**Student: Tran Minh Hieu.**

**ID: 17110135.**

## Giới thiệu đề tài:

Vấn đề mang tính thời sự hiện tại của Việt Nam nói riêng và cả thế giới nói chung là vấn nạn dịch bệnh COVID-19 đang hoành hành khắp nơi, gieo rắc cái chết khắp nơi trên thế giới. Ta có thể thấy được mức độ nghiêm trọng của dịch bệnh này bởi số người tử vong lên đến hàng triệu chỉ trong một ngày. Sau đây là báo cáo thống kê cho đến ngày 5/6/2021:



Figure 1 Số liệu thống kế số người tử vong và ca nhiễm Covid-19 đến ngày 5/6/2021

Qua bảng số liệu trên ta thấy được sự gia tăng của số lượng ca nhiễm và tử vong của vi-rút Corona gây ra. Và các con đường lây lan của vi-rút Corona này:



Figure 2 Những con đường lây nhiễm vi-rút Corona (COVID-19).

Ta nhận thấy rõ ràng 2 con đường “nhanh” nhất để nhiễm bệnh đó là qua không khí. Và cách phòng chống chủ động cho 2 con đường lây lan này là giãn cách và đeo khẩu trang để tránh vi-rút trong không khí.

Vì vậy công cuộc xem người dân có tuân thủ việc đeo khẩu trang khi di chuyển hay sinh hoạt nơi đông người hay không giúp ích rất nhiều trong việc phòng và chống lay nhiễm COVID-19, từ đó giảm số lượng người bị lây nhiễm, giảm số ca tử vong và giúp sức khỏe trong cộng đồng được ổn định. Chính vì vậy chúng tôi đã cố gắng xây dựng một ứng dụng có thể phát hiện ra người có đeo khẩu trang hay không và đeo có đúng cách hay không.

## Mô tả các thư viện được sử dụng

### Keras

**Keras là gì?**

Keras chạy trên các thư viện máy mã nguồn mở như TensorFlow, Theano hoặc Bộ công cụ nhận thức (CNTK). Theano là một thư viện python được sử dụng cho các tác vụ tính toán số nhanh. TensorFlow là thư viện toán học biểu tượng nổi tiếng nhất được sử dụng để tạo mạng nơ-ron và mô hình học sâu. TensorFlow rất linh hoạt và lợi ích chính là tính toán phân tán. CNTK là khung học sâu được phát triển bởi Microsoft. Nó sử dụng các thư viện như Python, C #, C ++ hoặc các bộ công cụ học máy độc lập. Theano và TensorFlow là những thư viện rất mạnh nhưng khó hiểu để tạo mạng nơ-ron.

Keras dựa trên cấu trúc tối thiểu, cung cấp một cách dễ dàng và dễ dàng để tạo các mô hình học sâu dựa trên TensorFlow hoặc Theano. Keras được thiết kế để xác định nhanh các mô hình học sâu. Chà, Keras là một lựa chọn tối ưu cho các ứng dụng học sâu.

**Tính năng của Keras**

* API nhất quán, đơn giản và có thể mở rộng.
* Cấu trúc tối thiểu - dễ dàng đạt được kết quả mà không cần rườm rà.
* Hỗ trợ nhiều nền tảng và backend.
* Thân thiện với người dùng chạy trên cả CPU và GPU.
* Khả năng mở rộng tính toán cao.

**Lợi ích**

Keras năng động , mạnh mẽ và có những ưu điểm sau:

* Cộng động lớn hỗ trợ.
* Dễ dàng để kiểm tra.
* Mạng nơ-ron Keras được viết bằng Python giúp mọi thứ đơn giản hơn.
* Keras hỗ trợ cả mạng convolution và recurrent.
* Mô hình học sâu là các thành phần rời rạc, do đó, bạn có thể kết hợp thành nhiều cách.

### Open-CV

**Open-cv là gì?**

* OpenCV viết tắt của Open Source Computer Vision được xem là một trong những thư viện mã nguồn mở hàng đầu cho việc xử lý ảnh theo thời gian thực.
* OpenCV chính thức được ra mắt đầu tiên vào năm 1999, OpenCV là thư viện mã nguồn mở miễn phí cho cả học thuật và thương mại.
* OpenCV hỗ trợ đa nền tảng gồm Windows, Linux, Mac OS, iOS và Android.
* OpenCV hỗ trợ ngôn ngữ lập trình C/ C++, Python và Java.
* OpenCV được viết bằng C / C ++ và tích hợp OpenCL.
* Thư viện có hơn 2500 thuật toán được tối ưu hóa.

**Chức năng có trong thư viện OpenCV**

Thư viện OpenCV bao gồm một số tính năng nổi bật như:

* Bộ công cụ hỗ trợ 2D và 3D
* Nhận diện khuôn mặt
* Nhận diện cử chỉ
* Nhận dạng chuyển động, đối tượng, hành vi,
* Tương tác giữa con người và máy tính
* Điều khiển Robot
* Hỗ trợ thực tế tăng cường

**Ứng dụng OpenCV trong thế giới thực**

OpenCV hiện có hơn 47 nghìn người dùng và ước tính hơn 14 triệu số lượt tải xuống, thư viện OpenCV được sử dụng rộng rãi trong nhiều lĩnh vực.

* Giám sát tự động
* Tìm kiểm, phục hồi, xử lý ảnh
* Nhận dạng khuôn mặt, cử chỉ.
* Nhận dạng chữ viết, con số, ký tự.

### Tensorflow

**Tensorflow là gì?**

Tensorflow là thư viện mã nguồn mở, được các nhà phát triển công nghệ sử dụng chủ yếu ở lĩnh vực Machine learning với mục đích chính là làm cho tốc độ được gia tăng và hoạt động dễ dàng hơn.

**Kiến trúc của Tensorflow**

Trong Tensorflow gồm có 3 phần kếu cấu:

* Phần tiền xử lí dữ liệu
* Phần dựng model
* Phần train, ước tính model

**Lợi ích**

* Tạo cái nhìn tổng quan về dữ liệu
* Dễ chỉnh sửa
* Được goodle hỗ trợ phat triển

**Ứng dụng**

Thực chất thì hiện nay TensorFlow đã được ứng dụng rất phổ biến và được nhiều người sử dụng. Nó cho phép người dùng khám phá các vấn đề như: dịch google, phân tích tình cảm, tóm tắt văn bản, nhận diện hình ảnh….  TensorFlow này đang được dùng phổ biến ở nhiều công ty nổi tiếng trên thế giới như: Facebook, SAP,  Instagram, Intel, Airbnb, Dropbox, eBay, Snapchat, Twitter, Google… Các ứng dụng này đều được đánh giá rất cao về sự phân tích sâu của nó.

### Matplotlib

**Matplotlib là gì?**

Matplotlib là một thư viện vẽ đồ họa được sử dụng cho đồ họa 2D bằng ngôn ngữ lập trình python . Nó có thể được sử dụng trong các tập lệnh python, shell, máy chủ ứng dụng web và các bộ công cụ giao diện người dùng đồ họa khác.

**Thành phần của Matplotlib**

Một Matplotlib figure có thể được phân loại thành nhiều phần như dưới đây:

* ***Figure***: Như một cái cửa sổ chứa tất cả những gì bạn sẽ vẽ trên đó.
* ***Axes***: Thành phần chính của một figure là các axes (những khung nhỏ hơn để vẽ hình lên đó). Một figure có thể chứa một hoặc nhiều axes. Nói cách khác, figure chỉ là khung chứa, chính các axes mới thật sự là nơi các hình vẽ được vẽ lên.
* ***Axis***: Chúng là dòng số giống như các đối tượng và đảm nhiệm việc tạo các giới hạn biểu đồ.
* ***Artist***: Mọi thứ mà bạn có thể nhìn thấy trên figure là một artist như Textobjects, Line2D objects, collection objects. Hầu hết các Artists được gắn với Axes.

**Lợi ích**

Nó là một thư viện vẽ đồ thị rất mạnh mẽ, hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotib là Pyplot. Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở.

**Ứng dụng**

Vẽ đồ thị

### PIL

**PIL là gì?**

Python cho phép giải quyết vấn đề các về xử lý hình ảnh thông qua thư viện **Imaging (PIL).** Thư viện này hỗ trợ nhiều định dạng tập tin, và cung cấp khả năng xử lý hình ảnh và đồ hoạ mạnh mẽ.

**Ứng tụng**

* Imaging cho phép bạn tạo được một hình ảnh trong Python với đầy đủ kích thước, màu sắc
* Cho phép mở một tấm ảnh đã có sẵn lên để thực hiện các thao tác khác như: Sao chép, chỉnh sửa,…
* Hiển thị ảnh
* Lưu ảnh

### Tqdm

**Tqdm là gì?**

tqdm là một tiện ích của python, từ này có nghĩa là “tiến trình” trong tiếng Ả Rập (taqadum, تقدّم) . Trong python, nó giúp hiển thị các vòng lặp dưới dạng một giao diện tiến độ một cách thông minh – chỉ cần bọc bất kỳ vòng lặp nào bằng tqdm và bạn không phải lo cài đặt hiển thị tiến trình cho nó nữa.

**Cài đặt**

Vì tqdm là một phần của Chỉ mục gói Python ( [PyPI](https://pypi.org/) ), nó có thể được cài đặt bằng lệnh **pip install tqdm**

### Mtcnn

**mtcnn là gì?**

MTCNN là viết tắt của Multi-task Cascaded Convolutional Networks. Nó là bao gồm 3 mạng CNN xếp chồng và đồng thời hoạt động khi detect khuôn mặt. Mỗi mạng có cấu trúc khác nhau và đảm nhiệm vai trò khác nhau trong task. Đầu ra của MTCNN là vị trí khuôn mặt và các điểm trên mặt như: mắt, mũi, miệng…

**Cài đặt**

Hiện tại nó chỉ được hỗ trợ Python3.4 trở đi. Nó có thể được cài đặt thông qua pip:

$ pip install mtcnn

**Ứng dụng**

Sử dụng trong nhận diện khuôn mặt.

### Tham khảo :

<https://nguyenvanhieu.vn/thu-vien-tensorflow/#thu-vien-tensorflow-la-gi>

<https://teamvietdev.com/opencv-la-gi-ung-dung-opencv-trong-the-gioi-thuc/>

<https://timviec365.com/blog/tensorflow-la-gi-new2591.html>

<https://ichi.pro/vi/thanh-tien-trinh-cho-python-voi-tqdm-5341342649164>

## Dữ liệu (Dataset) tham khảo và Xử lý dữ liệu.

### Dữ liệu tham khảo.

Hiện nay nguồn dữ liệu về nhận diện khuôn mặt có đeo khẩu trang hay không rất nhiều trên internet với nhiều nguồn khác nhau có thể kể đến như Kaggle,…Và dưới đây là một số nguồn dataset mà chúng tôi đã sử dụng để triển khai ứng dụng:

***RMFD***

* ***Tác giả***: ***Baojin Huang***.
* ***Repositories***: [X-zhangyang/Real-World-Masked-Face-Dataset: Real-World Masked Face Dataset](https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset)
* ***Bài báo liên quan***: [Masked Face Recognition Dataset and Application](https://arxiv.org/abs/2003.09093).
* ***Chi tiết***: Tác giả đã thu thập dữ liệu các mẫu từ trang web. Sau khi làm sạch và dán nhãn, nó chứa ***5.000*** khuôn mặt che của ***525*** người và ***90.000*** khuôn mặt bình thường. (Thực tế )
* ***Nhãn dán***: correct mask (2203), not mask (90468).
* ***Link download***: [RMFD.zip](https://drive.google.com/file/d/1UlOk6EtiaXTHylRUx2mySgvJX9ycoeBp/view)

**Face Mask Detection ~12K Images Dataset**

* ***Tác giả***: ***Ashish Jangra***
* ***Chi tiết***: Tham khảo từ nguồn CelebFace dataset được tạo bởi Jessica Li (<https://www.kaggle.com/jessicali9530>)
* ***Nhãn dán***:

|  |  |  |
| --- | --- | --- |
|  | correct mask | not mask |
| Train | 5000 | 5000 |
| Validation | 400 | 400 |
| Test | 483 | 509 |

* ***Link download***: [Face Mask Detection ~12K Images Dataset | Kaggle](https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset)

**Face mask detector(mask ,not mask, incorrect mask)**

* ***Tác giả***: ***Spandan Patnaik.***
* ***Chi tiết***: Tập dữ liệu được tạo để đào tạo mô hình phát hiện mặt nạ bằng CV techniques.
* ***Nhãn dán***: correct mask (690), not mask (686), incorrect mask (703).
* ***Link download***: [Face mask detector(mask ,not mask, incorrect mask) | Kaggle](https://www.kaggle.com/spandanpatnaik09/face-mask-detectormask-not-mask-incorrect-mask)

**Face mask detector(mask ,not mask, incorrect mask)**

* ***Tác giả***: ***Vijay Kumar.***
* ***Chi tiết***: Tham khảo từ: [Face Mask Detection ~12K Images Dataset | Kaggle](https://www.kaggle.com/ashishjangra27/face-mask-12k-images-dataset) và [Face Mask Detection | Kaggle](https://www.kaggle.com/andrewmvd/face-mask-detection)
* ***Nhãn dán***: correct mask (2994), not mask (2994), incorrect mask (2994).
* ***Link download***: [Face Mask Detection | Kaggle](https://www.kaggle.com/vijaykumar1799/face-mask-detection)

**Flickr-Faces-HQ Dataset (FFHQ)**

* ***Tác giả: Flickr.***
* ***Repositories***: [NVlabs/ffhq-dataset: Flickr-Faces-HQ Dataset (FFHQ)](https://github.com/NVlabs/ffhq-dataset)
* ***Chi tiết***: Flickr-Faces-HQ (FFHQ) là tập dữ liệu hình ảnh chất lượng cao về khuôn mặt người, ban đầu được tạo ra để làm chuẩn cho các mạng đối thủ chung (GAN)
* ***Nhãn dán:*** not mask (133,783).
* ***Bài viết liên quan***: [[1812.04948] A Style-Based Generator Architecture for Generative Adversarial Networks (arxiv.org)](https://arxiv.org/abs/1812.04948)
* ***Link download***: có trong repositories.

***MaskedFace-Net***

* ***Tác giả***: ***Adnane Cabani*** *và các cộng sự*.
* ***Repositories***: [cabani/MaskedFace-Net: MaskedFace-Net is a dataset of human faces with a correctly and incorrectly worn mask based on the dataset Flickr-Faces-HQ (FFHQ)](https://github.com/cabani/MaskedFace-Net)
* ***Bài báo liên quan***: [MaskedFace-Net – A dataset of correctly/incorrectly masked face images in the context of COVID-19 - ScienceDirect](https://www.sciencedirect.com/science/article/pii/S2352648320300362?via%3Dihub).
* ***Chi tiết***: Với bộ dữ liệu ban đầu là [dataset Flickr-Faces-HQ (FFHQ)](https://github.com/NVlabs/ffhq-dataset). Chia ra bao gồm 2 tập dataset lớn là:

+ 67,049 ảnh với Correctly Masked Face Dataset (CMFD) kích thước 1024×1024

+ 66,734 ảnh với Incorrectly Masked Face Dataset (IMFD) kích thước 1024×1024

* ***Nhãn dán***: correct mask (67,049), incorrect mask (66,734).
* ***Link download***: có trong repositories.

### Xử lý dữ liệu.

Sau khi tải những phần dữ liệu về máy. Ta bắt đầu tiến hành xử lý data:

Ta quy ước các nhãn dán như sau:

* ***correct mask***: Hình mẫu chứa khuôn mặt đeo khẩu trang đúng cách.
* ***incorrect mask***: Hình mẫu chứa khuôn mặt đeo khẩu trang không đúng cách.
* ***not mask***: Hình mẫu chứa khuôn mặt không có khẩu trang.

Gom các mẫu về chung một thư mục:

* Các mẫu data có trong nhiều thư mục khác nhau nên chúng ta phải gom về chung một thư mục. Ta có thể dùng hàm sau:

# define function get all file in subfolder to single folder

# @srcPath: array with src path folder constain sample image.

# @desPath: des path folder constain sample image.

# @pre: naming prefix

def copy\_all\_file(srcPath, desPath, pre='image'):

    num = 1

    max\_pad = 6 # padding with number 0

    for path\_root in srcPath:

        for path, subdirs, files in os.walk(path\_root):

            for name in files:

                ori\_name = name

                new\_name = '{}\_{}.{}'.format(pre, (str(num)).zfill(max\_pad), name[name.index('.')+1:])

                num += 1

                print('File copy: {} => {}'.format(ori\_name, new\_name))

                src = os.path.join(path, name)

                des = os.path.join(desPath, new\_name)

                # Copy file

                copyfile(src, des)

Cắt khung hình có chứa khuôn mặt và khẩu trang có đệm (padding):

* Ta có thể dùng thư viện MTCNN (face detecor) để tìm khung hình chứa khuôn mặt trong ảnh và mấy ra ***bounding box*** chứa khuôn mặt.
* Sau khi có được ***bounding box*** ta sẽ cắt ảnh ra là resize lại về kích thước (800, 600) (thường kích thước ảnh khuôn mặt người có tỉ lệ 3x4) với phần đệm (padding) bên ngoài màu đen.
* Ta có thể thực hiện với phần code sau:

import cv2

import numpy as np

from PIL import Image

import tensorflow as tf

from mtcnn.mtcnn import MTCNN

# get an instance of MTCNN

detector = MTCNN()

# crop image

def crop\_image(image, x, y, w, h):

    return image[y:(y+h),x:(x+w),:]

# get image and bounding box face

def detect\_face(image\_path):

    try:

        image = cv2.cvtColor(cv2.imread(image\_path), cv2.COLOR\_BGR2RGB)

        box = detector.detect\_faces(image)[0]

        box = [x if x >= 0 else 0 for x in box['box']]

        return image, box

    except:

        return None, None

# convert tenfsor data to image array

def tensor\_to\_image(tensor):

    tensor = tensor\*255

    tensor = np.array(tensor, dtype=np.uint8)

    if np.ndim(tensor)>3:

        assert tensor.shape[0] == 1

        tensor = tensor[0]

    return Image.fromarray(tensor)

# resize image array and padding

def resize\_and\_pad(image\_array, target\_size=[800, 600]):

    image = tf.image.resize\_with\_pad(image\_array, target\_height=target\_size[0], target\_width=target\_size[1])

    image = tensor\_to\_image(image)

    return image

# detect face, crop image with face and padding

def crop\_face\_and\_pad(image\_path, target\_size=[800, 600]):

    image, box = detect\_face(image\_path) # get bbox

    if box:

        image = crop\_image(image, \*box) # crop with bbox

        image = resize\_and\_pad(image / 255, target\_size)

        return image

    return None

* Mẫu thử như sau:

image = crop\_face\_and\_pad('./datasets/original/correct\_mask/00012\_Mask.jpg')

import matplotlib.pyplot as plt

plt.figure(figsize=(5, 5))

plt.axis('off')

plt.imshow(image)

plt.show()

|  |  |
| --- | --- |
|  |  |

Sau khi cắt và đệm xong gần như ta đã có được dữ liệu như mong muốn (dataset version 1). Và chúng hiện diện trong thư mục **“./datasets/processed/”**. Thì trong bộ data set này chúng tôi đã sử dụng chỉ 4500 đến dưới 5000 mẫu cho mỗi lớp. Số lượng cụ thể như sau:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Correct mask | Incorrect mask | Not mask |
| Số lượng | 4,619 | 4,735 | 4,996 |

Một số thông tin khi xử lý dữ liệu cắt khung ảnh chứ khuôn mặt như sau:

* Trên nhãn ***Correct mask.***



* Trên nhãn ***Incorrect mask.***



* Trên nhãn ***Not mask.***



Sau khi thực nghiệm huấn luyện trên bộ dataset version 1 thì kết quả khá tốt nhưng vẫn còn nhiễu sai dữ liệu:

|  |  |
| --- | --- |
|  |  |
| **Correct mask** (nên là **Incorrect mask**) | **Correct mask** (nên là **Incorrect mask**) |

|  |  |
| --- | --- |
|  |  |
| **Incorrect mask** (nên là **Not mask**) | **Incorrect mask** (nên là **Correct mask**) |

* Khá nhiều mẫu ở 2 nhãn ***Incorrect mask*** và ***Correct mask*** trùng khá nhiều. Nên chúng tôi quyết định làm sạch dữ liệu 1 lần nữa bằng cách thủ công quan sát và lọc bỏ nhưng mẫu chưa chính xác nhãn sán.

Và sau khi thực hiện lọc bỏ dữ liệu và thêm một ít dữ liệu nữa vào lần 2 ta được bộ dataset version 2. Với thông tin sau:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Correct mask | Incorrect mask | Not mask |
| Số lượng | 4,542 | 4,700 | 4,996 |

### Tổng quan mô tả dữ liệu.

Tổng số mẫu trong tập dữ liệu là: ***14,664 ảnh***.

Được chia thành 3 nhãn: ***Correct mask***, ***Incorrect mask*** và ***Not mask.***

Kích thước mỗi mẫu ảnh là: ***800x600*** (Tương quan với khuôn mặt người tỉ lệ ảnh 3x4, mỗi mẫu ảnh có đệm (padding) để không phá vỡ tỉ lệ khuôn mặt khi bị kéo giãn (scale)).

Bộ dữ liệu chủ yếu được dùng chủ yếu ở các tập dataset: ***Flickr-Faces-HQ Dataset (FFHQ) và MaskedFace-Net (CMFD & IMFD)***

## Quy trình huấn luyện và Ứng dụng.

Ta sẽ có các quy trình sau:

### Xử lý dữ liệu và tăng cường dữ liệu:

Sau khi đã tiền xử lý dữ liệu trước đó ta có thể tải dữ liệu lên trang [Kaggle](https://www.kaggle.com/) và tiếp tục các quy trình tiếp theo:

Đọc dữ liệu với các thông tin và nhãn dán.

import pandas as pd

LABELS = []

IMAGES = []

for subfolder in list\_subfolder:

    sunfolder\_path = os.path.join(TRAIN\_DIR, subfolder)

    print(sunfolder\_path)

    for file\_name in os.listdir(sunfolder\_path):

        file\_path = os.path.join(subfolder, file\_name)

        IMAGES.append(file\_path)

        LABELS.append(subfolder)

df\_train = pd.DataFrame({

    'image': IMAGES,

    'label': LABELS

})

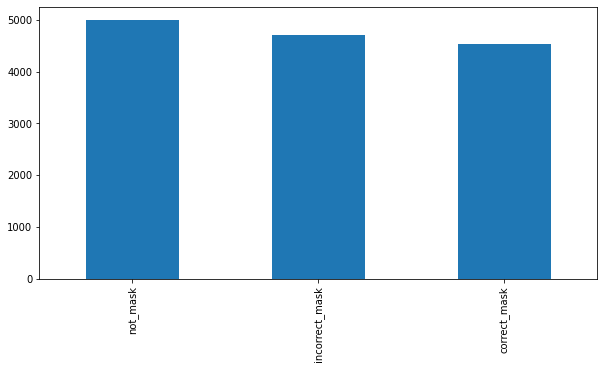


Figure 3 Số lượng mẫu cho mỗi nhãn.

Sau khi đọc đường dẫn ảnh và nhãn xong ta sẽ chia dữ liệu cho tập train và val:

from sklearn.model\_selection import train\_test\_split

tf.random.set\_seed(42)

np.random.seed(42)

df\_train\_set, df\_val\_set = train\_test\_split(df\_train, test\_size=0.2, random\_state=25)

df\_train\_set.to\_csv('df\_train\_set.csv')

df\_val\_set.to\_csv('df\_val\_set.csv')

Kết quả:

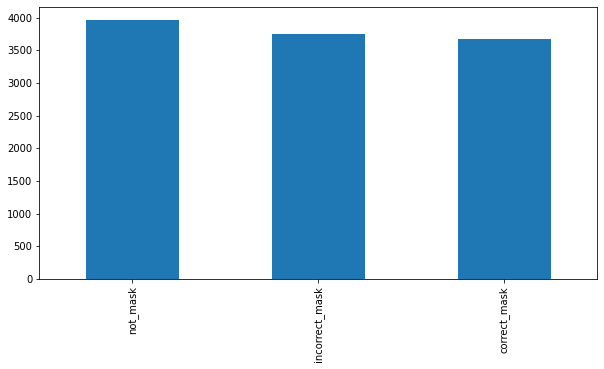


Figure 4 Số lượng mẫu và nhãn cho tập traning.

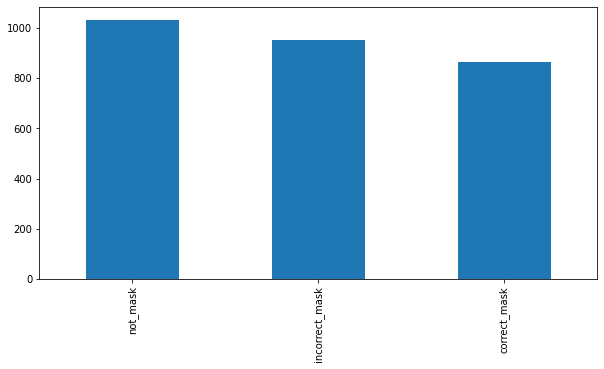


Figure 5 Số lượng mẫu và nhãn cho tập validation.

Tăng cường dữ liệu với các API cung cấp bởi Keras. Với dịch chuyển lên xuống, lật ngang lật dọc,…của tập train. Và trên tập validation ta chỉ cần scale chia cho 255. Thực hiện ở đoạn code dưới:

from keras.preprocessing.image import ImageDataGenerator,load\_img ,img\_to\_array

# for trainning

train\_datagen = ImageDataGenerator(rescale=1./255,

                                   rotation\_range=5,

                                   width\_shift\_range=0.05,

                                   height\_shift\_range=0.05,

                                   horizontal\_flip=True,

                                   samplewise\_center=True,

                                   samplewise\_std\_normalization=True,

                                   fill\_mode='nearest')

train\_set = train\_datagen.flow\_from\_dataframe(dataframe=df\_train\_set,

                                              directory=TRAIN\_DIR,

                                              x\_col="image",

                                              y\_col="label",

                                              shuffle=True,

                                              class\_mode="sparse",

                                              batch\_size=BATCH\_SIZE,

                                              target\_size=TARGET\_SIZE)

# for validation

val\_datagen = ImageDataGenerator(rescale=1./255)

val\_set = val\_datagen.flow\_from\_dataframe(dataframe=df\_val\_set,

                                          directory=TRAIN\_DIR,

                                          x\_col="image",

                                          y\_col="label",

                                          class\_mode="sparse",

                                          batch\_size=BATCH\_SIZE,

                                          target\_size=TARGET\_SIZE)

Example:



Figure 6 Mẫu thử sau khi sử dụng tăng cường dữ liệu.

### Xây dựng mô hình phân loại và Huấn luyện:

Chúng tôi đã sử dụng phương pháp tranfer learning trên(MobileNet V2 và Xception). Với kích thước ban đầu là (400, 300) (Kích thước này tương tự như lúc đã cắt và đệm ở bước xử lý ảnh) nhưng thực sự đã không mang lại hiệu quả bằng kích thước (224, 224). Model được định nghĩa như sau:

# build model

def define\_model():

    base\_model = MobileNetV2(weights="imagenet",

                             include\_top=False,

                             input\_tensor=Input(shape=(224, 224, 3)))#(400, 300, 3)))

    # base\_model = Xception(weights="imagenet",

    #                          include\_top=False,

    #                          input\_shape=(224, 224, 3))

    for layer in base\_model.layers:

        layer.trainable = False

    model = Sequential([

        base\_model,

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

        Flatten(),

        Dense(units=512, activation='relu'),

        Dropout(0.2),

        Dense(units=128, activation='relu'),

        Dropout(0.5),

        Dense(3, activation='softmax')

    ])

    return model

Tiếp theo chúng tôi sẽ đi tìm learning rate phù hợp để huấn luyện model. Chúng tôi đã tham khảo code thầy đã cung cấp với notebook [handson-ml2/11\_training\_deep\_neural\_networks.ipynb at master · ageron/handson-ml2](https://github.com/ageron/handson-ml2/blob/master/11_training_deep_neural_networks.ipynb). Sau khi tìm được điểm thấp nhất (chỉ số loss tốt nhất có thể - Good Loss - GL) tôi sẽ tìm điểm gần nhất có thể có chỉ số loss tệ nhất (chóp nhọn). Và chúng tôi sẽ chọn learning rate tại đỉnh chóp (BLL – Bad Loss Lest) đó để thực hiện quá trình huấn luyện. Ta thực hiện như sau:

class ExponentialLearningRate(keras.callbacks.Callback):

    def \_\_init\_\_(self, factor):

        self.factor = factor

        self.rates = []

        self.losses = []

    def on\_epoch\_begin(self, epoch, logs=None):

        self.prev\_loss = 0

    def on\_batch\_end(self, batch, logs=None):

        batch\_loss = logs["loss"] \* (batch + 1) - self.prev\_loss \* batch

        self.prev\_loss = logs["loss"]

        self.rates.append(K.get\_value(self.model.optimizer.lr))

        self.losses.append(batch\_loss)

        K.set\_value(self.model.optimizer.lr, self.model.optimizer.lr \* self.factor)

def find\_learning\_rate(model, X, y, epochs=1, batch\_size=32, min\_rate=10\*\*-4, max\_rate=0.2):

    model.compile(loss="sparse\_categorical\_crossentropy", optimizer='nadam', metrics=["accuracy"])

    init\_weights = model.get\_weights()

    iterations = math.ceil(len(X)) \* epochs

    factor = np.exp(np.log(max\_rate / min\_rate) / iterations)

    init\_lr = K.get\_value(model.optimizer.lr)

    K.set\_value(model.optimizer.lr, min\_rate)

    exp\_lr = ExponentialLearningRate(factor)

    history = model.fit(X, validation\_data=y, epochs=epochs, batch\_size=batch\_size, callbacks=[exp\_lr])

    K.set\_value(model.optimizer.lr, init\_lr)

    model.set\_weights(init\_weights)

    return exp\_lr.rates, exp\_lr.losses

def plot\_lr\_vs\_loss(rates, losses):

    print("{} => {}".format("GL", "Good Loss"))

    print("{} => {}".format("BLL", "Bad Loss Lastest"))

    plt.figure(figsize=(10,5))

    plt.plot(rates, losses)

    plt.gca().set\_xscale('log')

    plt.hlines(min(losses), min(rates), max(rates))

    plt.axis([min(rates), max(rates), min(losses), (losses[0] + min(losses))])

    good\_loss = [rates[losses.index(min(losses))], min(losses)]

    plt.plot(good\_loss[0], good\_loss[1], 'go')

    plt.text(good\_loss[0], good\_loss[1]+10\*\*-2,

             "GL: (lr = {:f})".format(good\_loss[0]), fontsize=12)

    bad\_index\_lastest = losses.index(min(losses))

    for index in range(losses.index(min(losses))+1, losses.index(min(losses))+20):

        if (losses[index] > losses[bad\_index\_lastest]) and (losses[index] < 1.6):

            bad\_index\_lastest = index

        else: break

    bad\_loss\_lastest = [rates[bad\_index\_lastest], losses[bad\_index\_lastest]]

    plt.plot(bad\_loss\_lastest[0], bad\_loss\_lastest[1], 'ro')

    plt.text(bad\_loss\_lastest[0], bad\_loss\_lastest[1]+10\*\*-2,

             "BLL: (lr = {:f})".format(bad\_loss\_lastest[0]), fontsize=12)

    good\_lr = bad\_loss\_lastest[0]

    plt.xlabel("Learning rate")

    plt.ylabel("Loss")

    return good\_lr

Kết quả như sau:

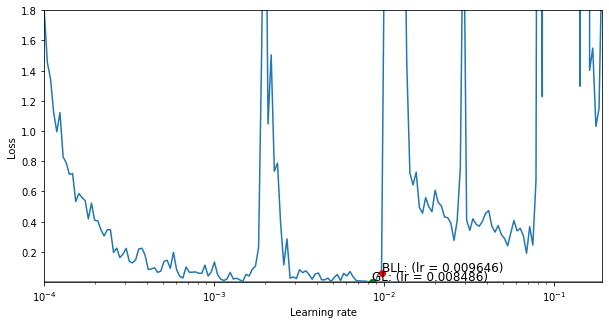


Figure 7 Kết quả tìm kết learning rate với base model MobileNet V2.

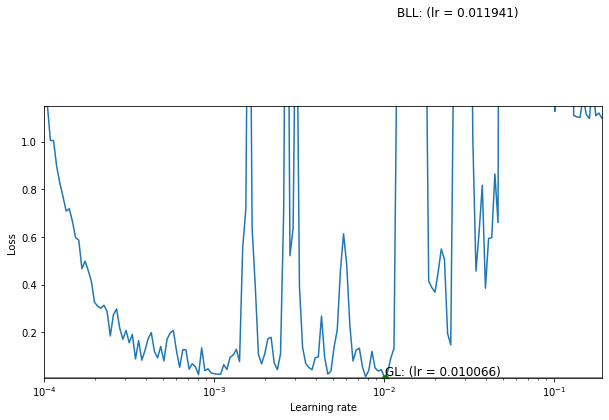


Figure 8 Kết quả tìm kết learning rate với base model Xception.

Sau, ta bắt đầu huấn luyện với learning rate đã tìm ra với các callbacks ***ModelCheckpoint***, ***EarlyStopping***:

from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.optimizers import Nadam

tf.random.set\_seed(42)

np.random.seed(42)

model = load\_model(BASEMODEL\_PATH)

model.compile(loss="sparse\_categorical\_crossentropy",

              optimizer=Nadam(learning\_rate=good\_lr),

              metrics=["accuracy"])

print(model.optimizer.lr)

modelCheckpoint = ModelCheckpoint(

    BESTMODEL\_PATH,

    monitor="val\_loss",

    save\_best\_only=True,

)

earlyStopping = EarlyStopping(

    monitor="val\_accuracy",

    patience=3,

    restore\_best\_weights=True,

)

history = model.fit(train\_set,

                    validation\_data=val\_set,

                    epochs=EPOCHS,

                    callbacks=[modelCheckpoint, earlyStopping])

### Đánh giá mô hình:

Ta có thể xem được thông số loss và accuracy trong quá trình huấn luyện như sau:

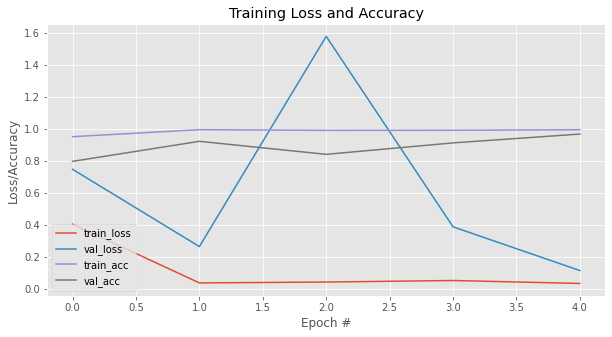


Figure 9 Thông số loss và accuracy huấn luyện trên MobileNet V2.

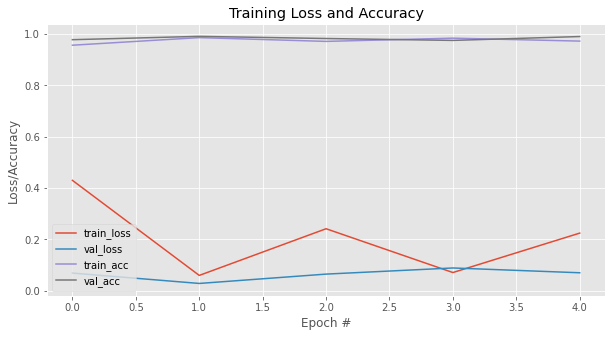


Figure 10 Thông số loss và accuracy huấn luyện trên Xception.

Classification Report, thực hiện như sau:

from keras.preprocessing.image import load\_img, img\_to\_array

from keras.applications.mobilenet\_v2 import preprocess\_input

y\_pred = []

for path in df\_val\_set.image:

    image\_path = os.path.join(TRAIN\_DIR, path)

    my\_image = load\_img(image\_path, target\_size=(224, 224))

    #preprocess the image

    my\_image = img\_to\_array(my\_image)

    my\_image = my\_image.reshape((1, my\_image.shape[0], my\_image.shape[1], my\_image.shape[2]))

    my\_image = preprocess\_input(my\_image)

    y\_pred.append(np.argmax(model.predict(my\_image)))

from sklearn.metrics import classification\_report, confusion\_matrix

print('Classification Report')

print(classification\_report(val\_set.classes, y\_pred, target\_names=list\_subfolder))

Kết quả:

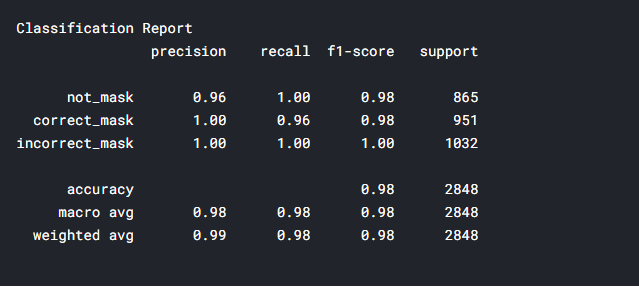


Figure 11 Classification Report của model MobileNet V2.

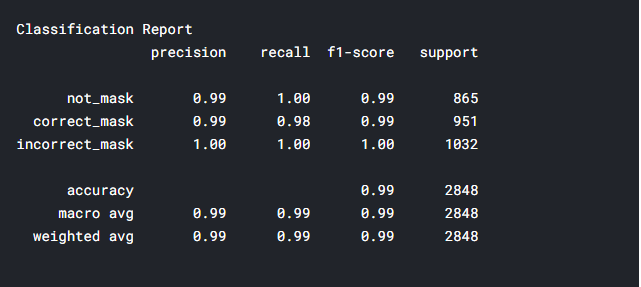


Figure 12 Classification Report của model Xception.

Confusion Matrix, thực hiện như sau:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report,confusion\_matrix

cm = confusion\_matrix(val\_set.classes, y\_pred)

cm = pd.DataFrame(cm , index = list\_subfolder , columns = list\_subfolder)

plt.figure(figsize = (10,10))

sns.heatmap(cm, cmap= "Blues", linecolor = 'black' ,

            linewidth = 1 , annot = True, fmt='' ,

            xticklabels = list\_subfolder , yticklabels = list\_subfolder)

Kết quả:

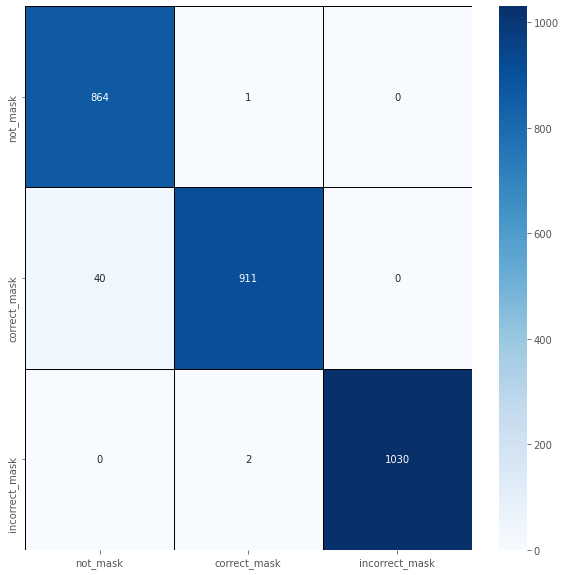


Figure 13 Confusion Matrix của model MobileNet V2.

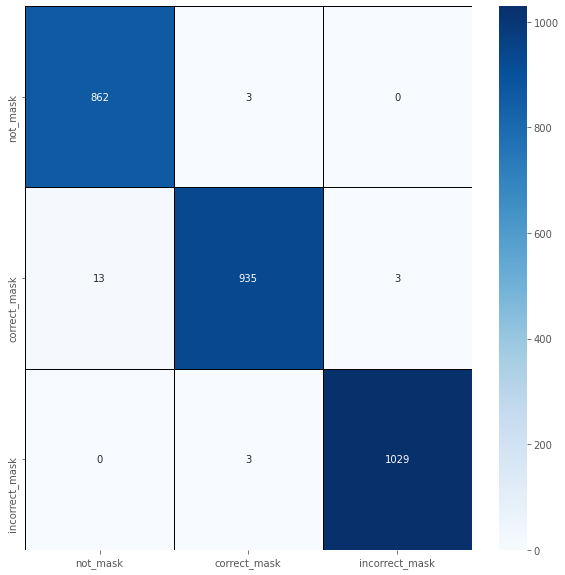


Figure 14 Confusion Matrix của model Xception.

## Ứng dụng thực tế.

Nhằm tiết kiệm công sức cũng như tăng độ hiệu quả của mô hình nên chúng tôi đã sử dụng công cụ hỗ trợ nhận diện khung mặt đó là thư viện DNN của OpenCV. Tham khảo ở bài viết: [Face detection with OpenCV and deep learning - PyImageSearch](https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/).

### Thử nghiệm trên hình ảnh tĩnh với model Xception:

Với Not Mask:

|  |  |
| --- | --- |
|  | True Label: **Not Mask** |
| Predict Label: **Not Mask** |

Với Correct Mask:

|  |  |
| --- | --- |
|  | True Label: **Correct Mask** |
| Predict Label: **Correct Mask** |

Với Incorrect Mask:

|  |  |
| --- | --- |
|  | True Label: **Incorrect Mask** |
| Predict Label: **Incorrect Mask** |

### Thử nghiệm trên video camera với model Xception:

Với Not Mask:

|  |  |
| --- | --- |
|  | True Label: **Not Mask** |
| Predict Label: **Not Mask** |

Với Correct Mask:

|  |  |
| --- | --- |
|  | True Label: **Correct Mask** |
| Predict Label: **Correct Mask** |

Với Incorrect Mask:

|  |  |
| --- | --- |
|  | True Label: **Incorrect Mask** |
| Predict Label: **Incorrect Mask** |