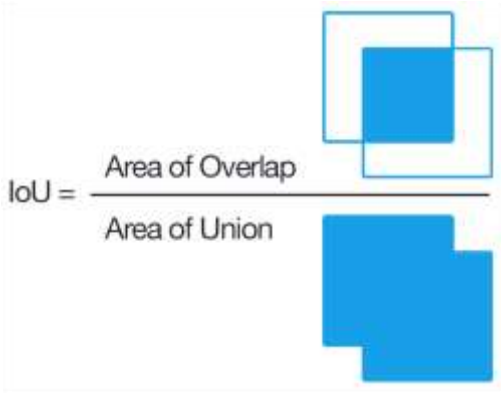
Nhận xét: cả 4 giá trị loss_rpn_cls, loss_rpn_regr, loss_class_cls, loss_class_regr đều có giá trị tốt và nhỏ. Trong đó loss_class_cls có giá trị lớn nhất trong 4 giá trị loss đang xét là 0.122. Nguyên nhân có thể là bởi sự chênh lệch số lượng nhãn của tập dữ liệu huấn luyện.

Kết luận: Mô hình có khả năng phát hiện tốt (total_loss = 0.225), nhưng vẫn khó phát hiện khi chạy với các đối tượng có kích thước nhỏ và với những vật di chuyển nhanh.

Các metric đánh giá:

- **IoU**

IoU hay Intersection over Union là độ đo để biểu diễn độ tương đồng giữa ground truth bounding box với predicted bounding box của mô hình, giá trị trong [0,1], 2 box càng khớp nhau thì giá trị càng gần 1 và ngược lại.



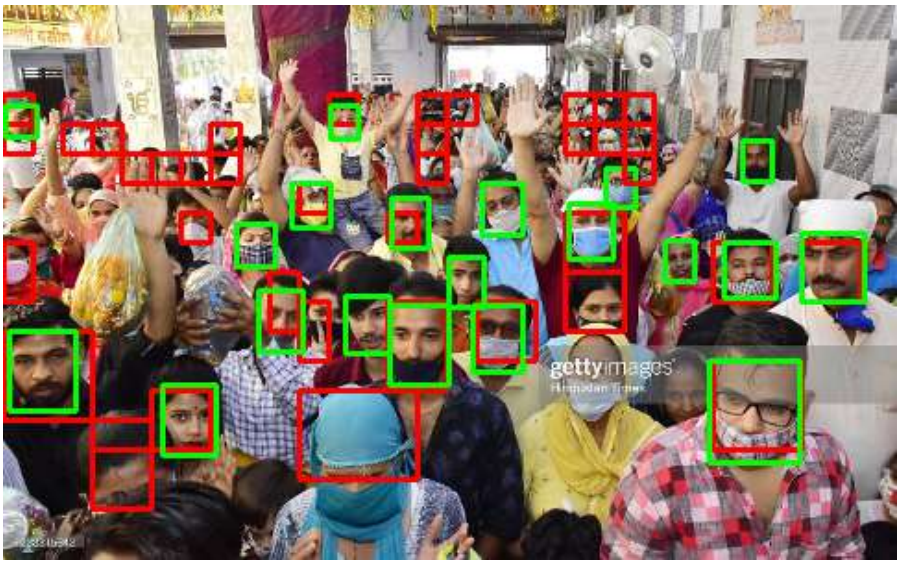
Màu xanh lục là ground truth bounding box, màu đỏ là predicted bounding box.

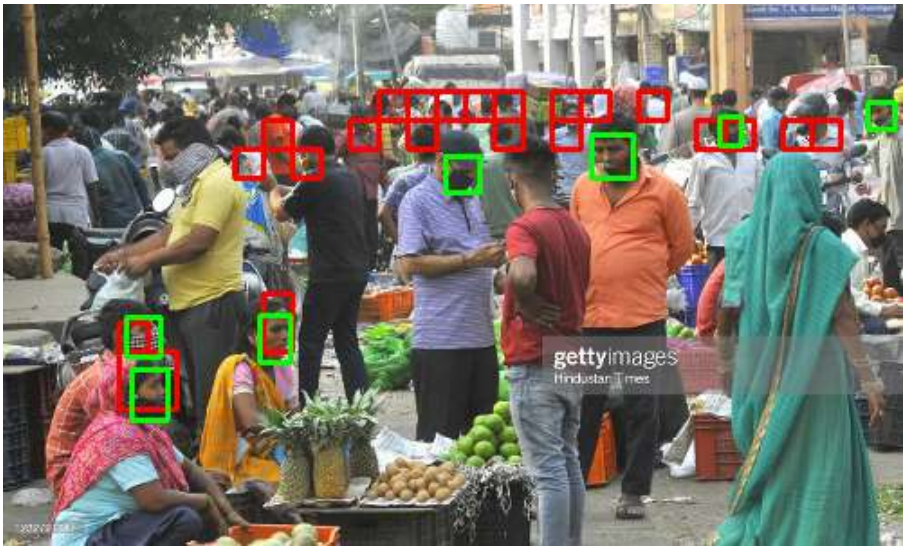
mAP (mean Average Precision): 0.542

Một số trường hợp dự đoán của mô hình

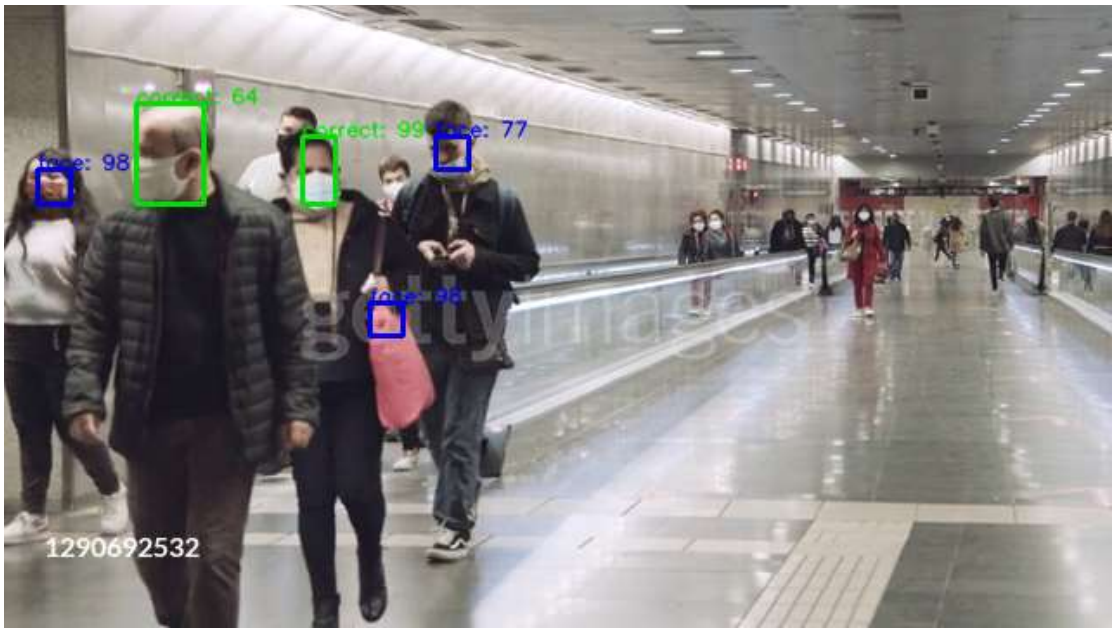
- Đối với ảnh nhiều người

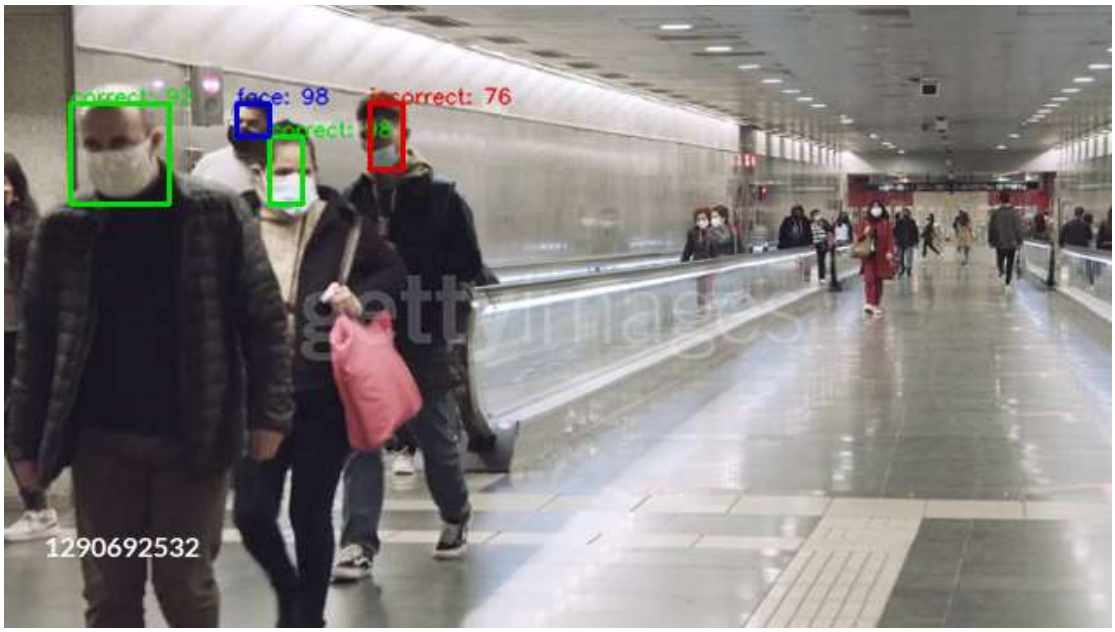

gettyimages
China News Service


gettyimages


gettyimages
Hindustan Times

- Khi gán nhãn thì những chọn những khuôn mặt rõ, hoặc hơi mờ để gán, còn những khuôn mặt quá nhỏ thì nhóm bỏ qua. Tuy nhiên khi mô hình dự đoán các hộp boxes mô hình sẽ bị "loạn" với ảnh có nhiều người (có nhiều khuôn mặt, đầu,…) dẫn đến mô hình có những boxes nhỏ nối nhau. Tuy nhiễn boxes này không ảnh hưởng đến kết quả dự đoán của mô hình.





- Với những trường hợp mang khẩu trang sai, mô hình dễ dự đoán thành không mang khẩu trang. Nguyên nhân có thể do người đó cúi mặt xuống hoặc khi kéo khẩu trang sát cổ nên mô hình xác định boxes là khuôn mặt.
- Ngoài ra thì mô hình còn nhận diện sai phần tay của người (bàn tay, cánh tay) thành một đối tượng khuôn mặt.

4.2.4. Link bài làm, demo của model faster R-CNN

1. Link Colab train model: cs114_frcnn_train_vgg16.ipynb
2. Link Colab test model: cs114_frcnn_test_vgg16.ipynb
3. Link Drive chứa các video demo: F-RCNN_VideoDemo_ObjectDetect

CHƯƠNG 5. ỨNG DỤNG VÀ HƯỚNG PHÁT TRIỂN

5.1. ỨNG DỤNG

Đối với mô hình sử dụng ssd_resnet50 do số lượng convolutional layer nhiều, không nên chạy mô hình này với những chiếc máy tính chỉ có CPU do sẽ mất rất nhiều thời gian để xử lý xong một frame, máy tính cần có một chiếc VGA để tăng tốc độ tính toán và giảm thời gian xử lý một frame. Do không có chi phí để lắp đặt được một hệ thống máy tính có VGA nên nhóm vẫn chưa xây dựng hoàn thiện ứng dụng để thực thi model, nhóm mới xây dựng xong phần detect và vẫn chưa xây dựng xong phần lấy được khuôn mặt của các đối tượng không đeo khẩu trang hoặc đeo khẩu trang sai cách rồi hiển thị lên một cửa sổ riêng.

Video về sản phẩm của nhóm: https://youtu.be/S_kK8tD0f8c

Cấu hình máy tính nhóm đề xuất để chạy realtime (25FPS) với ssd_resnet50:

- CPU: Intel Core i3 8100
- VGA: RTX 3060
- RAM: 8GB
- PSU: Bất kì PSU có công suất 650W trở lên

5.2. HƯỚNG PHÁT TRIỂN

Để model chạy realtime thì cần một hệ thống máy tính có VGA, nhưng ở hiện tại (10/08/2021) giá của một chiếc VGA phổ thông vẫn ở mức cao do cơn sốt về Bitcoin, dẫn đến chi phí lắp đặt một bộ máy tính để chạy model khá cao nên nhóm vẫn chưa có hướng phát triển về sau cho model này.

TÀI LIỆU THAM KHẢO

[1] S. Sethi, M. Kathuria, và T. Kaushik, "Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread", *J Biomed Inform*, vol 120, tr 103848, tháng 8 2021, doi: 10.1016/j.jbi.2021.103848.

[2] S. Singh, U. Ahuja, M. Kumar, K. Kumar, và M. Sachdeva, "Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment", *Multimed Tools Appl*, tr 1–16, tháng 3 2021, doi: 10.1007/s11042-021-10711-8.

[3] S. Ren, K. He, R. Girshick, và J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *arXiv:1506.01497 [cs]*, tháng 1 2016, Truy cập: tháng 8 10, 2021. [Online]. Available at: http://arxiv.org/abs/1506.01497

[4] Y. Xu, "Faster R-CNN (object detection) implemented by Keras for custom data from Google's Open Images…", *Medium*, tháng 2 25, 2019. https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a (truy cập tháng 8 10, 2021).

[5] "[Deep Learning] - Thuật toán Faster-RCNN với bài toán phát hiện đường lưỡi bò - Faster-RCNN object detection algorithm for Nine-dash-line detection!", *Viblo*, tháng 5 25, 2020. https://viblo.asia/p/deep-learning-thuat-toan-faster-rcnn-voi-bai-toan-phat-hien-duong-luoi-bo-faster-rcnn-object-detection-algorithm-for-nine-dash-line-detection-bJzKmREOZ9N (truy cập tháng 8 10, 2021).

[6] "Training Custom Object Detector — TensorFlow 2 Object Detection API tutorial documentation". https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#preparing-the-dataset (truy cập tháng 8 10, 2021).

[7] *Welcome to the Model Garden for TensorFlow*. tensorflow, 2021. Truy cập: tháng 8 10, 2021. [Online]. Available at: https://github.com/tensorflow/models/blob/a2c19be8c0ff912a507b6c2789228b51e731bf9a/research/object_detection/g3doc/tf2_detection_zoo.md