

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH



ĐỒ ÁN NHẬN DẠNG

Age Prediction

Thành viên: Phạm Ngọc Trường	: 18521571
Nguyễn Tuấn Quang	: 18521302
Nguyễn Kiều Vinh	: 18521653
Nguyễn Văn Thịnh	: 18521448
Nguyễn Minh Quang	: 18521299

Lớp: CS338.L22.KHCL

GVHD: Th.S Đỗ Văn Tiến

Mục lục

I. Bảng phân công	4
II. GIỚI THIỆU VỀ ĐỀ TÀI.....	5
1. Lý do chọn đề tài:	5
2. Tổng quan về tình hình nghiên cứu hiện nay	6
3. Đối tượng nghiên cứu của đề tài:	7
4. Các thách thức:.....	9
5. Các bước triển khai:.....	11
III. Cơ sở lý thuyết.....	12
1. Phân loại bài toán trong machine learning	12
2. Một số vấn đề trong tiền xử lý dữ liệu	13
2.1. Vai trò của data.....	13
2.2. Một số kĩ thuật giúp tăng data.....	14
2.3. Data Augmentation	15
2.4. Imbalance dataset	17
3. Một số vấn đề trong quá trình training model	18
3.1. Transfer learning	18
3.2. Một số hàm activation	21
3.4. Một số layer thường dùng	27
4. Tìm hiểu một số kiến trúc mạng	29
4.1. Tổng quan.....	29
4.2. MobileNet V1.....	30
4.3. MobileNet V2.....	30
4.4. MobileNet V3.....	31
4.5. So sánh MobileNet V1 V2 V3	32
IV. Xây dựng model.....	33
1. Tổng quan về cấu trúc model:.....	33
2. Dataset preprocessing	34
2.1. Khởi tạo biến toàn cục.....	34
2.2. Đọc dữ liệu với tensorflow.....	34
2.3. Tách dữ liệu thành train/test/val	34



2.4. Tăng tốc độ huấn luyện với AUTOTUNE	35
2.5. Xây dựng layer Data augment.....	35
2.6. Tạo CallBack.....	36
3. Build model	37
3.1. Download Model từ tensorflow	37
3.2. Xây dựng fully connected layers	38
3.3. Transfer learning	40
4. Đánh giá các model.....	42
4.1. Fully-Connected Layers có sử dụng Data Augment.....	42
4.2. Fully-Connected Layers không sử dụng Data Augment.....	44
4.3. Thử nghiệm model	45
V. Frontend React.....	46
1. Tổng quan	46
2. Axios.....	46
3. Triển khai thực tế	47
VI. Backend Flask.....	49
1. Tổng quan	49
2. Flask cors.....	49
3. SSL Flask.....	50
3.1. Certificate	50
3.2. Decrypts	51
4. Triển khai thực tế	52
5. Server Requirements.....	53
VIII. Phương hướng phát triển	55
IX. Reference.....	56
Webapp:.....	56
Github:	56



I. Bảng phân công

Thông tin thành viên		Công việc
Tên	MSSV	
Phạm Ngọc Trường (Leader)	18521571	<ul style="list-style-type: none">• Xây dựng kiến trúc hệ thống• Lựa chọn công nghệ• Build và training model• API Flask server
Nguyễn Tuấn Quang	18521302	<ul style="list-style-type: none">• Losses, Metrics• Transfer learning• Data Augmentation• Batch normalization• Dropout, GlobalAveragePooling2D
Nguyễn Kiều Vinh	18521653	<ul style="list-style-type: none">• React• Connect Flask với React• Deploys project
Nguyễn Văn Thịnh	18521448	<ul style="list-style-type: none">• Flask• Imbalance Dataset
Nguyễn Minh Quang	18521299	<ul style="list-style-type: none">• MobileNet V1, V2, V3• VGG 16, 19• GoogleNet, ResNets, Densenet• Activation• Tổng quan Flask



II. GIỚI THIỆU VỀ ĐỀ TÀI

1. Lý do chọn đề tài:

- Lý do khách quan:

- Đối với ngành thương mại thương mại nói chung và thương mại điện tử nói riêng, việc tiếp thị và nhận diện đối tượng khách hàng là một phần không thể thiếu trong quá trình marketing. Trong khi đó, marketing là một quá trình không thể thiếu trong việc đưa sản phẩm, dịch vụ của doanh nghiệp đến tay người tiêu dùng. Nó ảnh hưởng trực tiếp đến tốc độ tăng trưởng, khả năng mở rộng thị trường và lợi nhuận ròng của một doanh nghiệp. Sự thành công của một chiến dịch marketing nằm phần lớn trong việc xác định được đối tượng khách hàng chính của sản phẩm và dịch vụ mà doanh nghiệp đang hướng tới.
- Trong khi việc vận hành một đội ngũ sales để triển khai các chiến dịch marketing trên phạm vi rộng gây ra lãng phí tài nguyên cho doanh nghiệp. Thay vào đó, chúng ta sẽ sử dụng dữ liệu về độ tuổi của khách hàng, xác định nhóm khách hàng tiềm năng để thu nhỏ phạm vi triển khai, giảm thiểu đáng kể thất thoát tài chính không đáng có. *Vì thế, việc dự đoán được độ tuổi của khách hàng và xác định được nhóm khách hàng tiềm năng thông qua ảnh chụp, video từ mạng xã hội, từ các camera an ninh của các trung tâm mua sắm, ... được xem là một yếu tố quan trọng cấu thành một chiến dịch marketing thành công.*

- Lý do chủ quan:

- Áp dụng được một số công nghệ mới trong machine learning vào việc tăng độ chính xác và giảm thời gian huấn luyện mô hình
- Tìm hiểu cách triển khai phần mềm chạy trong thực tế với môi trường phát triển là web, thiết kế theo dạng là client – server, công nghệ sử dụng ở frontend là Reactjs, công nghệ sử dụng ở backend là Flask



2. Tổng quan về tình hình nghiên cứu hiện nay

- Thống kê theo Paper with Code Benchmarks trên tập dữ liệu MORPH (55134 bức ảnh về khuôn mặt của 13617 đối tượng có độ tuổi từ 16 tới 77)

Filter: untagged Edit Leaderboard

Rank	Model	MAE ↓	Extra Training Data	Paper	Code	Result	Year	Tags
1	DLDL-v2 (ThinAgeNet)	1.969	×	Learning Expectation of Label Distribution for Facial Age and Attractiveness Estimation	🔗	📄	2020	
2	DLDL+VGG-Face	2.42±0.01	×	Deep Label Distribution Learning with Label Ambiguity	🔗	📄	2016	
3	DLDL+VGG-Face (KL, Max)	2.42	×	Deep Label Distribution Learning with Label Ambiguity	🔗	📄	2016	
4	MegaAge (w. IMDB-WIKI)	2.52	✓	Quantifying Facial Age by Posterior of Age Comparisons	🔗	📄	2017	
5	CORAL	2.59	×	Rank consistent ordinal regression for neural networks with application to age estimation	🔗	📄	2019	
6	MegaAge	2.87	✓	Quantifying Facial Age by Posterior of Age Comparisons	🔗	📄	2017	













- Mức độ chính xác của mô hình dựa vào chỉ số MAE (mean absolute error – sai số tuyệt đối trung bình), với chỉ số MAE tốt nhất đạt 0.0 (càng thấp càng tốt).
- Các mô hình được huấn luyện ngày càng đạt được độ chính xác cao hơn.
- Ví dụ với mô hình DLDL-v2 được triển khai năm 2020 đạt MAE ở mức 1.969 so với 2.87 của MegaAge được triển khai năm 2017. Điểm chung của các mô hình trên là đều xây dựng trên kiến trúc deep learning.
- Các nguyên nhân ảnh hưởng đến sự thành công của deep learning trong thời gian gần đây:
 - Sự nâng cấp của sức mạnh về khả năng tính toán của GPU.
 - Sự cải tiến của một số kiến trúc mạng nền tảng như GoogleNet, VGG, ResNet,...
 - Sự ra đời và cải tiến của một số kỹ thuật: transfer learning, fine tuning, freeze layer, dropout, batch normalization, data augmentation, ...



- Nhiều framework hỗ trợ deeplearning với GPU như: theano, caffe, mxnet, tensorflow, pytorch, keras,...
- Nhiều kỹ thuật tối ưu quá trình huấn luyện như: Adagrad, RMSProp, Adam,...

3. Đối tượng nghiên cứu của đề tài:

- Bao gồm 9778 bức ảnh chụp gương mặt đã qua quá trình face detect được gắn nhãn, có độ tuổi trải dài từ 1 đến 110 tuổi
- Dataset được sử dụng trong quá trình nghiên cứu: [facial age | Kaggle](#)
- Cấu trúc tổ chức dữ liệu: các dữ liệu cùng một nhãn được lưu trong cùng 1 folder với tên folder chính là nhãn của tập dữ liệu đó

Tên ↑	Chủ sở hữu	Sửa đổi lần cuối	Kích cỡ tệp
 001	tôi	5 thg 7, 2021 tôi	—
 002	tôi	5 thg 7, 2021 tôi	—
 003	tôi	5 thg 7, 2021 tôi	—
 004	tôi	5 thg 7, 2021 tôi	—
 005	tôi	5 thg 7, 2021 tôi	—
 006	tôi	5 thg 7, 2021 tôi	—
 007	tôi	5 thg 7, 2021 tôi	—
 008	tôi	5 thg 7, 2021 tôi	—
 009	tôi	5 thg 7, 2021 tôi	—
 010	tôi	5 thg 7, 2021 tôi	—
 011	tôi	5 thg 7, 2021 tôi	—
 012	tôi	5 thg 7, 2021 tôi	—

- Kết quả trích xuất dữ liệu từ python:

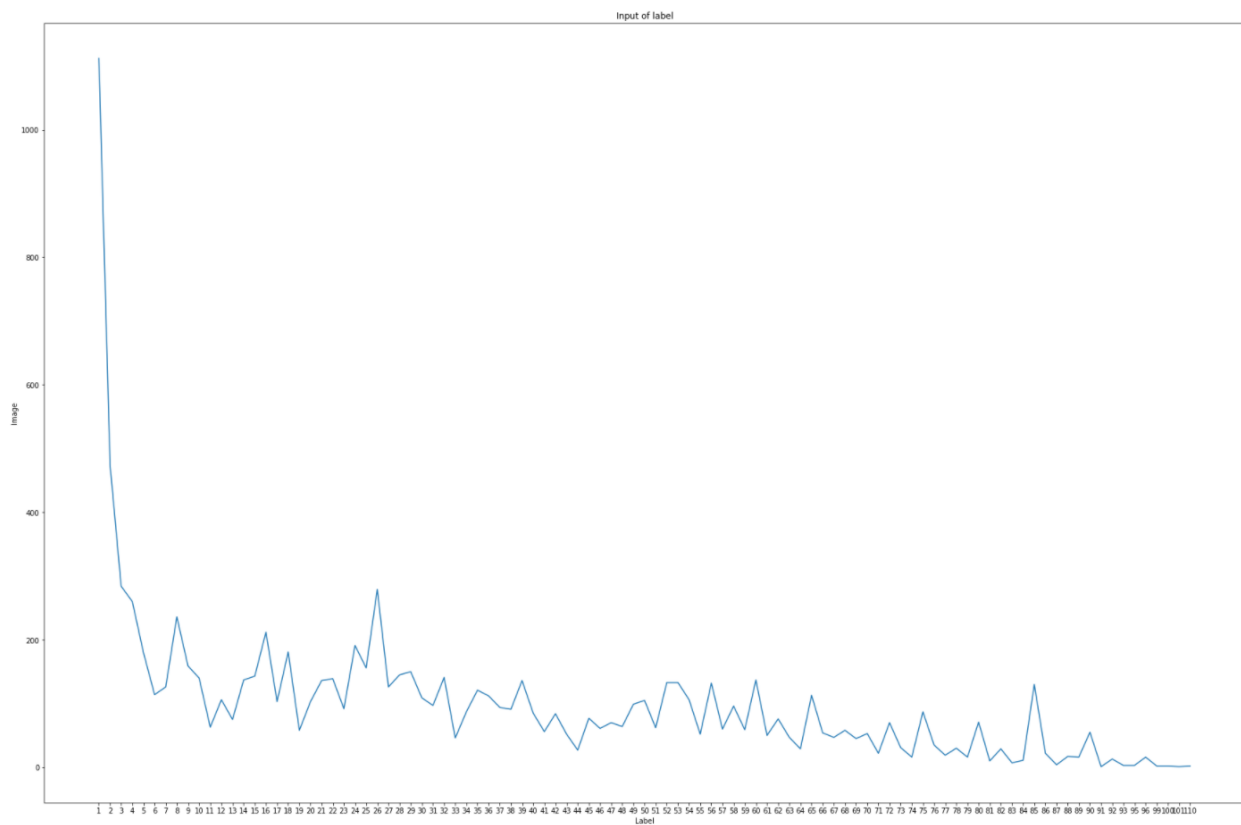
Found 9778 files belonging to 99 classes.

- Ví dụ về các nhãn:

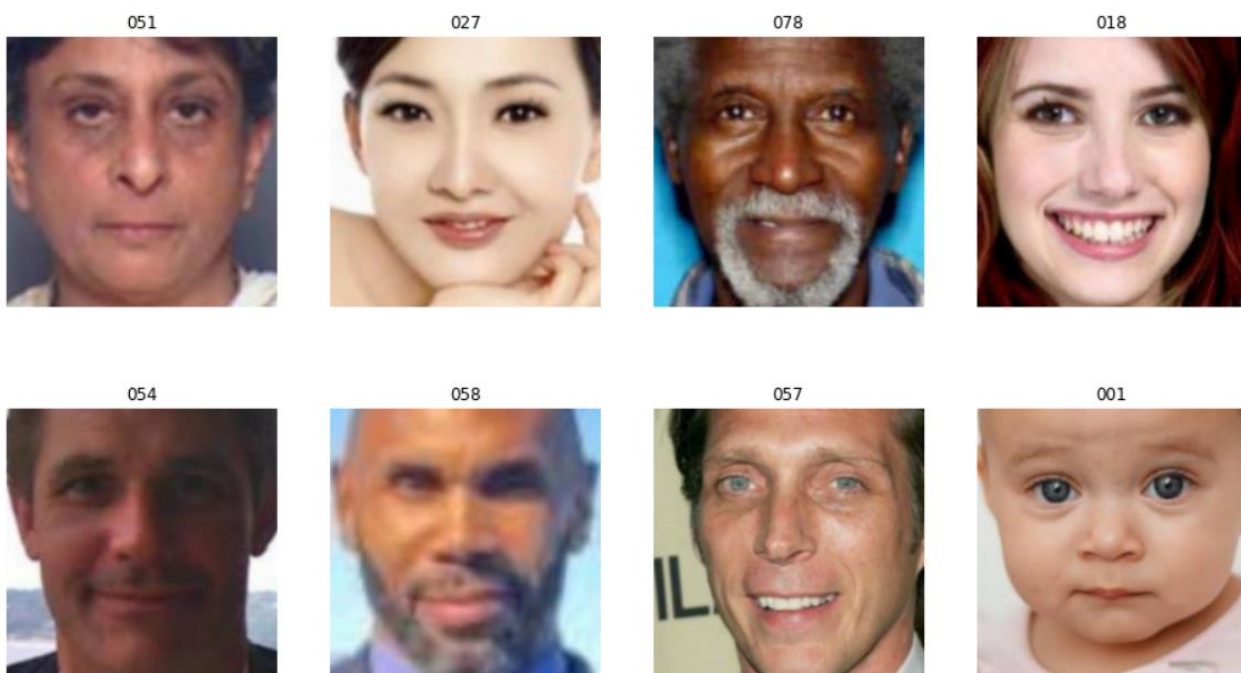
```
99 : ['001', '002', '003', '004', '005', '006', '007', '008', '009', '010',
```



- Phân bố dữ liệu theo lớp

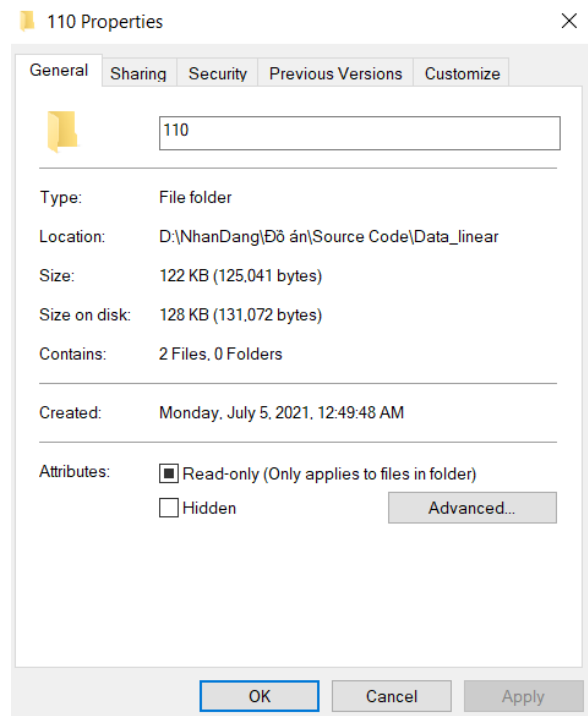
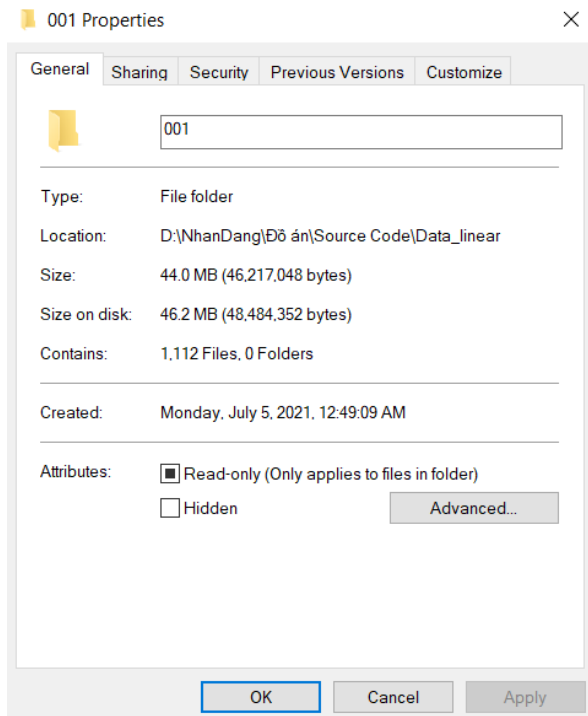


- Dưới đây là một số hình ảnh được trích xuất từ dữ liệu có kèm theo nhãn



4. Các thách thức:

- Đối với dữ liệu huấn luyện
 - Lượng dữ liệu để huấn luyện còn nhỏ trong khi số lượng class lớn, 9978 images trong khi có tới 99 class, trung bình 101 image / class.
 - Dữ liệu quá nhỏ nên chưa đạt được độ phủ cao.
 - Có sự chênh lệch giữa số lượng input của class: class 001 có 1112 image trong khi class 110 có 2 image. Dẫn đến xảy ra tình trạng imbalanced dataset (mất cân bằng dữ liệu)



- Đối với quá trình thử nghiệm trên model huấn luyện:
 - Dữ liệu input đầu có độ phân giải cao gây chậm quá trình tính toán nhất là ở quá trình face detection
 - Trong quá trình thử nghiệm trong thực tế model thường cho độ chính xác không cao, nguyên nhân gồm có:
 - Nguyên nhân khách quan: ảnh hưởng từ môi trường sống, ăn uống, giờ giấc sinh hoạt, di truyền từ bố mẹ gây tác động lên tuổi tác của một người.
 - Nguyên nhân chủ quan: hình ảnh dùng để thử nghiệm lấy từ các mạng xã hội như facebook, zalo, instagram,... Thông thường người dùng trước khi đăng ảnh đã xử dụng các biện pháp làm đẹp như trang điểm, các ứng dụng filter làm đẹp,... Ngoài ra, đối với một số trường hợp là người nổi tiếng, việc phẫu thuật thẩm mỹ cũng làm sai khác đi độ tuổi thực tế của họ.
 - Ví dụ về các tác nhân khách quan và chủ quan:



Hình 1. Mặt bị lão hóa và không bị lão hóa



Hình 2. Mặt không trang điểm và trang điểm

- Đối với quá trình triển khai thực tế
 - Vì triển khai theo dạng client – server nên kết nối mạng là vấn đề tối quan trọng.
 - Backend sử dụng model deep learning nên yêu cầu server cần có hỗ trợ GPU

5. Các bước triển khai:

1. Xây dựng model dữ liệu đoán độ tuổi
2. Xây dựng frontend với React
3. Xây dựng backend với Flask API
4. Deploys React trên heroku
5. Deploys Flask trên colabs
6. Connect React với Flask thông qua url của ngrok



III. Cơ sở lý thuyết

1. Phân loại bài toán trong machine learning

- Classification:

- Là một quá trình tìm kiếm một hàm giúp chia tập dữ liệu thành các lớp dựa trên các tham số khác nhau.
- Nhiệm vụ của classification là tìm hàm ánh xạ biến đầu vào (x) với biến đầu ra rời rạc (y)
- Ví dụ: phân loại chó mèo, phân loại email spam, phân loại giới tính,...
- Một số thuật toán classification:
 - Logistic Regression
 - K-Nearest Neighbours
 - Support Vector Machines
 - Kernel SVM
 - Naïve Bayes
 - Decision Tree classification
 - Random Forest classification

- Regression:

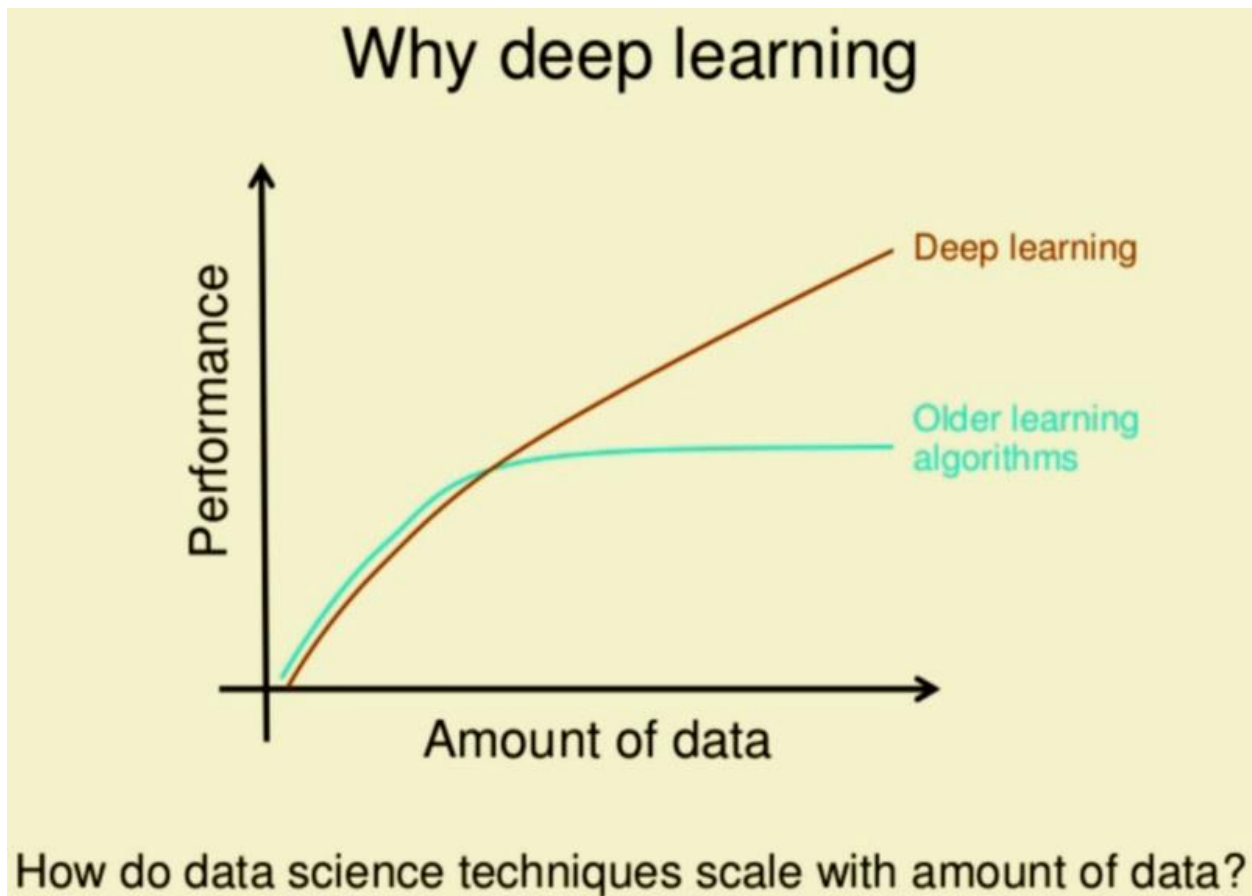
- Là một quá trình tìm kiếm mối tương quan giữa các biến phụ thuộc và biến độc lập.
- Nhiệm vụ của Regression là tìm hàm ánh xạ biến đầu vào (x) với biến đầu ra liên tục (y)
- Ví dụ: dự đoán thời tiết, dự đoán giá cổ phiếu, dự đoán tuổi,...
- Một số thuật toán Regression:
 - Simple Linear Regression
 - Polynomial Regression
 - Support Vector Regression
 - Decision Tree Regression
 - Random Forest Regression



2. Một số vấn đề trong tiền xử lý dữ liệu

2.1. Vai trò của data

- Độ chính xác của một mô hình deep learning phụ thuộc phần lớn vào 2 yếu tố chính:
 - Kiến trúc model
 - Dữ liệu dùng để huấn luyện model
- Deep learning là thuật toán dựa trên việc tìm hiểu mối liên hệ giữa data (data – driven approach), điều này có nghĩa là đối với mô hình huấn luyện có data càng lớn, càng rộng, thì mô hình càng đạt được độ chính xác cao.



2.2. Một số kĩ thuật giúp tăng data

Three ways to improve data

1 - Collect more



- expensive
- requires manual labor

2 - Synthesize



- complicated
- might not truly represent the real data

3 - Augment







- simple
- but finding a good augmentation strategy takes lots of trial & error (**=time of AI engineers**)





- Có 3 kĩ thuật để tăng dữ liệu:
 - Collect more (thu thập thêm dữ liệu): chi phí cao, thực hiện một cách thủ công, tốn công sức và tiền bạc
 - Data synthesis (tạo dữ liệu fake): đối với một số bài toán có thể mô phỏng dữ liệu thông qua computer graphic, dữ liệu có thể không tồn tại trong thực tế
 - Data Augmentation: sử dụng các phép biến đổi tuyến tính hoặc phi tuyến để xử lý dữ liệu, tạo ra “một phiên bản” khác của dữ liệu
- Trong 3 kĩ thuật trên, phương pháp dễ thực hiện nhất là data augmentation.

2.3. Data Augmentation

- Một số kỹ thuật Data Augmentation thường dùng trong xử lý ảnh 2D:
 - *Flip*: lật ảnh theo chiều ngang hoặc chiều dọc
 - *Rotation*: xoay ảnh theo các góc khác nhau
 - *Scale*: thu phóng ảnh
 - *Crop*: cắt một vùng ảnh sau đó resize vùng ảnh đó về kích thước ban đầu

Original	Flip	Rotation	Random crop
			
<ul style="list-style-type: none"> • Image without any modification 	<ul style="list-style-type: none"> • Flipped with respect to an axis for which the meaning of the image is preserved 	<ul style="list-style-type: none"> • Rotation with a slight angle • Simulates incorrect horizon calibration 	<ul style="list-style-type: none"> • Random focus on one part of the image • Several random crops can be done in a row

- *Translation*: dịch chuyển ảnh theo trục x hoặc y
- *Noise addition*: thêm nhiễu vào bức ảnh
- *Information loss*: làm mất mát một phần thông tin của bức ảnh
- *Contrast change*: thay đổi độ tương phản, độ bão hòa của bức ảnh

Color shift	Noise addition	Information loss	Contrast change
			
<ul style="list-style-type: none"> • Nuances of RGB is slightly changed • Captures noise that can occur with light exposure 	<ul style="list-style-type: none"> • Addition of noise • More tolerance to quality variation of inputs 	<ul style="list-style-type: none"> • Parts of image ignored • Mimics potential loss of parts of image 	<ul style="list-style-type: none"> • Luminosity changes • Controls difference in exposition due to time of day

- Geometry based: các phép biến đổi xoay, lật, scale, padding, biến dạng đối tượng
- Color based: các phép biến đổi về độ sáng, độ sắc nét, tương phản hay phép biến đổi sang hình ảnh âm bản
- Noise/ Occlusion: tạo nhiễu, làm mờ, gây mất mát thông tin của hình ảnh
- Weather: thêm các hiệu ứng tác dụng của môi trường lên hình ảnh như mưa, tuyết, sương mờ,...

Base Augmentations

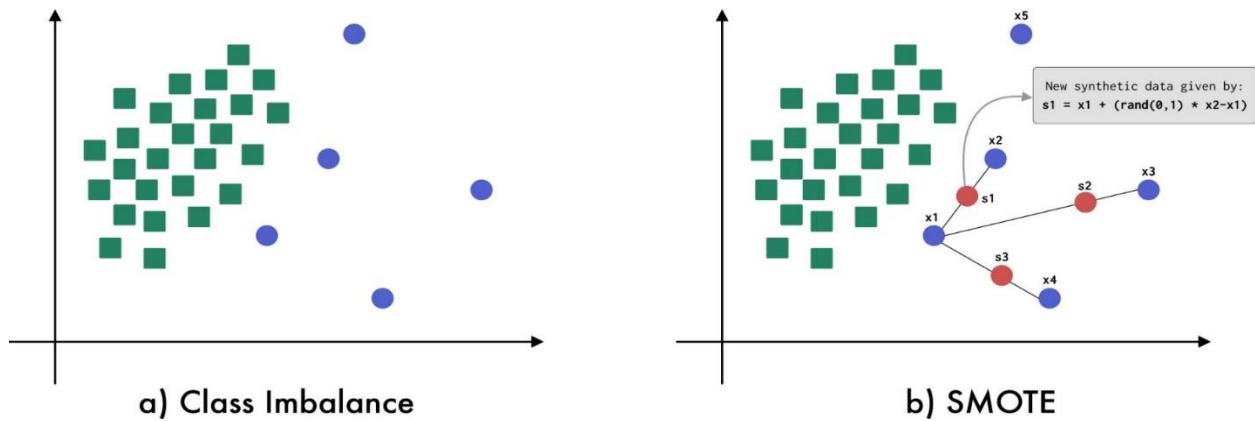


- Tùy vào yêu cầu của bài toán và ứng dụng trong thực tế mà mỗi bộ dữ liệu sẽ có cách thức riêng để augmentation sao cho ra có kết quả tốt nhất.
- Ngoài ra còn có nhiều kỹ thuật Data Augmentation khác được áp dụng cho các dạng dữ liệu ngoài ảnh 2D như âm thanh hoặc video, trong nội dung đề tài này sẽ không được đề cập đến.

2.4. Imbalance dataset

- Imbalance dataset là hiện tượng dữ liệu phân bố không đồng đều trong một tập dữ liệu
- *Ví dụ:* trong tập dữ liệu phát hiện thẻ tín dụng thực hiện giao dịch gian lận, hầu hết các giao dịch đều không phải là gian lận, một số ít trong đó là giao dịch gian lận.
- Điều này dẫn đến mô hình máy học thiên về lớp đa số, gây ảnh hưởng lớn đến độ chính xác của mô hình.
- Các phương pháp giải quyết:
 - *Phương pháp lấy mẫu dưới (Undersampling methods):*
 - Phương pháp này hoạt động với lớp đa số
 - Là loại bỏ ngẫu nhiên các trường hợp của lớp đa số. Nó giảm số lượng quan sát từ lớp đa số để làm cho tập dữ liệu cân bằng.
 - Nhược điểm: mất thông tin nghiêm trọng.
 - *Phương pháp lấy mẫu quá mức (Oversampling methods):*
 - Phương pháp này hoạt động với lớp thiểu số.
 - Là sao chép các trường hợp ngẫu nhiên của lớp thiểu số để cân bằng dữ liệu.
 - Nhược điểm: Dữ liệu trùng lặp quá nhiều.
 - *Phương pháp tạo dữ liệu tổng hợp (Synthetic data generation):*
 - Phương pháp này hoạt động với lớp thiểu số.
 - Là tạo ra dữ liệu nhân tạo để khắc phục sự mất cân bằng dữ liệu.
 - Loại phổ biến nhất là SMOTE: Đối với mỗi một trong các mẫu của lớp thiểu số, k hàng xóm gần nhất của nó được định vị và sau đó giữa các cặp điểm được tạo bởi mẫu và mỗi hàng xóm của nó, một dữ liệu tổng hợp được tạo ra.

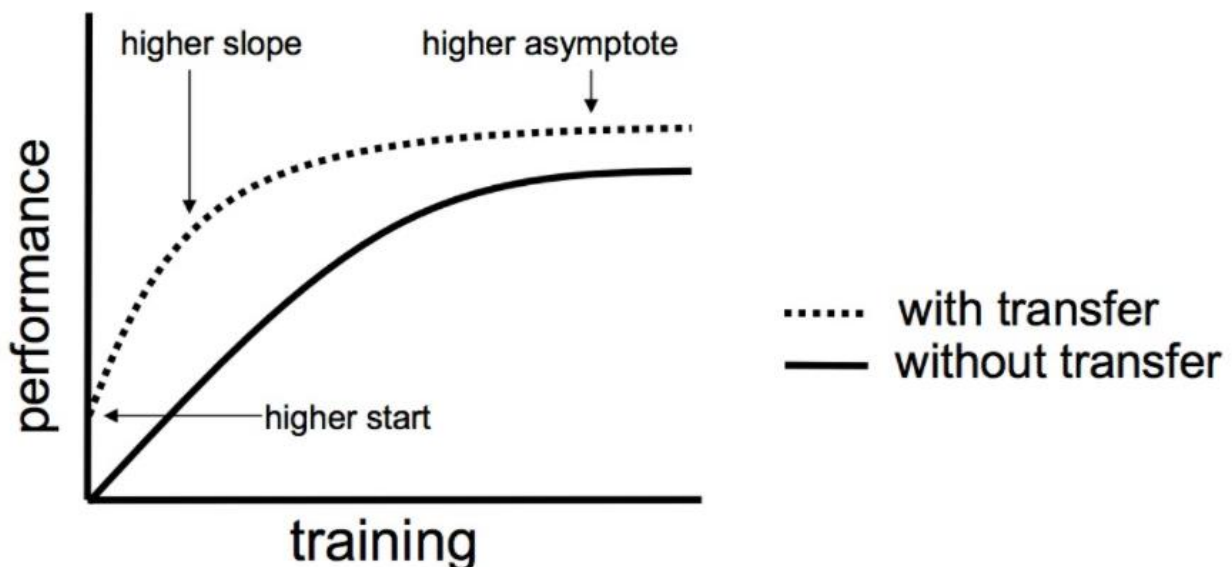




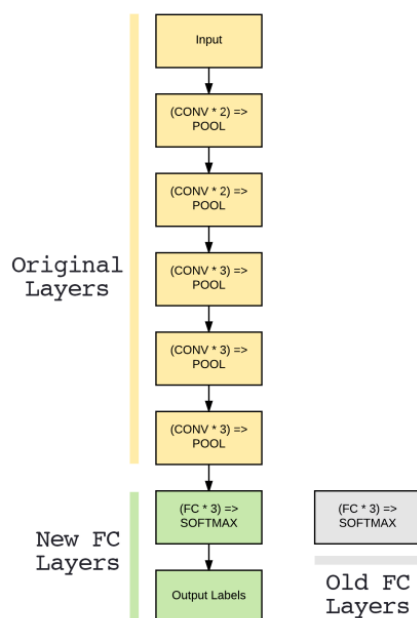
3. Một số vấn đề trong quá trình training model

3.1. Transfer learning

- Nguyên nhân một mô hình không đạt được độ chính xác cao thường do các nguyên nhân sau đây:
 - Dữ liệu nhỏ, không đại diện được
 - Mô hình bị mất cân bằng dữ liệu
 - Kiến trúc mô hình quá phức tạp dẫn đến xảy ra overfitting
- Quá trình tính toán tối ưu hóa gặp khó khăn
- Để giải quyết vấn đề trên trong những năm gần đây người ta thường sử dụng transfer learning.



- Transfer learning là quá trình chuyển giao tri thức đã học được từ một mô hình được đào tạo từ trước sang bài toán hiện tại.
- *Ưu điểm của transfer learning so với huấn luyện lại mô hình từ đầu*: giảm thiểu chi phí tính toán, tăng độ chính xác đáng kể với số epoch nhỏ, tốc độ tính toán nhanh, có điểm khởi đầu tốt hơn.
- *Nhược điểm của transfer learning so với huấn luyện lại mô hình từ đầu*: yêu cầu hiểu rõ các layer của mô hình để tránh hiện tượng sai sót trong quá trình transfer learning, làm cho kết quả của mô hình thấp hơn rất nhiều so với việc huấn luyện từ đầu
- Transfer learning được chia thành 2 loại:
 - *Feature extractor*: rút trích đặc trưng bằng việc sử dụng ConvNet của pre-trained model, sau đó sử dụng các thuật toán machine learning (linear, SVM, softmax classifier,...) để học và dự đoán kết quả từ features đã trích xuất
 - *Fine tuning*: giống với feature extractor ở chỗ rút trích đặc trưng của pre-trained model, tiếp theo đó chúng ta sẽ thêm các layer phù hợp với mục đích của bài toán nhằm mục đích khiến cho 2 model fit với nhau một cách hoàn chỉnh và tạo thành một model mới.

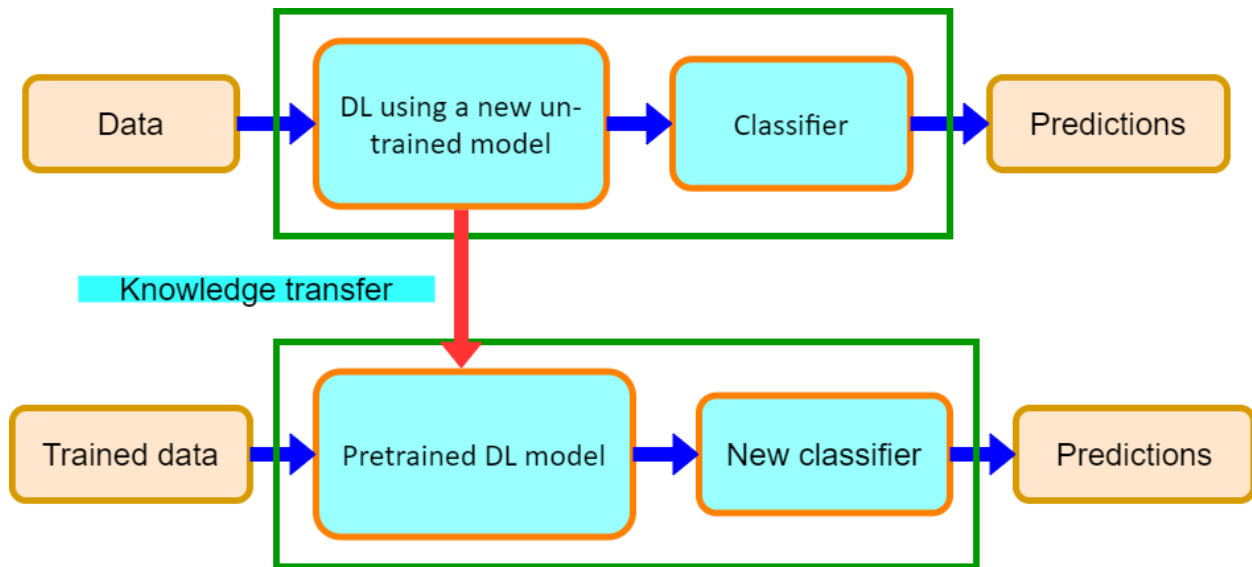


- Freeze layer là quá trình đóng băng layer, có thể hiểu một cách đơn giản là giữ nguyên trọng số đã được đào tạo

➔ Accuracy của fine tuning tốt hơn đáng kể so với feature extractor tuy nhiên thời gian training cũng lâu hơn tùy theo việc freeze layer nhiều hay ít

- Quá trình transfer learning có thể được tóm tắt như sau:





- Tùy vào bài toán hiện tại và mô hình pre-trained mà ta sẽ có các cách transfer learning khác nhau:
 - Đối với dữ liệu nhỏ: train lại toàn bộ các layers sẽ làm mất đi các đặc trưng đã được học từ model pre-trained dẫn tới kết quả dự đoán sẽ không chính xác. Chúng ta chỉ nên train lại các fully connected layers cuối.
 - Đối với dữ liệu lớn và giống domain: có thể train lại model trên toàn bộ layers. Nhưng để quá trình huấn luyện nhanh hơn thì chúng ta sẽ thực hiện bước khởi động (warm up) và sau đó mới fine tuning lại mô hình.
 - Đối với dữ liệu lớn và khác domain: chúng ta nên train lại trên toàn bộ layers vì pre-trained model không trích xuất được các đặc trưng tốt cho dữ liệu khác domain
- Tổng kết lại, chúng ta nên sử dụng transfer learning trong những trường hợp sau:
 - 2 mô hình có cùng domain
 - Dữ liệu huấn luyện mô hình pre-trained lớn hơn đáng kể mô hình được transfer
 - Pre-trained model phải là mô hình có phẩm chất tốt

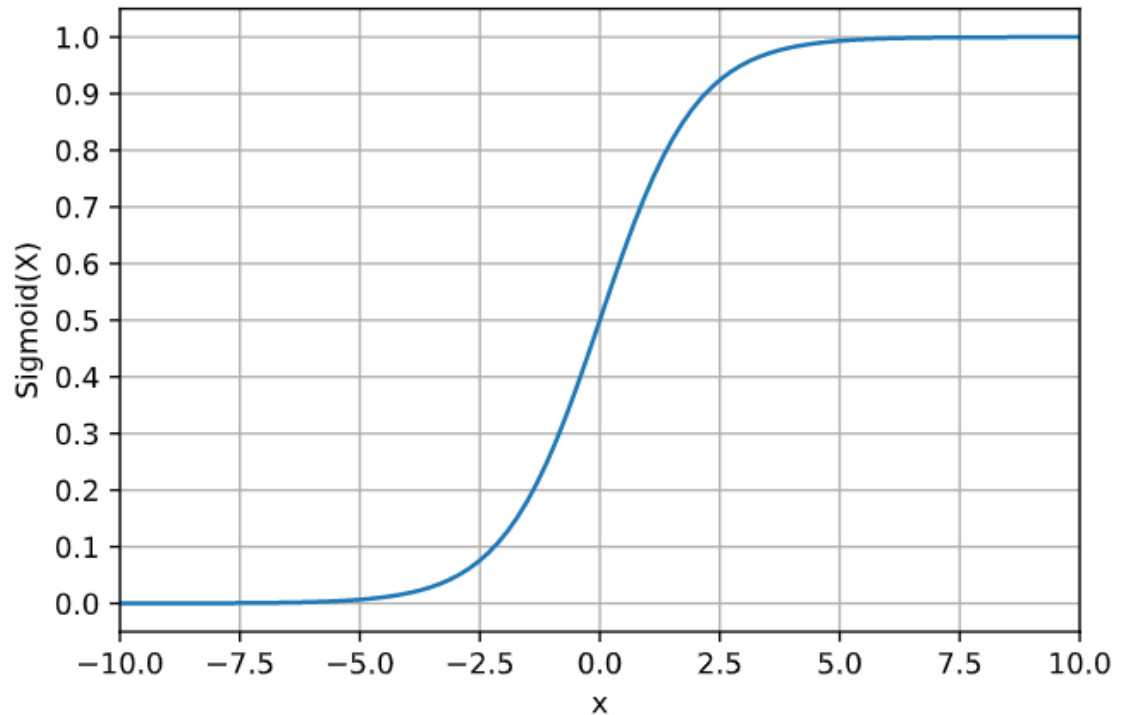
3.2. Một số hàm activation

3.2.1. Sigmoid

- Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Đồ thị:

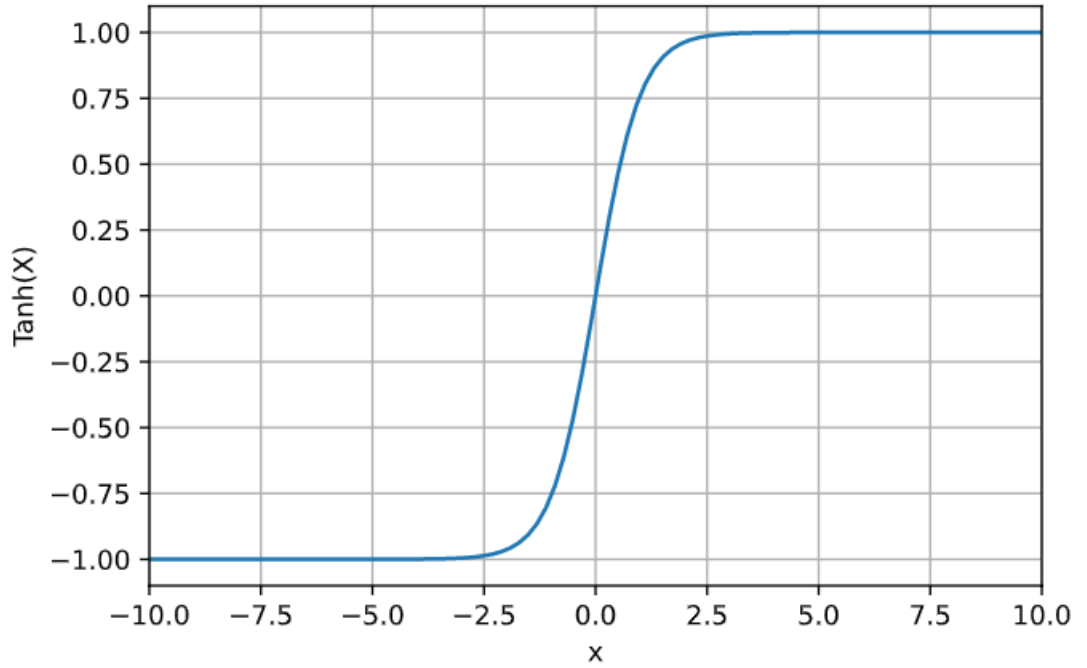


- Nhận giá trị đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0:1) Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1. Trong quá khứ hàm Sigmoid hay được dùng vì có đạo hàm rất đẹp. Tuy nhiên hiện nay hàm Sigmoid rất ít được dùng vì những nhược điểm sau:
 - Hàm Sigmoid bão hòa và triệt tiêu gradient: Một nhược điểm dễ nhận thấy là khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương), gradient của hàm số này sẽ rất gần với 0. Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật (còn được gọi là vanishing gradient).
 - Hàm Sigmoid không có trung tâm là 0 gây khó khăn cho việc hội tụ.

3.2.2. Tanh

- Công thức:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Đồ thị



- Hàm nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (-1;1). Cũng như Sigmoid, hàm Tanh bị bão hoà ở 2 đầu (gradient ít thay đổi ở 2 đầu)
- Hàm tanh còn có thể được biểu diễn bằng hàm sigmoid như sau:

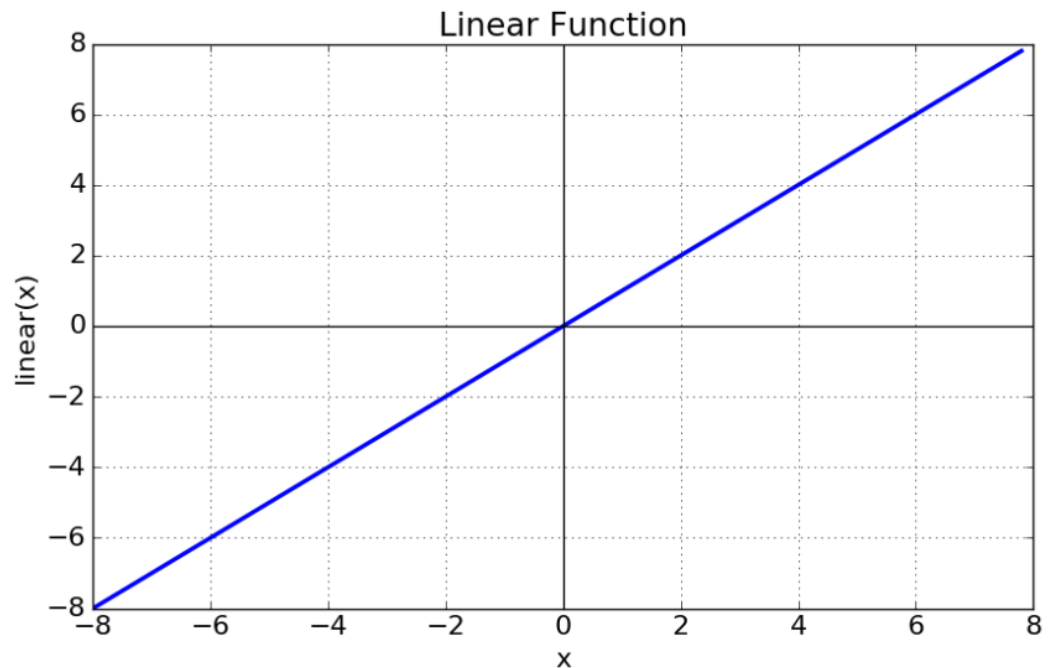
$$\tanh(x) = 2\sigma(2x) - 1$$

- Chức năng kích hoạt tanh được sử dụng trong mạng chuyển tiếp.
- Ưu điểm
 - Nhận 0 làm điểm giữa—tạo ra sự tách biệt giữa âm, dương và giá trị 0 (tương ứng positive, negative, neutral)
 - Đạo hàm mịn, tránh “nhảy” giá trị output.
 - Khoảng giá trị output nằm giữa -1 và 1, dùng để chuẩn hóa giá trị output của neuron.

- Clear predictions - với X lớn hơn 2 hoặc nhỏ hơn -2, hàm này sẽ biến Y (predict) thành một giá trị gần rìa đường cong, xấp xỉ 1 hoặc -1. Điều này làm cho việc predict trở nên rõ ràng.
- Nhược điểm
 - Vanishing gradient (Khó tối ưu tham số ở các layer càng về cuối cùng) — với X lớn, giá trị của Y sẽ hầu như không đổi, gây ra vanishing gradient problem. Điều này khiến mạng nơ ron ngừng học, hoặc học rất chậm.
 - Chi phí tính toán cao

3.2.3. Linear

- Công thức: $f(x) = ax$
- Phạm vi từ: $(-\infty : +\infty)$
- Đồ thị:



- Là hàm tuyến tính, trong đó a là một hằng số
- Kết quả cuối cùng có thể rất lớn, không bị giới hạn bởi bất kì phạm vi nào
- Không nắm bắt được những cấu trúc phức tạp của dữ liệu
- Hàm kích hoạt tuyến tính hầu như không được sử dụng trong deep learning

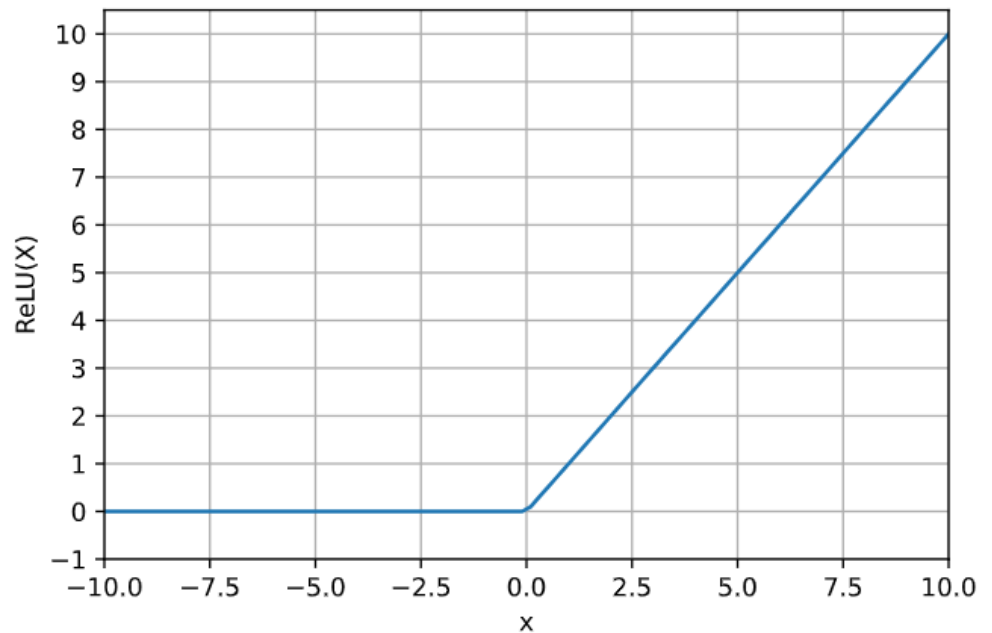
3.2.4. Relu

- Công thức:

$$f(x) = \max(0, x)$$

- Phạm vi: $(0 : +\infty)$

- Đồ thị:



- Nếu đầu vào nhỏ hơn hoặc bằng 0 thì Relu sẽ xuất ra 0. Nếu đầu vào lớn hơn 0 thì Relu sẽ xuất giá trị đó ra
- Ưu điểm vượt trội của Relu so với tanh và sigmoid:
 - Tốc độ hội tụ nhanh
 - Không bị bão hòa ở 2 đầu như tanh và sigmoid
 - Giảm thiểu chi phí và tăng tốc độ tính toán
- Bên cạnh đó Relu cũng có một số nhược điểm:
 - Với các node có giá trị nhỏ hơn 0, qua Relu activation sẽ thành 0, gây ra hiện tượng “Dying ReLU”. Các node có giá trị 0 sẽ không có ý nghĩa với bước linear activation ở lớp tiếp theo dẫn đến không cập nhật được gradient descent
 - Khi learning rate lớn, các trọng số (weights) có thể thay đổi theo cách làm tất cả các neuron dừng việc cập nhật



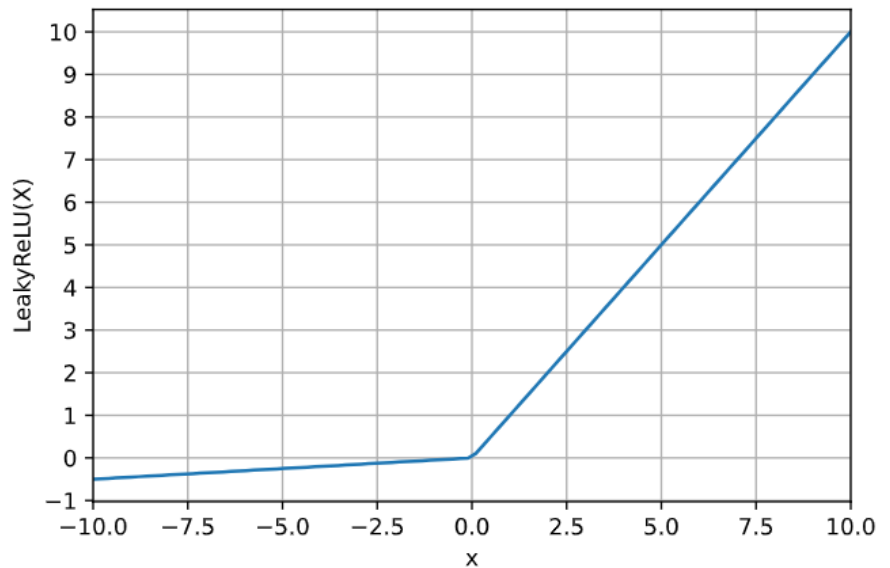
3.2.5. Leaky Relu

- Công thức: $f(x) = \max(\alpha x, x)$

$$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geq 0)(x)$$

với α là hằng số nhỏ.

- Phạm vi: $(-\infty : +\infty)$
- Đồ thị:



- Leaky Relu cố gắng loại bỏ hiện tượng “Dying Relu” bằng cách với các giá trị đầu vào < 0 , hàm Leaky Relu sẽ trả về giá trị là một đường xiên tăng dần đến 0 với độ dốc nhỏ

3.2.6. Prelu

- Công thức:

$$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

- Phạm vi: $(-\infty : +\infty)$
- Tương tự Leaky Relu nhưng Prelu cho phép neuron tự chọn hệ số α sao cho tốt nhất

3.3. Một số hàm metrics và losses trong tính toán tổn thất huấn luyện

- Về cơ bản cách thức hoạt động của metrics và losses giống nhau, điểm khác nhau cụ thể nằm ở việc:
 - Losses function dùng để tối ưu mô hình huấn luyện
 - Metrics dùng để đánh giá hiệu suất của mô hình
- Trong khuôn khổ đề án, ta chỉ tìm hiểu đến 2 dạng đó là Probabilistic và Regression
- Probabilistic (sai số xác suất)
 - BinaryCrossentropy: tính toán sự mất mát chéo giữa các nhãn thực và nhãn dự đoán. Chỉ áp dụng khi đầu ra chỉ có 2 lớp (0 và 1).
 - CategoricalCrossentropy: tính toán tổn thất chéo giữa các nhãn thực và nhãn dự đoán. Sử dụng khi các lớp đầu vào có nhãn là một chuỗi, đầu ra có 2 hay nhiều lớp.
 - Sparse CategoricalCrossentropy: tính toán tổn thất chéo giữa các nhãn thực và nhãn dự đoán. Sử dụng khi các lớp đầu vào có nhãn là một số nguyên, đầu ra có 2 hay nhiều lớp.
- Regression (sai số hồi quy)
 - MeanSquareError: tính toán sai số bình phương giữa nhãn thực và nhãn dự đoán
 - MeanAbsoluteError: tính toán sai số tuyệt đối trung bình giữa nhãn thực và nhãn dự đoán
 - CosineSimilarity: tính toán độ giống nhau về cosin giữa nhãn thực và nhãn dự đoán

$$\text{cosine similarity} = (a \cdot b) / ||a|| \ ||b||$$



3.4. Một số layer thường dùng

3.4.1. Dropout

- Dropout là việc bỏ qua các đơn vị (nút mạng) trong quá trình đào tạo một cách ngẫu nhiên. Bằng cách này thì đơn vị đó sẽ không được xem xét trong quá trình forward và backward
- Dropout layer quan trọng trong mô hình là vì: nếu 1 lớp fully connected có quá nhiều tham số và chiếm hầu hết tham số, các nút mạng trong lớp đó quá phụ thuộc lẫn nhau trong quá trình huấn luyện thì sẽ hạn chế sức mạnh của mỗi nút, dẫn đến việc kết hợp quá mức gây ra overfitting
- Một số kỹ thuật có cùng tác dụng như dropout:
 - Kỹ thuật chính quy hóa (regularization) sẽ làm giảm over-fitting bằng cách thêm 1 khoảng giá trị “phạt” vào hàm loss. Từ đó mô hình sẽ không học quá nhiều sự phụ thuộc giữa các trọng số.
 - Một số kỹ thuật làm giảm over-fitting dùng trong Logistic Regression phải kể đến là Laplacian và Gaussian

3.4.2. Batch normalization

- Batch normalization thực hiện chuẩn hóa một phần của model, và thực hiện trên mỗi đợt training minibatch. Batch normalization cũng cho phép sử dụng learning rate cao hơn nhiều và ít phải cẩn thận hơn khi khởi tạo. nó cũng hoạt động như một bộ điều chỉnh, trong một số trường hợp Dropout.
- Batch normalization layer thường được sử dụng với các mô hình MLP, CNN và RNN.
- Sử dụng Batch normalization giữa các lớp tuyến tính và phi tuyến trong mạng, vì nó normalize đầu vào cho activation function.



3.4.3. GlobalAveragePooling2D

- GlobalAveragePooling2D layer tính trung bình tất cả các giá trị theo last axis (trục cuối cùng). Điều này có nghĩa là hình dạng lớp đầu ra của layer này sẽ là (n_samples, last_axis).
- Ví dụ: input (None, 7, 7, 128) thì output sẽ là (None, 128)

3.4.4. Flatten

- Flatten layer làm phẳng input đầu vào.
- Ví dụ: input (None, 7, 7, 128) thì output sẽ là (None, 6272)
- Trong đó $6272 = 7 * 7 * 128$

3.5. Dense

- Dense layer hay còn gọi là Fully connected layer, nó là một lớp cổ điển trong mạng neuron nhân tạo. Mỗi neuron nhận đầu vào từ tất cả các neuron lớp trước đó



4. Tìm hiểu một số kiến trúc mạng

4.1. Tổng quan

- Bảng dưới đây trích từ [Keras Applications](#) thống kê lại kích thước model và accuracy tương ứng đối với tập dữ liệu imagenet

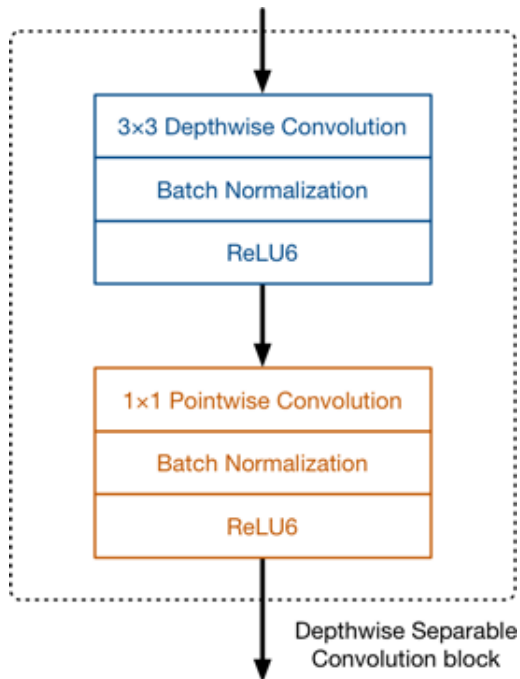
Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

- Do hạn chế về tài nguyên hệ thống, cụ thể ở đây là GPU, nên trong khuôn khổ đề án này chúng ta chỉ sử dụng các biến thể của MobileNet, để giảm thiểu tối đa chi phí tính toán và triển khai.



4.2. MobileNet V1

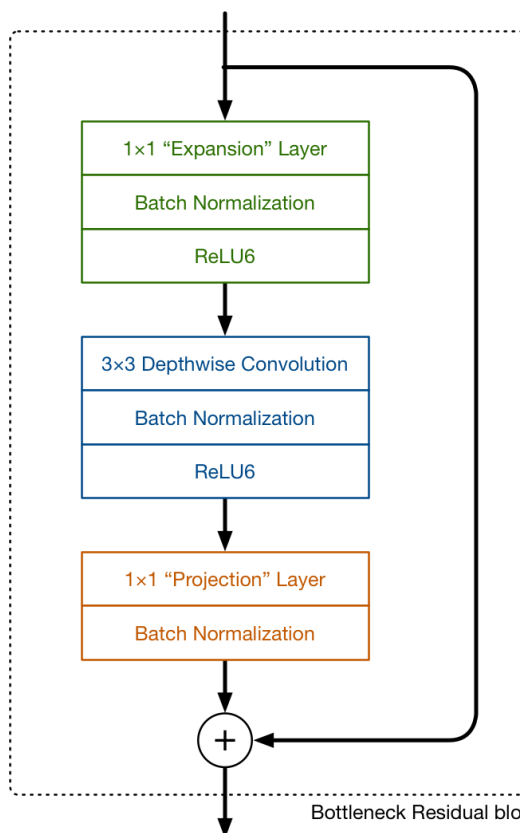


- MobileNet V1 là một tích chập phân tách theo chiều sâu (depth wise separable convolution) được sử dụng để cải thiện tốc độ tính toán của mạng. Tích chập này bao gồm depthwise convolution và pointwise convolution.

- Công việc của lớp tích chập: đầu tiên có một lớp tích chập theo chiều sâu lọc các giá trị đầu vào, tiếp theo là một lớp tích chập 1 x 1 kết hợp các giá trị đã lọc này để tạo nên tính năng mới

- Activation được sử dụng là ReLU6

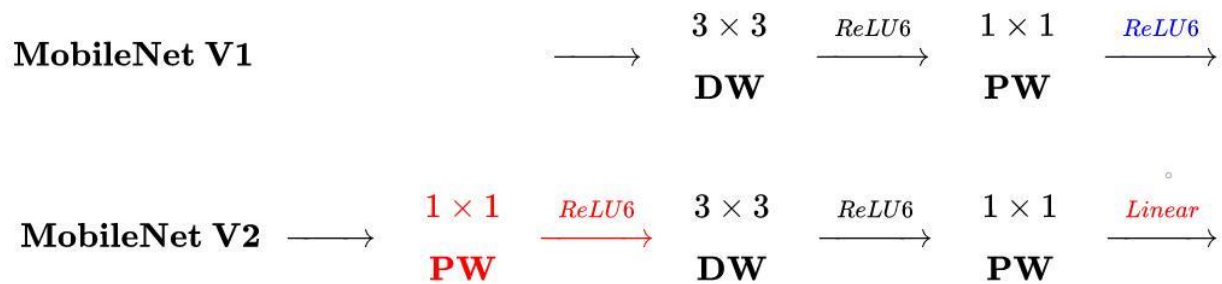
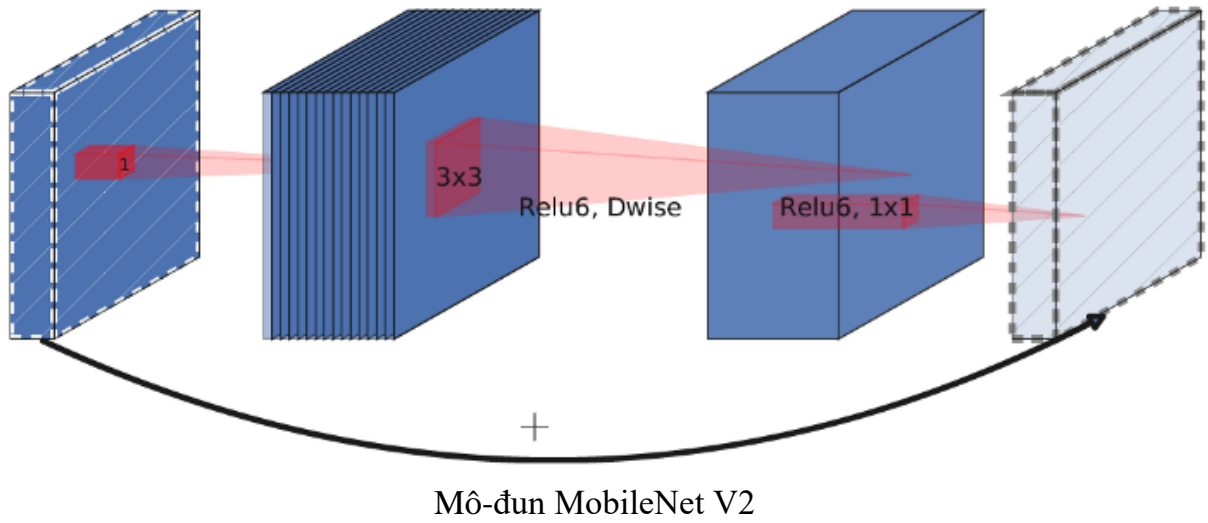
4.3. MobileNet V2



- MobileNet V2 có ba lớp tích hợp trong khối. Hai khối cuối cùng trong ba khối này tương tự như V1 nhưng ở lớp tích chập 1 x 1 thay vì giữ nguyên hoặc nhân đôi số lượng giá trị đầu ra thì ở đây V2 lại làm ngược lại. Nó làm giảm số lượng đầu ra của dữ liệu.

- Activation được sử dụng là ReLU6 ngoại trừ layer cuối cùng sử dụng linear

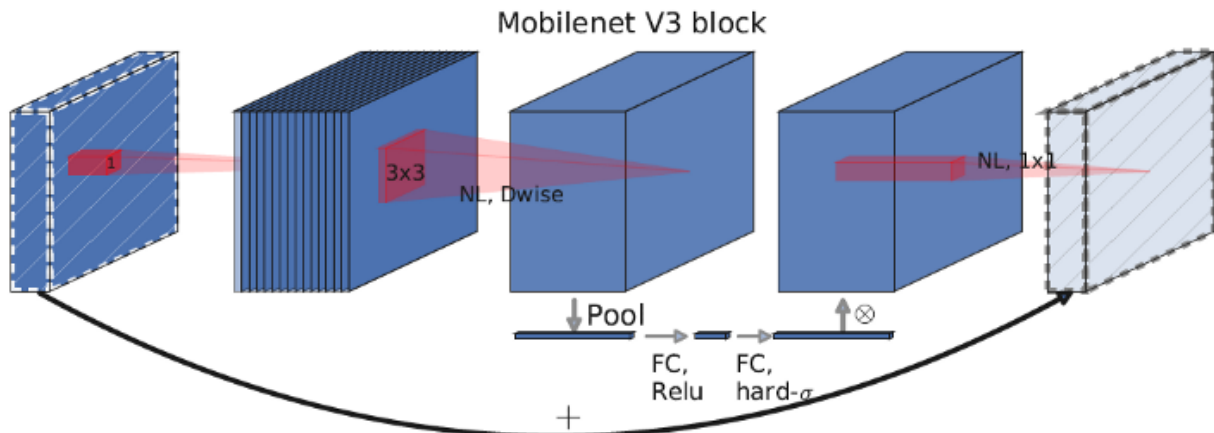
MobileNet V2: bottleneck with residual



4.4. MobileNet V3

- MobileNet V3 được điều chỉnh để phù hợp với CPU của di động thông qua sự kết hợp của tìm kiếm kiến trúc mạng nhận biết phân cứng (NAS), tức là MnasNet, được bổ sung bởi thuật toán NetAdapt
- So với MobileNetV2, MobileNetV3 đã chèn thêm mô-đun Squeeze and Excitation (SE), có nguồn gốc từ SENet
- Hard-Sigmoid được sử dụng để thay thế sigmoid trong mô-đun SE để tính toán hiệu quả hơn
- Hàm kích hoạt mới h-swish (x):

$$\text{h-swish}[x] = x \frac{\text{ReLU6}(x + 3)}{6}$$



- MobileNet V3 = MobileNet V2 + SE + hard-swish activation + half initial layers channel & last block do global average pooling first

4.5. So sánh MobileNet V1 V2 V3

- Trên bộ thử nghiệm COCO của SSDlite ta thấy:
 - MobileNet V3-Large nhanh hơn 27% so với MobileNet V2
 - MobileNet V3-Large nhanh hơn 48% so với MobileNet V1
 - Trong khi vẫn giữ được mAP ở mức tốt.

Backbone	mAP	Latency (ms)	Params (M)	MAdds (B)
V1	22.2	228	5.1	1.3
V2	22.1	162	4.3	0.80
MnasNet	23.0	174	4.88	0.84
V3	22.0	137	4.97	0.62
V3 [†]	22.0	119	3.22	0.51
V2 0.35	13.7	66	0.93	0.16
V2 0.5	16.6	79	1.54	0.27
MnasNet 0.35	15.6	68	1.02	0.18
MnasNet 0.5	18.5	85	1.68	0.29
V3-Small	16.0	52	2.49	0.21
V3-Small [†]	16.1	43	1.77	0.16

Table 6. Object detection results of SSDLite with different backbones on COCO test set. [†]: Channels in the blocks between C4 and C5 are reduced by a factor of 2.

Nguồn: [Everything you need to know about MobileNetV3 | by Vandit Jain | Towards Data Science](#)



IV. Xây dựng model

1. Tổng quan về cấu trúc model:

- Bước 1: đọc data từ dataset
- Bước 2: tách data thành train_ds, test_ds, val_ds theo tỉ lệ 70% / 15% / 15%
- Bước 3: biểu diễn trực quan dữ liệu theo label
- Bước 4: biểu diễn trực quan dữ liệu sau khi data argument
- Bước 5: load dữ liệu vào cache để tăng tốc độ đọc dữ liệu
- Bước 6: load model từ tensorflow.keras.application
- Bước 7: xây dựng model để huấn luyện với việc thêm các fully-connected layer và đóng băng model ở bước 5
- Bước 8: tiến hành training model trên tập dữ liệu train và val_data
- Bước 9: mở băng một số layers model ở bước 5, tiến hành fine tuning
- Bước 10: lưu lại history để tiến hành đánh giá model
- Bước 11: lưu model xuống kho lưu trữ
- Bước 12: đánh giá model dựa trên tốc độ hội tụ của losses và metrics
- Bước 13: tính toán sai số của model trên tập dữ liệu test
- Bước 14: đánh giá model dựa trên sai số tính toán
- Bước 15: đánh giá model dựa trên kích thước model ở kho lưu trữ
- Bước 16: lặp lại bước 6 đến khi đạt được kết quả như mong đợi
- Bước 17: thử nghiệm model với dữ liệu thực tế



2. Dataset preprocessing

2.1. Khởi tạo biến toàn cục

```
[ ] width = height = 224
    img_size = (width, height)
    img_shape = img_size + (3, )
    batch_size = 32
    initial_epochs = 50
    fine_tune_epochs = 50
    seed = 10
    learning_rate = 1e-4
    results = {}
```

2.2. Đọc dữ liệu với tensorflow

- Thư viện tensorflow hỗ trợ hàm `image_dataset_from_directory` để đọc dữ liệu nhanh chóng

```
from tensorflow.data.experimental import cardinality
from tensorflow.keras.preprocessing import image_dataset_from_directory
train_ds = image_dataset_from_directory(
    data_path,
    seed=seed,
    image_size=img_size,
    smart_resize=True,
    batch_size=batch_size)
train_batches = cardinality(train_ds)
```

2.3. Tách dữ liệu thành train/test/val

- Tạo dữ liệu `test_ds` và `val_ds` từ tập dữ liệu `train_ds` với hàm `take` và `skip`

Test Dataset

```
test_ds = train_ds.take(train_batches // 20 * 3)
train_ds = train_ds.skip(train_batches // 20 * 3)
```

Val Dataset

```
val_ds = train_ds.take(train_batches // 20 * 3)
train_ds = train_ds.skip(train_batches // 20 * 3)
```



2.4. Tăng tốc độ huấn luyện với AUTOTUNE

- AUTOTUNE hỗ trợ load dữ liệu vào bộ nhớ cache tăng tốc độ đọc dữ liệu từ lần thứ 2 trở đi

Data Performance

Data augment

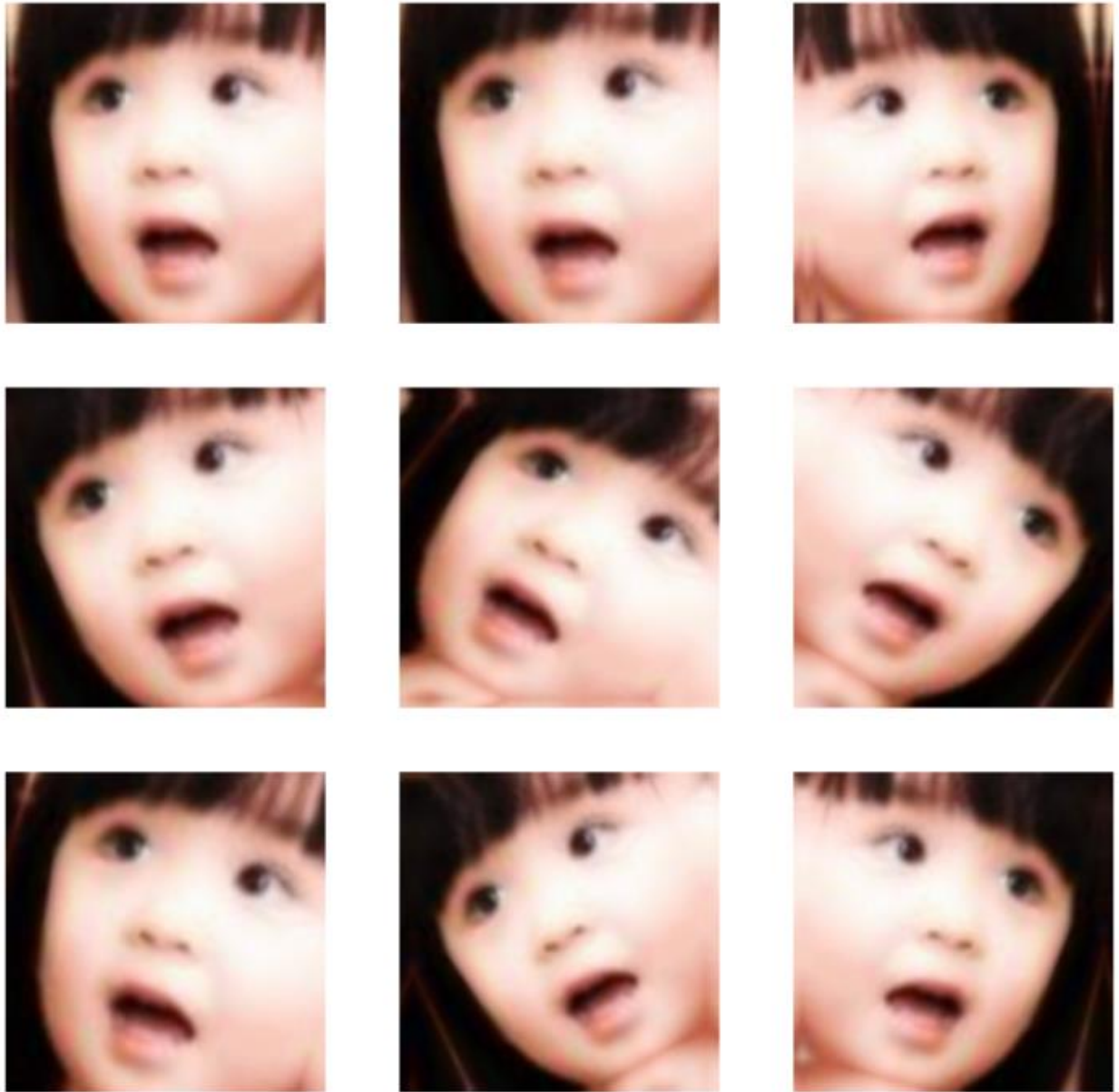
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers.experimental.preprocessing import RandomFlip, RandomRotation, RandomZoom
data_augmentation = Sequential(
    [
        RandomFlip("horizontal", input_shape = img_shape),
        RandomRotation(0.1),
        RandomZoom(-0.1,0.1),
    ]
)
```

2.5. Xây dựng layer Data augment

- Sử dụng Sequential của tensorflow.keras.models
- Trong đó :
 - RandomFlip : lật ảnh theo chiều ngang
 - RandomRotation : xoay ảnh
 - RandomZoom : thu phóng ảnh



- Đây là kết quả của một ảnh sau khi trải qua quá trình Data Augment



2.6. Tạo Callback

- Hàm Callback giúp điều chỉnh learning rate trong quá trình học tập đồng thời kết thúc sớm quá trình training khi model đạt được một kết quả nhất định



Call Back

```
from keras.callbacks import Callback, ReduceLROnPlateau, EarlyStopping
from timeit import default_timer as timer

class TimingCallback(Callback):
    def __init__(self, logs={}):
        self.logs=[]
    def on_epoch_begin(self, epoch, logs={}):
        self.starttime = timer()
    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(timer()-self.starttime)

early_stopping = EarlyStopping(
    patience=5, # wait for 5 epochs
    min_delta = 0.01, # if in 5 epochs the loss function doesn't increase (for accuracy)
                    # or decrease (for val_loss) by 1%, then stop
    verbose=1, # print the training epoch on which training was stopped
    mode = 'min',
    monitor='val_loss')

reduce_learning_rate = ReduceLROnPlateau(
    monitor="val_loss",
    patience=1, # if val_loss plateaus for 1 epochs such that it doesn't see
               # an improvement of size = epsilon
    epsilon= 0.01,
    factor=0.1, # then we reduce the learning rate by a factor of 0.1
    cooldown = 4, # and we wait for 4 epochs before we restart again
    verbose=1)

time_callback = TimingCallback()
```

- Trong trường hợp của bài toán này:
 - Callback sẽ kết thúc sớm quá trình training khi trong 5 epochs mà giá trị loss không tăng hoặc có dấu hiệu giảm
 - Callback sẽ giảm tỉ lệ learning rate khi từ epochs thứ 2 trở đi mà giá trị loss không giảm

3. Build model

3.1. Download Model từ tensorflow

```
from tensorflow.keras.layers import Dense, Dropout, Input, Flatten, Activation
from tensorflow.keras.layers import GlobalAveragePooling2D, BatchNormalization
from tensorflow.keras.layers import Convolution2D, ZeroPadding2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications.mobilenet import MobileNet, preprocess_input

# download model MobileNet V1
base_model_V1 = MobileNet(input_shape = img_shape,
                          include_top = False,
                          weights = 'imagenet')

# Freeze the base model
base_model_V1.trainable = False
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17227776/17225924 [=====] - 0s 0us/step



- Hình trên đây mô tả việc khởi tạo một base_model với việc download model MobileNet V1
- Với việc set thuộc tính `trainable = False` ta đóng băng toàn bộ các layer để thực hiện quá trình warnup

3.2. Xây dựng fully connected layers

```

▶ inputs = Input(shape=img_shape, dtype = tf.uint8)
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model_V1(x, training=False)
x = GlobalAveragePooling2D(input_shape=img_shape)(x)
x = Dropout(0.2)(x)
x = Dense(256, activation="tanh")(x)
outputs = Dense(1, activation="linear")(x)

# Final model
model_V1 = Model(inputs, outputs)
model_V1.compile(optimizer=Adam(learning_rate = learning_rate)
                 ,loss="mean_absolute_error"
                 ,metrics=['mean_absolute_error'])

```

- Layer 1 nhận giá trị input với shape = img_shape
- Layer 2 truyền giá trị input vào layer data_augmentation đã tạo ở trên
- Layer 3 nhận giá trị từ lớp trên truyền vào hàm preprocess_input của tensorflow
- Layer 4 nhận giá trị từ layer 3 truyền vào base_model_V1
- Layer 5 nhận giá trị output từ model V1 truyền vào GlobalAveragePooling2D để làm phẳng dữ liệu
- Layer 6 nhận giá trị từ layer 5 truyền vào hàm dropout với hệ số là 0.2
- Layer 7 nhận giá trị từ layer 6 truyền vào Dense() 256 phần tử với activation là tanh
- Layer 8 nhận giá trị từ layer 7 truyền vào outputs với Dense() 1 phần tử với activation là linear để dự đoán kết quả đầu ra tuyến tính



Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.math.truediv (TFOpLambda)	(None, 224, 224, 3)	0
tf.math.subtract (TFOpLambda)	(None, 224, 224, 3)	0
mobilenet_1.00_224 (Function)	(None, 7, 7, 1024)	3228864
global_average_pooling2d (Gl	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 256)	262400
dense_1 (Dense)	(None, 1)	257
=====		
Total params: 3,491,521		
Trainable params: 262,657		
Non-trainable params: 3,228,864		

- Trên đây là kết quả chi tiết của model được thể hiện thông qua model.summary()




```
model_v1_history = model_v1.fit(
    train_ds,
    validation_data=val_ds,
    epochs=initial_epochs,
    batch_size = batch_size,
    callbacks = [
        reduce_learning_rate,
        early_stopping,
        time_callback
    ],
    verbose=True
)
```

- Tiến hành train model với model_V1.fit() với số lượng epochs giới hạn ở mức 50

```
Epoch 32/50
216/216 [=====] - 9s 44ms/step - loss: 8.3994 - mean_absolute_error: 8.3994

Epoch 00032: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.
Epoch 00032: early stopping
```

- Sau 32 epochs, quá trình training đã bị buộc dừng lại với ảnh hưởng của hàm callbacks.
- Kết quả mean_absolute_error (MAE) nằm ở mức 8.3994

3.3. Transfer learning

```
[ ] base_model_v1.trainable = True
```

```
[ ] len(base_model_v1.layers)
```

 86

- Tiến hành warnup base_model_V1 với việc set trainable = True
- Số layers của base_model_V1 là 86
- Ta sẽ tiến hành đóng băng 10 layers đầu tiên, và train lại 76 layer phía sau




```
[ ] ft_at = 10
    for layer in base_model_V1.layers[:ft_at]:
        layer.trainable = False
    for layer in base_model_V1.layers[ft_at:]:
        layer.trainable = True
    model_V1.compile(optimizer=Adam(learning_rate = learning_rate)
                    ,loss="mean_absolute_error"
                    ,metrics=['mean_absolute_error'])
```

- Pretrain lại model với initial_epoch bắt đầu từ epoch cuối cùng của model_V1_history

```
▶ model_V1_fn = model_V1.fit(
    train_ds,
    validation_data=val_ds,
    epochs=initial_epochs + fine_tune_epochs,
    initial_epoch=model_V1_history.epoch[-1],
    callbacks = [
        reduce_learning_rate,
        early_stopping,
        time_callback
    ],
    verbose=True
)
```

- Sau 12 epoch (44 – 32), quá trình training dừng lại và đạt MAE ở mức 5.656

```
Epoch 44/100
216/216 [=====] - 30s 139ms/step - loss: 5.6256 - mean_absolute_error: 5.6256

Epoch 00044: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.
Epoch 00044: early stopping
```

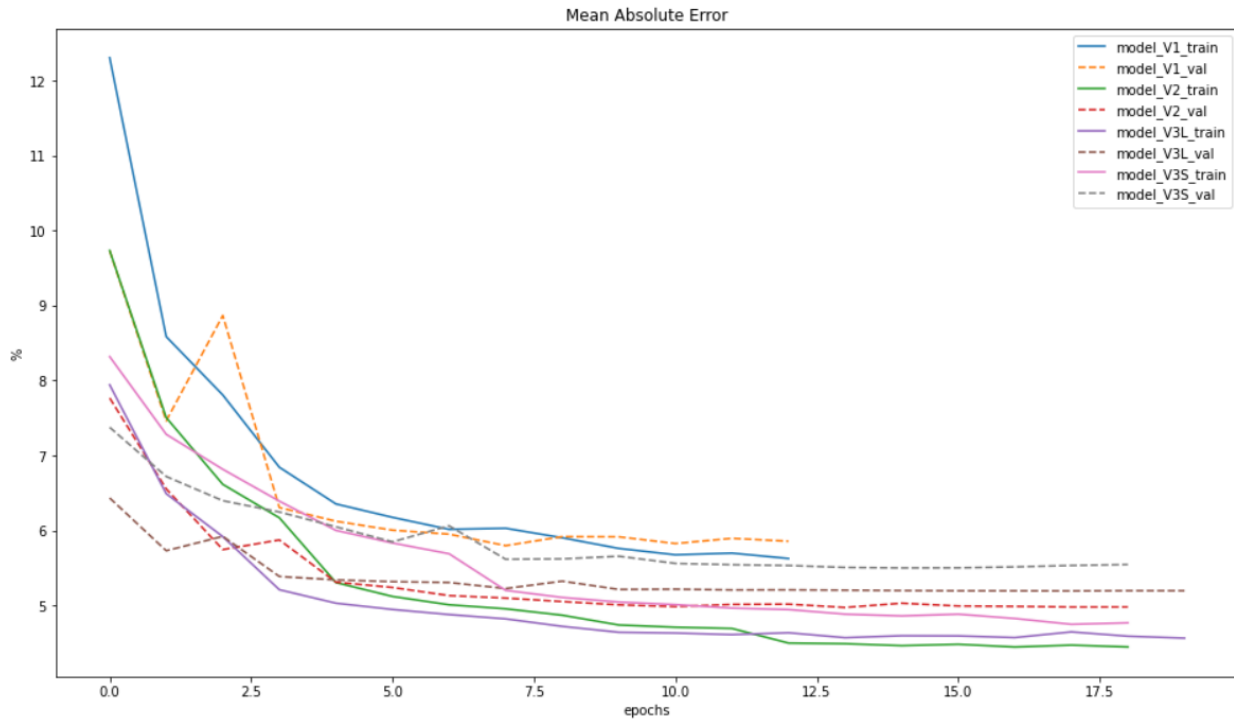
Làm tương tự với các model pre-trained khác, cùng với việc thay đổi fully - connected layer đến khi đạt được kết quả như mong muốn



4. Đánh giá các model

4.1. Fully-Connected Layers có sử dụng Data Augment

- Show dữ liệu từ history_training



- Sau khoảng 18 epochs model_V2 đạt được độ chính xác cao nhất

```
[ ] loss_V1, acc_V1 = model_V1.evaluate(test_ds)
```

```
45/45 [=====] - 2s 35ms/step - loss: 5.7721 - mean_absolute_error: 5.7721
```

```
[ ] loss_V2, acc_V2 = model_V2.evaluate(test_ds)
```

```
45/45 [=====] - 2s 40ms/step - loss: 4.9433 - mean_absolute_error: 4.9433
```

```
[ ] loss_V3L, acc_V3L = model_V3L.evaluate(test_ds)
```

```
45/45 [=====] - 6s 120ms/step - loss: 5.2527 - mean_absolute_error: 5.2527
```

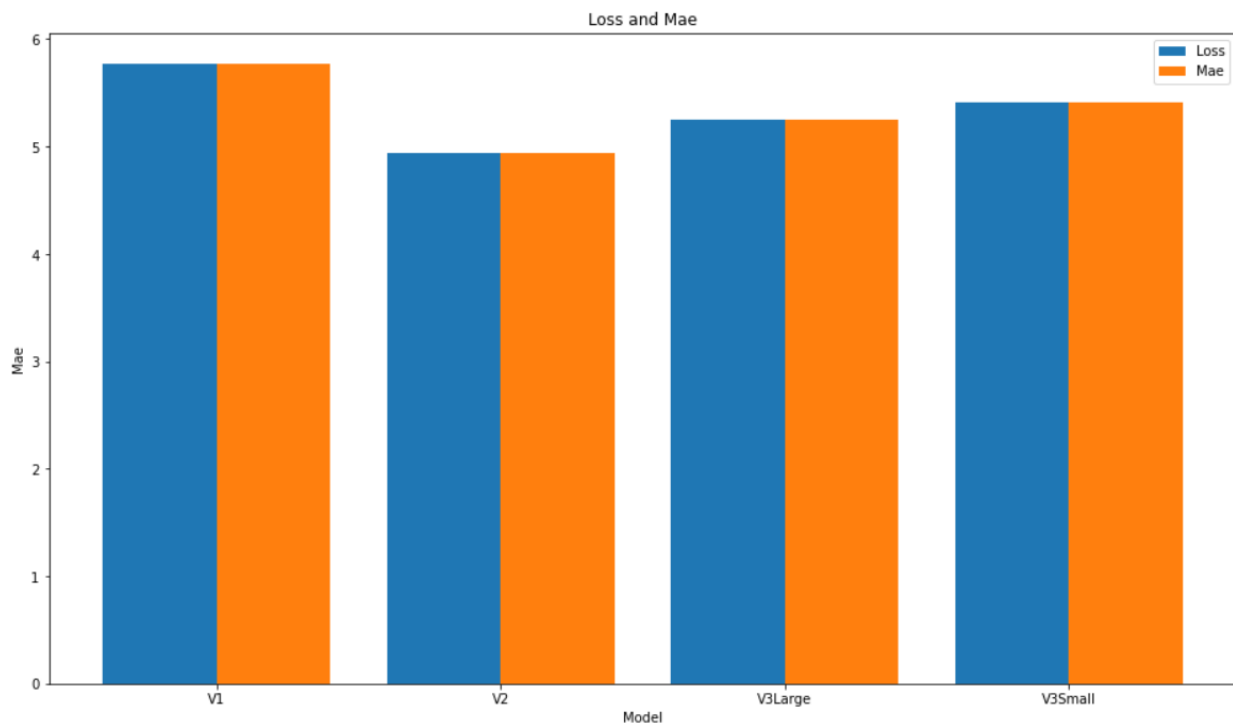
```
[ ] loss_V3S, acc_V3S = model_V3S.evaluate(test_ds)
```

```
45/45 [=====] - 1s 21ms/step - loss: 5.4152 - mean_absolute_error: 5.4152
```

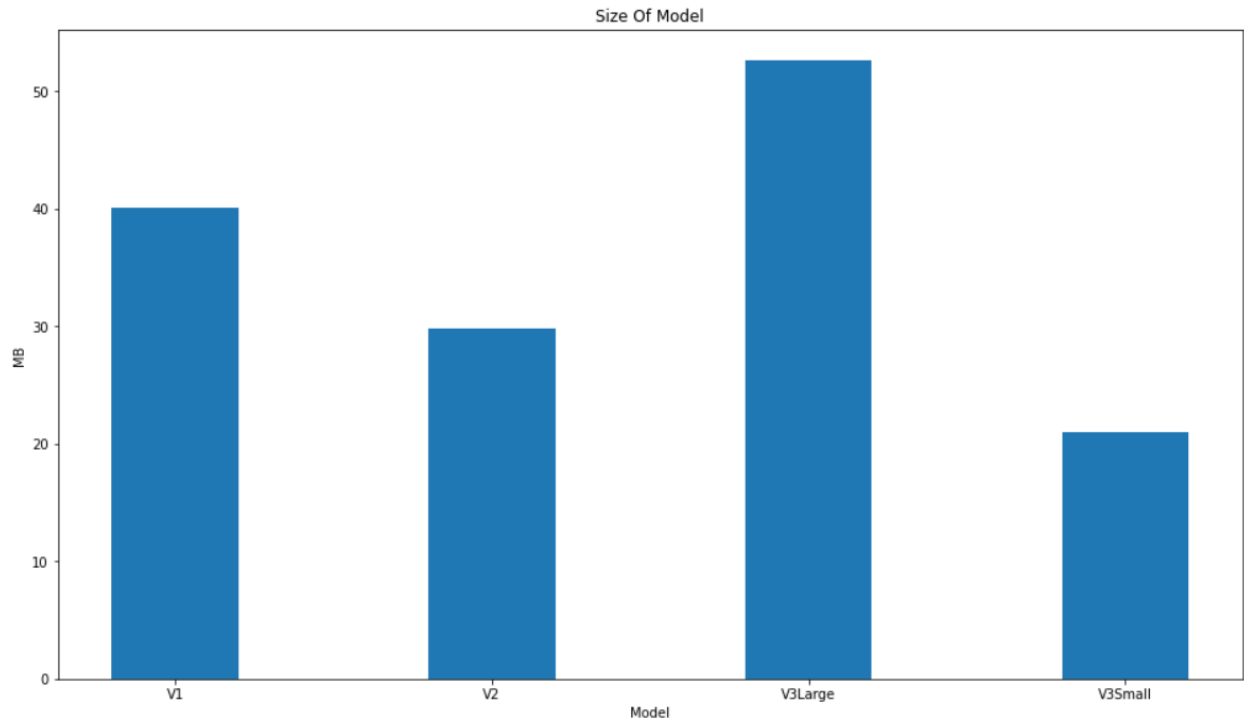
- Tính toán trên bộ dữ liệu test_ds cho thấy MAE của model_V2 đạt ngưỡng 4.9433



- So sánh kết quả tính toán của các model trên dữ liệu test_ds cũng cho kết quả tương tự như history_training với độ chính xác cao nhất thuộc về model_V2
- Nguyên nhân model_V3Large không đạt được độ chính xác cao như lý thuyết là do các fully-connected layer chưa được tối ưu và số lượng layers đóng băng trong quá trình transfer learning chưa hợp lý.



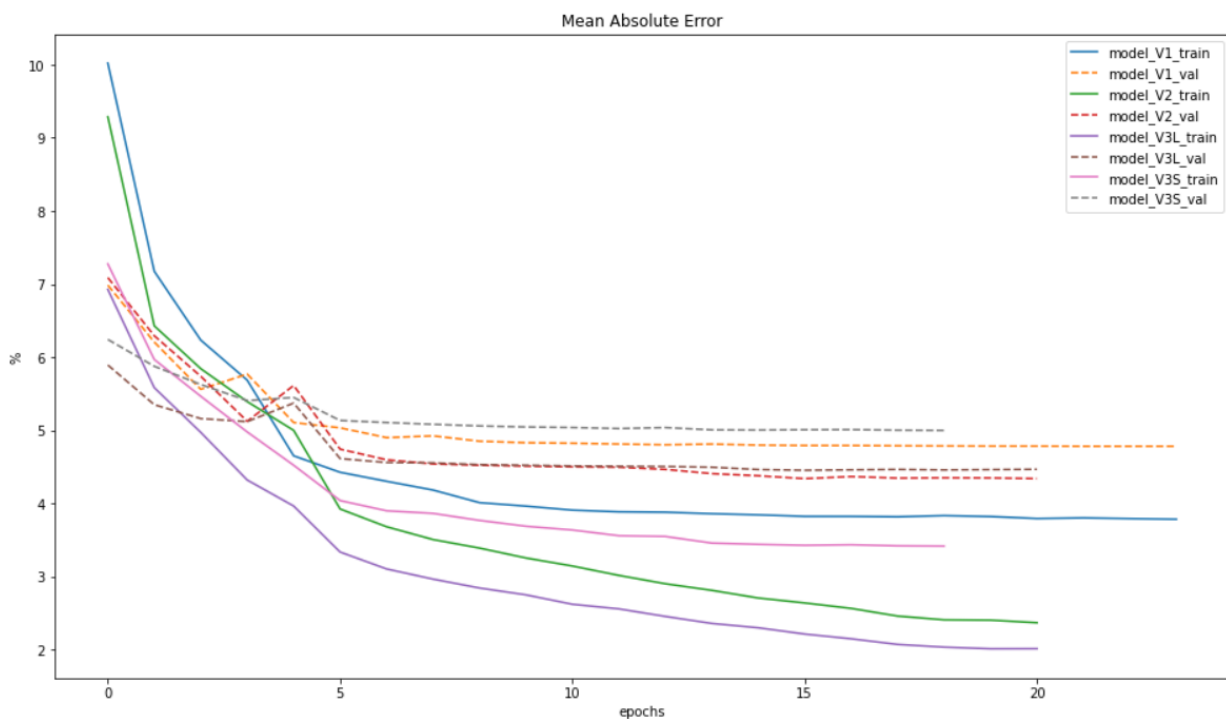
- Kích thước các model theo định dạng HDF5



- Sắp xếp theo thứ tự tăng dần về kích thước thì $V3Small < V2 < V1 < V3Large$

4.2. Fully-Connected Layers không sử dụng Data Augment

- Show dữ liệu từ history_training



- Sau khoảng 20 epochs thì model_V3Large đạt được độ chính xác cao nhất

```
[ ] loss_V1, acc_V1 = model_V1.evaluate(test_ds)
```

```
45/45 [=====] - 6s 122ms/step - loss: 4.6253 - mean_absolute_error: 4.6253
```

```
[ ] loss_V2, acc_V2 = model_V2.evaluate(test_ds)
```

```
45/45 [=====] - 2s 41ms/step - loss: 4.4813 - mean_absolute_error: 4.4813
```

```
[ ] loss_V3L, acc_V3L = model_V3L.evaluate(test_ds)
```

```
45/45 [=====] - 2s 38ms/step - loss: 4.8600 - mean_absolute_error: 4.8600
```

```
[ ] loss_V3S, acc_V3S = model_V3S.evaluate(test_ds)
```

```
45/45 [=====] - 1s 19ms/step - loss: 5.0165 - mean_absolute_error: 5.0165
```

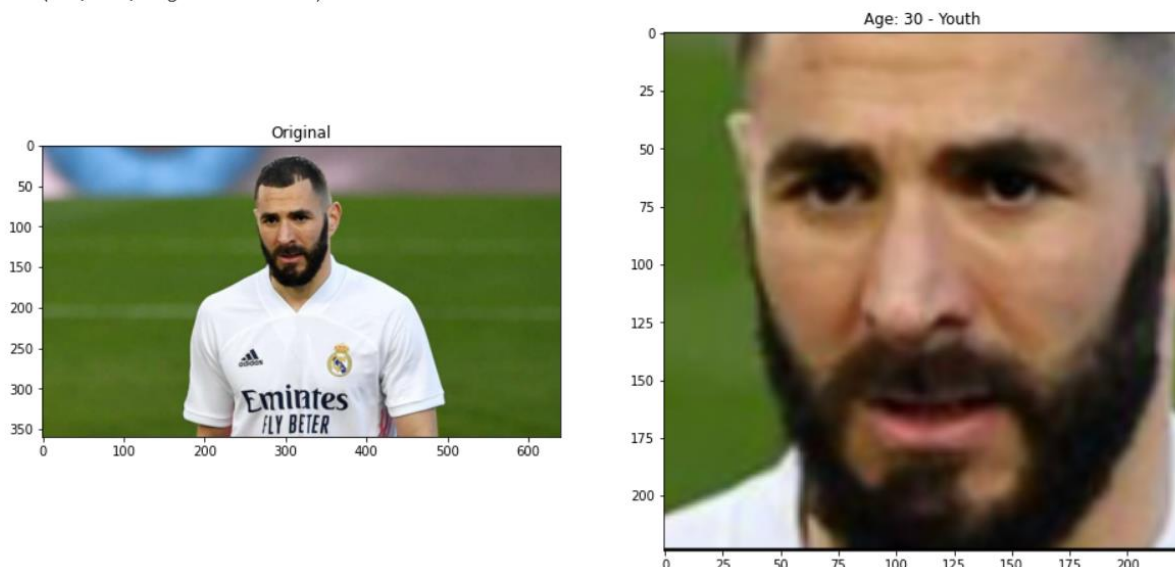
- Mặt khác, khi tính toán trên bộ dữ liệu test, model_V2 lại đạt độ chính xác MAE cao nhất ở mức 4.4813

4.3. Thử nghiệm model

- Kết quả thử nghiệm cho ra kết quả khá chính xác, chỉ sai lệch 3 tuổi so với thực tế

```
test = '/content/drive/MyDrive/FacialAge/Test_case/testcase8-33tuoi.jpg'
detected_face = detect_face(test)
age, age_class = predict_age(detected_face)
text = 'Age: ' + str(age) + ' - ' + age_class
plt.figure(figsize=(16,9))
ax = plt.subplot(1, 2, 1)
plt.imshow(mping.imread(test))
plt.title('Original')
ax = plt.subplot(1, 2, 2)
plt.imshow(detected_face)
plt.title(text)
```

Text(0.5, 1.0, 'Age: 30 - Youth')



V. Frontend React

1. Tổng quan

- React là một thư viện GUI nguồn mở JavaScript với mã nguồn mở miễn phí để xây dựng giao diện người dùng hoặc các thành phần UI. Nó được sử dụng phổ biến trên thế giới bao gồm nhiều tập đoàn lớn về công nghệ như Facebook, Netflix, WhatsApp, eBay, Instagram,...
- Ưu điểm của việc sử dụng React để phát triển front-end web:
 - React là một DOM (Document Object Model) ảo nơi mà các component thực sự tồn tại trên đó, dễ dàng trong vấn đề test giao diện.
 - Dễ dàng tái sử dụng các component từ các ứng dụng khác nhau.
 - Hiệu năng cao đối với các ứng dụng dữ liệu thay đổi liên tục
 - Dễ dàng bảo trì và sửa lỗi
- Nhược điểm của React:
 - Chỉ hỗ trợ view trong mô hình MVC vì thế khó kết hợp với các framework MVC truyền thống
 - React khá nặng so với các framework JavaScript khác
 - Khó tiếp cận đối với người mới học web

2. Axios

- Axios là một HTTP client được viết dựa trên Promises được dùng để hỗ trợ cho việc xây dựng các ứng dụng API từ đơn giản đến phức tạp và có thể được sử dụng cả ở trình duyệt hay Node.js
- Việc tạo ra một HTTP request dùng để fetch hay lưu dữ liệu là một nhiệm vụ thường thấy mà một ứng dụng JavaScript phía client cần phải làm khi muốn giao tiếp với phía server.
- Các thư viện bên thứ 3, đặc biệt là jQuery từ xưa đến nay vẫn là một trong những cách phổ biến để giúp cho các browser API tương tác tốt hơn, rõ ràng và rành mạch hơn, xóa đi những khác biệt giữa các browser với nhau.



3. Triển khai thực tế

- Tiền xử lý ảnh đầu vào
 - Ảnh người dùng nhập vào sẽ được mã hóa về dạng base64 và được lưu trữ thành biến canvas

```
15 |
16 | function getBase64Image(img) {
17 |     var canvas = document.createElement("canvas");
18 |     canvas.width = img.naturalWidth;
19 |     canvas.height = img.naturalHeight;
20 |     console.log("Kích thước canvas", canvas.width, canvas.height);
21 |     var ctx = canvas.getContext("2d");
22 |     ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
23 |     var dataURL = canvas.toDataURL("image/png");
24 |     return dataURL.replace(/^data:image\/(png|jpg);base64/, "");
25 | }
```

- Dữ liệu trước khi được gửi lên server dưới dạng formdata với key là 'img' và có type data là base 64

```
27 |
28 | function onImageLoaded(e){
29 |     const img = e.target;
30 |     var base64 = getBase64Image(img);
31 |     var bodyFormData = new FormData();
32 |     bodyFormData.append('img', base64);
33 | }
```



- Xử dụng Axios để gửi dữ liệu từ người dùng và nhận dữ liệu từ server trả về

```
34  axios({
35      method: "post",
36      url: Url,
37      data: bodyFormData
38  })
39      .then(function (response) {
40          const res_age = response['data'].age;
41          const res_cl = response['data'].cl_age;
42          // console.log(res_age);
43          const resultss = 'Độ tuổi: ' + res_age + ' Nhóm: ' + res_cl;
44          setResult(resultss);
45          setResultStyle("alert alert-primary") // Chuyển khung result sang màu xanh
46      })
47      .catch(function (error){
48          console.log(error);
49          console.log("Cannot detect face")
50          setResult("Sorry! Can not find any face in this image")
51          setResultStyle("alert alert-danger") // Chuyển khung result sang màu đỏ
52      })
53  }
```

- Dữ liệu sẽ được gửi lên server từ url người dùng nhập vào, method POST, sau đó nhận kết quả trả về từ server và hiển thị kết quả đó lên client



VI. Backend Flask

1. Tổng quan

- **Flask** là một web frameworks, nó thuộc loại micro-framework được xây dựng bằng ngôn ngữ lập trình Python. Flask cho phép bạn xây dựng các ứng dụng web từ đơn giản tới phức tạp. Nó có thể xây dựng các api nhỏ, ứng dụng web chẳng hạn như các trang web, blog, trang wiki hoặc một website dựa theo thời gian hay thậm chí là một trang web thương mại. Flask cung cấp cho bạn công cụ, các thư viện và các công nghệ hỗ trợ bạn làm những công việc trên.
- **Flask** là một micro-framework. Điều này có nghĩa Flask là một môi trường độc lập, ít sử dụng các thư viện khác bên ngoài.
 - Do vậy, Flask có ưu điểm là nhẹ, có rất ít lỗi do ít bị phụ thuộc cũng như dễ dàng phát hiện và xử lý các lỗi bảo mật.
 - Nhược điểm là đôi khi bạn phải tự thêm các danh sách phụ thuộc bằng việc thêm các plugin. Trong Flask, các phụ thuộc đó là Werkzeug WSGI và Jinja2.

2. Flask cors

- **CORS** là một cơ chế cho phép nhiều tài nguyên khác nhau (fonts, Javascript, v.v...) của một trang web có thể được truy vấn từ domain khác với domain của trang đó. CORS là viết tắt của từ Cross-origin resource sharing.
- **CORS** là một cơ chế xác nhận thông qua Header của request. Cụ thể là trên Server sẽ nói với browser về quy định chấp nhận những request từ domain nào và phương thức ra sao (GET, POST, PUT, v.v..)
 - *Access-Control-Allow-Origin*: Những origin mà server cho phép. (ví dụ server loda.his chỉ chấp nhận loda.me request lên)
 - *Access-Control-Allow-Headers*: Những Header được server cho phép. (ví dụ x-authorize, origin, cái này do server bạn quy định)
 - *Access-Control-Allow-Methods*: Những Method được server cho phép (POST, GET, v.v..)



3. SSL Flask

- SSL là viết tắt của từ Secure Sockets Layer. SSL là tiêu chuẩn của công nghệ bảo mật, truyền thông mã hoá giữa máy chủ Web server và trình duyệt. Tiêu chuẩn này hoạt động và đảm bảo rằng các dữ liệu truyền tải giữa máy chủ và trình duyệt của người dùng đều riêng tư và toàn vẹn.
- SSL hiện tại cũng là tiêu chuẩn bảo mật cho hàng triệu website trên toàn thế giới, nó bảo vệ dữ liệu truyền đi trên môi trường internet được an toàn.
- SSL đảm bảo rằng tất cả các dữ liệu được truyền giữa các máy chủ web và các trình duyệt được mang tính riêng tư, tách rời.

3.1. Certificate

- **SSL certificate** là "a bit of code" trên webserver, cung cấp bảo mật cho các giao tiếp trực tuyến. Khi web browser liên lạc với website được bảo mật, chứng chỉ SSL sẽ cho phép kết nối được mã hóa. Nó giống như niêm phong một lá thư trong một phong bì trước khi gửi đi.
- **SSL certificates** có độ tin cậy khá cao vì mỗi SSL certificate đều chứa thông tin nhận dạng. Khi bạn yêu cầu chứng chỉ SSL, bên thứ ba sẽ xác minh thông tin của tổ chức và cấp chứng chỉ duy nhất cho bạn với thông tin đó - gọi là quá trình xác thực (authentication process).
- **Certificate Authority (CA)** là tổ chức phát hành các chứng thực các loại chứng thư số cho người dùng, doanh nghiệp, máy chủ (server), mã code, phần mềm. Nhà cung cấp chứng thực số đóng vai trò là bên thứ ba (được cả hai bên tin tưởng) để hỗ trợ cho quá trình trao đổi thông tin an toàn.
- **SSL Certificate có một cặp khóa:** một công khai (public key) và khóa riêng (private key). Các khóa này làm việc cùng nhau để thiết lập một kết nối được mã hóa. SSL Certificate cũng có chứa những thông tin của chứng chỉ / website.
 - **Khóa công khai - Public key:** được share với bất kì ai/client/browser nào có nhu cầu giao tiếp với server.



- **Khóa bí mật - Private key:** được server lưu giữ cẩn thận, không được phép tiết lộ ra bên ngoài.
- Cặp khóa này được sinh ra bởi thuật toán dùng để mã hóa và giải mã data. Cụ thể là, data được mã hóa bởi Public key chỉ có thể được giải mã bởi Private key của cặp khóa đó (và ngược lại).

3.2. Decrypts

- **Decryption SSL** là một phương pháp mà các kết nối internet được giữ an toàn, cho dù chúng là máy khách với máy khách, máy chủ đến máy chủ hoặc (thường xuyên hơn) máy khách với máy chủ. Điều này ngăn các bên thứ ba trái phép nhìn thấy hoặc thay đổi bất kỳ dữ liệu nào được trao đổi trên internet.
- Khi bạn kết nối với các trang web, một kết nối được mã hóa giữa trình duyệt của bạn và trang web (ứng dụng đám mây) sẽ được thiết lập.
- Mặc dù mã hóa này ngăn những con mắt tò mò xem dữ liệu nhạy cảm của bạn, nhưng điều quan trọng là phải giảm thiểu rủi ro trong lưu lượng truy cập này. Các mối đe dọa nâng cao và phần mềm độc hại thường xuyên được phân phối trong lưu lượng được mã hóa. Đây là lúc **Decryption SSL** xuất hiện. **Decryption SSL** cho phép các tổ chức phá vỡ lưu lượng được mã hóa mở và kiểm tra nội dung của nó. Lưu lượng sau đó được mã hóa lại và gửi trên đường đi. Nhưng việc kiểm tra lưu lượng được mã hóa không phải là điều quá quan trọng và nó yêu cầu một kiến trúc proxy).



4. Triển khai thực tế

- Trong khuôn khổ đồ án, Flask API được triển khai trên server colab, hạn chế về tài nguyên tính toán và tốc độ xử lý còn khá chậm, nên server mới chỉ giao tiếp theo phương thức 1 vs 1, chưa cấu hình cài đặt bất đồng bộ và việc cài đặt SSL là không cần thiết.

```
CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

@app.route('/image', methods=['POST'])
@cross_origin(origin='*')
def process():
    img_arg_base64 = request.form.get('img')
    img1 = ConvBase64toImage(img_arg_base64)
    img2 = detect_face(img1)
    age = predict_age(img2)
    cl_age = predict_class(age)
    result = {
        'age': age,
        'cl_age': cl_age
    }
    return result
```

- Trỏ route tới '/image' với methods là 'POST' để nhận data từ Frontend
 - Bước 1: đọc dữ liệu từ frontend với request.form.get
 - Bước 2: convert data từ base64 sang image
 - Bước 3: detect face từ image với module deepface
 - Bước 4: đưa ảnh gương mặt sau detect vào model để dự đoán tuổi
 - Bước 5: trả kết quả tuổi về Frontend theo dạng dictionary

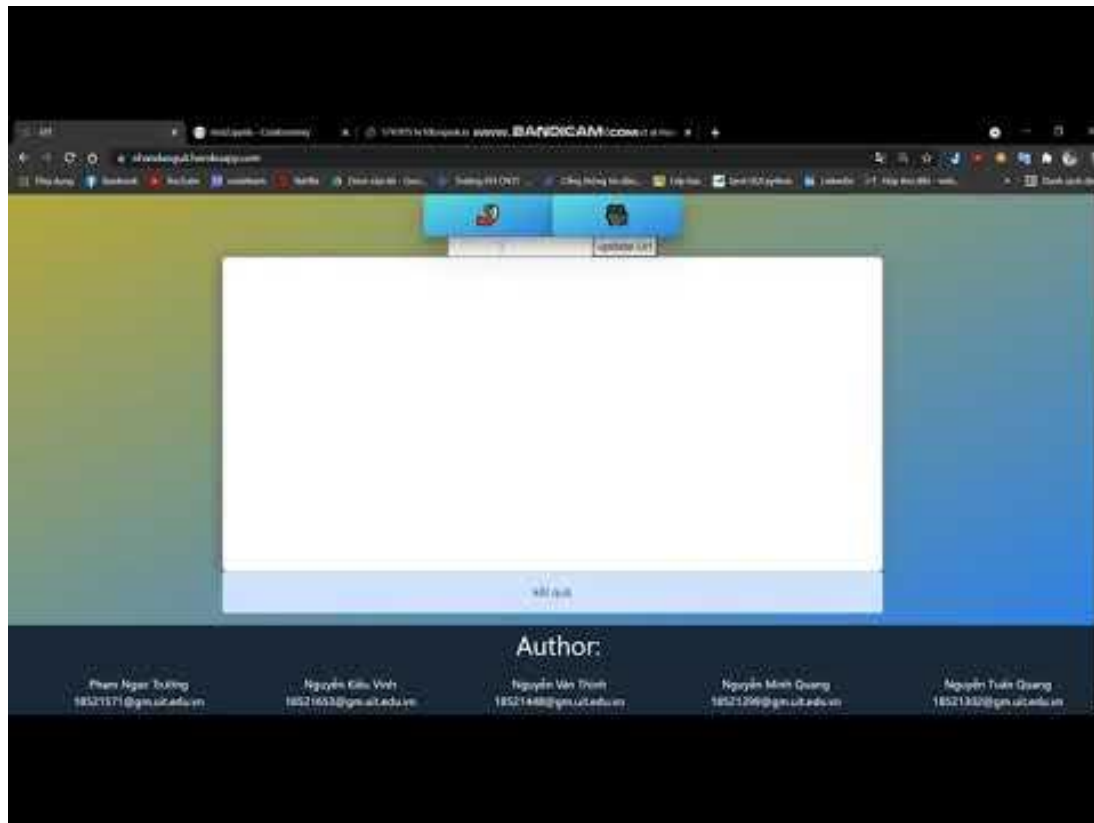


5. Server Requirements

- base64 == 1.1.4
- deepface == 0.0.65
- dlib == 19.22.0
- flask == 2.0.1
- flask_cors == 3.0.10
- numpy == 1.19.1
- opencv == 4.5.3.56
- pillow == 7.1.1
- python == 3.7
- tensorflow == 2.5
- [Yêu cầu server hỗ trợ GPU](#)

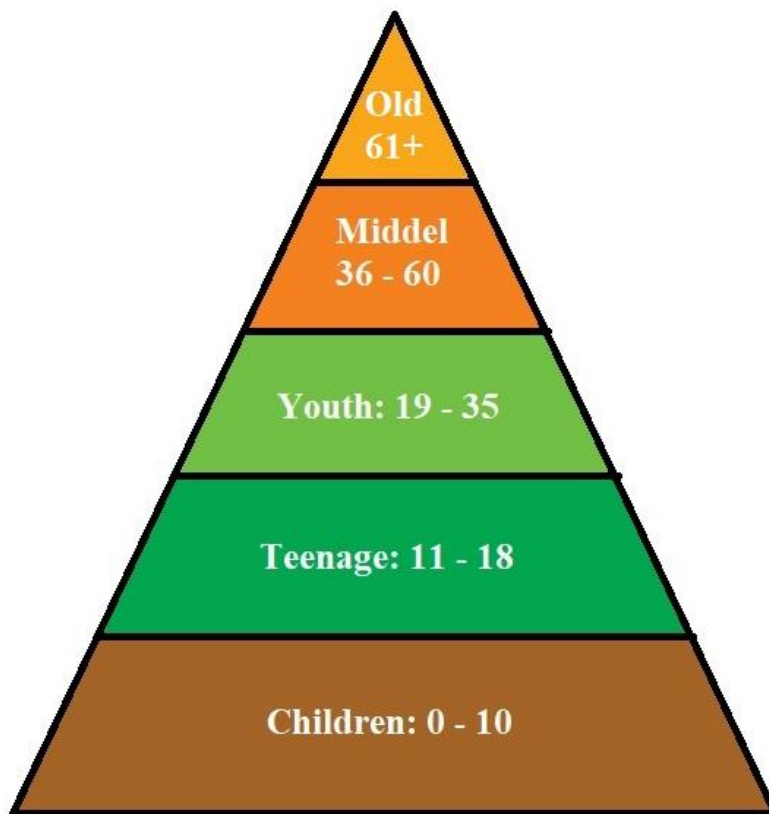


VII. Demo App



VIII. Phương hướng phát triển

- Trong thời điểm hiện tại frontend React đã có chức năng quay video trực tiếp. Trong tương lai sẽ khai thác khả năng dự đoán tuổi tác của mô hình trong thời gian thực khi server đủ mạnh.
- Cài đặt giải thuật xử lý bất đồng bộ cho server.
- Triển khai mô hình trên diện rộng với các thiết bị giám sát như camera an ninh. Từ đó, thống kê và lưu trữ số lượng khách hàng theo từng độ tuổi, xuất hiện tại mỗi khu vực để đưa ra chiến dịch marketing phù hợp.



- Mỗi nhóm tuổi sẽ có nhu cầu mua sắm và tiêu dùng khác nhau. Trên đây là cơ cấu thống kê theo nhóm tuổi.

IX. Reference

- [1]. [Keras Applications](#)
- [2]. [Losses \(keras.io\)](#)
- [3]. [Metrics \(keras.io\)](#)
- [4]. [Keras layers API](#)
- [5]. [Machine Learning cơ bản \(machinelearningcoban.com\)](#)
- [6]. [Transfer learning and fine-tuning | TensorFlow Core](#)
- [7]. [Save and load models | TensorFlow Core](#)
- [8]. [Data augmentation | TensorFlow Core](#)
- [9]. [Working with preprocessing layers | TensorFlow Core](#)
- [10]. [Better performance with the tf.data API | TensorFlow Core](#)
- [11]. [Module: tf.keras.applications | TensorFlow Core v2.5.0](#)
- [12]. [Review: MobileNetV2 — Light Weight Model \(Image Classification\)](#)
- [13]. [Everything you need to know about MobileNetV3](#)
- [14]. [LightFace: A Hybrid Deep Face Recognition Framework](#)
- [15]. [DeepFace: A Lightweight Deep Face Recognition and Facial Attribute Analysis](#)
- [16]. [Getting Started – React](#)
- [17]. [Axios - npm](#)
- [18]. [React-webcam](#)
- [19]. [React-magic-dropzone](#)

Webapp:

nhandanguit.herokuapp.com (Yêu cầu url từ API Flask server Colab để hoạt động)

Github:

[PhamTruongUit/CS338.L22.KHCL](https://github.com/PhamTruongUit/CS338.L22.KHCL)

