

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



BÁO CÁO ĐỒ ÁN CUỐI KỲ
MÁY HỌC - CS114.L21

Đề tài: *Nhận dạng món ăn đường phố Việt Nam*
(30 món)

Giảng viên: *Phạm Nguyễn Trường An*
Lê Đình Duy

Sinh viên thực hiện:		
STT	Họ tên	MSSV
1	Phạm Quang Vinh (leader)	19522526
2	Nguyễn Minh Trí	19522389
3	Trương Xuân Linh	19521759

TP. HỒ CHÍ MINH – 07/2021

MỤC LỤC

A.	GIỚI THIỆU:	1
B.	NỘI DUNG:	2
I.	Bài toán:	2
II.	Bộ dữ liệu:.....	3
1.	Cách thức xây dựng bộ dữ liệu:	3
2.	Số lượng, độ đa dạng:	7
3.	Phân chia Train/Test/Validation:	8
4.	Tiền xử lý dữ liệu:.....	9
III.	Đặc trưng của dữ liệu:	10
IV.	Thuật toán và cài đặt, tinh chỉnh tham số:.....	11
1.	SVM với rút trích đặc trưng HOG:.....	11
2.	VGG16:	15
3.	VGG16 Transfer learning:	21
4.	EfficientNet Transfer learning:.....	25
V.	Đánh giá kết quả:.....	35
VI.	Xây dựng web app:.....	36
1.	Phân tích thị trường, đối tượng sử dụng:	36
2.	Các chức năng của web-app:	36
3.	Thiết kế web-app:	36
4.	Chọn nền tảng deploy:.....	37
C.	TÀI LIỆU THAM KHẢO	38

A. GIỚI THIỆU:

Văn hóa ẩm thực Việt Nam vô cùng phong phú, đa dạng với vô vàn những món ngon cùng cách chế biến hòa trộn độc đáo, tinh tế, hài hòa giữa các nguyên liệu tự nhiên với thói quen ăn uống của người Việt. Philip Kotler, cha đẻ của học lý tiếp thị hiện đại đã khuyến khích: “Việt Nam hãy là bếp ăn của thế giới”, để thấy rằng ngoài giá trị mang tính chuyên chở lịch sử, văn hóa bản địa, thì ẩm thực Việt còn như một “đại sứ” đặc biệt để quảng bá hình ảnh VN ra thế giới đặc biệt là ẩm thực đường phố truyền thống Việt.

Ngày nay, ẩm thực đường phố Việt Nam đã và đang thu hút lượng lớn khách du lịch hằng năm, tuy nhiên với sự đa dạng các món ngon cùng tên gọi vùng miền và cách phối trộn các phong phú các loại gia vị nguyên liệu địa phương cũng gây ra một số rào cản trong việc tiếp cận của du khách đến ẩm thực đường phố, như sau:

- Ẩm thực đường phố Việt đa dạng về món ăn và tên gọi địa phương của chúng cùng với rào cản ngôn ngữ, khiến khách du lịch khó khăn trong việc ghi nhớ tên các món ăn để có thể tìm hiểu thêm về văn hóa ẩm thực hay chỉ đơn giản là để thưởng thức lại món ăn đó.
- Với sự hòa trộn nhiều nguyên liệu, gia vị đặc trưng của miền nhiệt đới, du khách sẽ khó có thể biết được các nguyên liệu có trong món ăn, điều này đặc biệt ảnh hưởng đến những du khách có cơ địa dị ứng với một số loại thực phẩm hay gia vị.
- Hiện nay, một bộ phận nhỏ tiểu thương buôn bán ẩm thực đường phố có tình trạng "chặt chém" tăng giá đối với khách du lịch.
- Lí do cuối cùng đó là hiện nay trên thế giới chỉ tập trung xây dựng tập dữ liệu món ăn phương Tây, nên từ việc nhận dạng các món ăn đường phố chúng ta có thể mở rộng ra với tập dữ liệu rộng hơn là tất cả các món ăn Việt giúp ích rất nhiều trong việc nhận dạng món ăn Việt và hơn thế nữa là phát triển ứng dụng tính lượng calo trong thức ăn từ đó giúp người Việt thoát khỏi các căn bệnh thế kỷ như béo phì, tim mạch,...

Chính vì những lí do trên chúng em đề xuất đề án: "**Nhận dạng món ăn đường phố Việt Nam qua hình ảnh**" bằng cách xây dựng model máy học nhận để nhận dạng hình ảnh món ăn đường phố Việt và từ đó có thể phát triển một ứng dụng cung cấp thông tin về tên món (cách đọc), thành phần nguyên liệu, khoản giá, cũng như các địa điểm có bán nổi tiếng (trên địa bàn TP.HCM) từ ảnh chụp.

B. NỘI DUNG:

I. Bài toán:

Với đề tài này chúng em mong muốn xây dựng một model máy học có thể thực hiện việc **nhận dạng được món ăn** (*food recognition*) hay giải quyết bài toán **phân lớp** (*classification*) cho 30 món ăn qua đầu vào hình ảnh với:

1. Input:

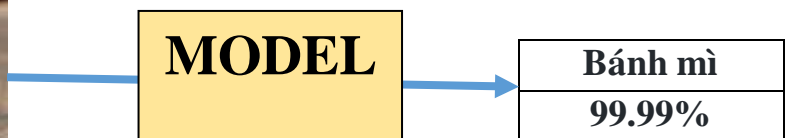
- Một bức ảnh chụp món ăn:
 - Góc chụp tùy ý đủ thấy toàn bộ món ăn.
 - Chỉ có một món ăn trong ảnh.

2. Output:

- Tên món ăn trong ảnh
- Xác suất dự đoán món đó so với các món trong danh sách.

Nếu món ăn có xác suất cao nhất bé hơn 70% thì xuất “Can’t identify this food”

Ví dụ:



II. Bộ dữ liệu:

1. Cách thức xây dựng bộ dữ liệu:

a. Thiết kế danh sách các món ăn đường phố Việt:

Sau khi xem xét các món ăn đường phố ở Việt Nam, chúng em đưa ra được danh sách 30 món ăn, ưu tiên những món ăn là đặc trưng của Việt Nam và thường có thể bắt gặp được ở bất kỳ đâu trên đường phố Việt, là những món ăn quen thuộc đối với người Việt hoặc là một đặc sản vùng miền nào đó.

Danh sách các món ăn như sau:

- | | | |
|-------------------|-------------------------|------------------|
| – Bánh bèo | – Cơm cháy | – Bún thịt nướng |
| – Bánh khọt | – Phá lấu | – Cơm tấm |
| – Bánh tráng trộn | – Bánh căn | – Phở |
| – Bún chả | – Bánh mì nướng muối ớt | – Bánh cuốn |
| – Chuối chiên | – Bắp xào | – Bánh giò |
| – Nem chua | – Bánh tráng nướng | – Bánh xèo |
| – Bánh bột lọc | – Bột chiên | – Bún bò Huế |
| – Bánh mì | – Cao lầu | – Cháo long |
| – Xôi gấc | – Gỏi cuốn | – Mỳ quảng |
| – Bún đậu mắm tôm | – Súp cua | – Xoài lắc |



b. Tiêu chí thu thập dữ liệu:

Do tình hình dịch bệnh nên không việc thu thập ảnh chụp món ăn thực tế là vô dùng khó khăn chính vì thế chúng em đã chuyển qua thu thập hình ảnh từ Internet. Với số lượng đồ sộ các hình ảnh trên Internet để có một bộ dữ liệu tốt thì chúng ta cần thiết kế một bộ tiêu chí cụ thể và chi tiết nhất.

Lấy những ảnh có đặc điểm sau:

- Ảnh những món có trong danh sách nêu trên.
- Ảnh chỉ có một món ăn.
- Ảnh chụp toàn phần món ăn, ưu tiên góc chụp từ trên xuống.

Không lấy những ảnh có đặc điểm:

- Ảnh có bao gồm nhiều món ăn.
- Ảnh có nhiều chi tiết phụ che lấp món ăn (chữ, hiệu ứng, đồ vật, người...)
- Ảnh món ăn có bao bì che bên ngoài.
- Ảnh trùng lặp.
- Ảnh không liên quan đến món ăn.

Ví dụ một số ảnh đúng tiêu chí, sẽ lấy:



Một số ảnh không lấy:



Có nhiều món



Không rõ món ăn



Bị chữ che lấp



Ảnh bị trùng lặp

c. Lưu ý về các món có ngoại hình giống nhau:

- **Bánh căn và Bánh khọt:** giống nhau vì gồm nhiều cái hình tròn
 - **Bánh căn:** có màu trắng, ở giữa có trứng, ăn với nước chấm xú mại.
 - **Bánh khọt:** có màu vàng, ở giữa là tôm, thịt, ăn với nước mắm pha.



Bánh căn



Bánh khọt

- **Phở và Bún bò Huế:** đều là món nước với sợi trắng và thịt bò và rau
 - **Phở:** sợi phở trắng đẹp, nước dùng trong
 - **Bún bò Huế:** sợi bún tròn, nước dùng thường có màu dầu điều.



Phở



Bún bò Huế

- **Bún thịt nướng và Bún chả:** đều gồm bún và thịt nướng
 - **Bún thịt nướng:** bún, rau mùi và nước mắm pha thường được trộn chung và có topping thịt nướng, chả giò, đậu phộng và đồ chua.
 - **Bún chả:** bún, nước dùng để riêng, nước dùng gồm thịt nướng, đu đủ.



Bún thịt nướng



Bún chả

- Bên cạnh các món rất giống nhau khó nhận biết kể trên, còn có nhiều món do gốc chạp, độ sáng, cách trình bày khiến chúng trông giống nhau.

d. Thu thập, lọc và gán nhãn dữ liệu:

Sau khi đã xây dựng bộ tiêu chí thu thập dữ liệu chúng em bắt đầu vào việc thu thập dữ liệu từ các nguồn: Google image, Bing, Instagram, Foody. Với số lượng ảnh vô cùng phong phú.

- Với **Bing, Instagram, Foody** do không có tool nên chúng em download ảnh thủ công chỉ cài thêm extension của Google Chrome để có thể down ảnh trên Instagram nhanh hơn. Do download thủ công nên chúng em chủ động trong việc lọc ảnh từ đầu đối với các nguồn này.
- Với **Google Image** chúng em sử dụng đoạn code java script tìm được trên mạng để crawl hết tất cả các url của ảnh search theo món sau đó code một đoạn python để chuyển tập url đấy về file ảnh .jpg. Do crawl tự động nên dữ liệu vô cùng lộn xộn, nên chúng em phải lọc lại một lần nữa theo bộ tiêu chí trên.

Các ảnh sau khi tải về và lọc sẽ được lưu trong các folder như sau:



Sau đó các file ảnh sẽ được đặt lại tên theo format: [Tên folder]_[STT]



2. Số lượng, độ đa dạng:

Sau khi thu thập và label cho tất cả dữ liệu, kết quả thu được tổng cộng **23152** ảnh cho **30** class món ăn, cụ thể như sau:

Class	SL	Class	SL	Class	SL
banh_beo	591	banh_xeo	1173	com_chay	815
banh_bot_loc	720	bap_xao	607	com_tam	942
banh_can	744	bot_chien	675	goi_cuon	856
banh_cuon	1140	bun_bo_hue	1530	mi_quang	884
banh_gio	641	bun_cha	510	nem_chua	542
banh_khot	835	bun_dau_mam_tom	927	pha_lau	686
banh_mi	1336	bun_thit_nuong	747	pho	807
banh_mi_nuong	423	cao_lau	618	sup_cua	687
banh_trang_nuong	795	chao_long	1073	xoai_lac	541
banh_trang_tron	494	chuoi_chien	619	xoi_gac	220

Nhận xét:

- Do dữ liệu thu thập từ Internet nên số lượng giữa các class không cân nhau nhưng đã phân giao động trong khoảng từ 700 – 1000 ảnh.
- Các ảnh được chụp từ nhiều góc độ và điều kiện ánh sáng cũng như độ phân giải khác nhau gây khó khăn cho quá trình huấn luyện
- Class xoi_gac số lượng khá ít nên chúng em sẽ dùng những phương pháp làm tăng số lượng lên.
- Bên cạnh các món rất giống nhau khó nhận biết kể trên, còn có nhiều món do góc chụp, độ sáng, cách trình bày khiến chúng trông gần giống nhau ảnh hưởng đến quá trình train dữ liệu

3. Phân chia Train/Test/Validation:

Vì chúng em sử dụng nhiều mô hình huấn luyện khác nhau nên bộ dữ liệu sẽ được chia cố định thành hai phần vào 2 folder **Train** và **Test**, với tỉ lệ:

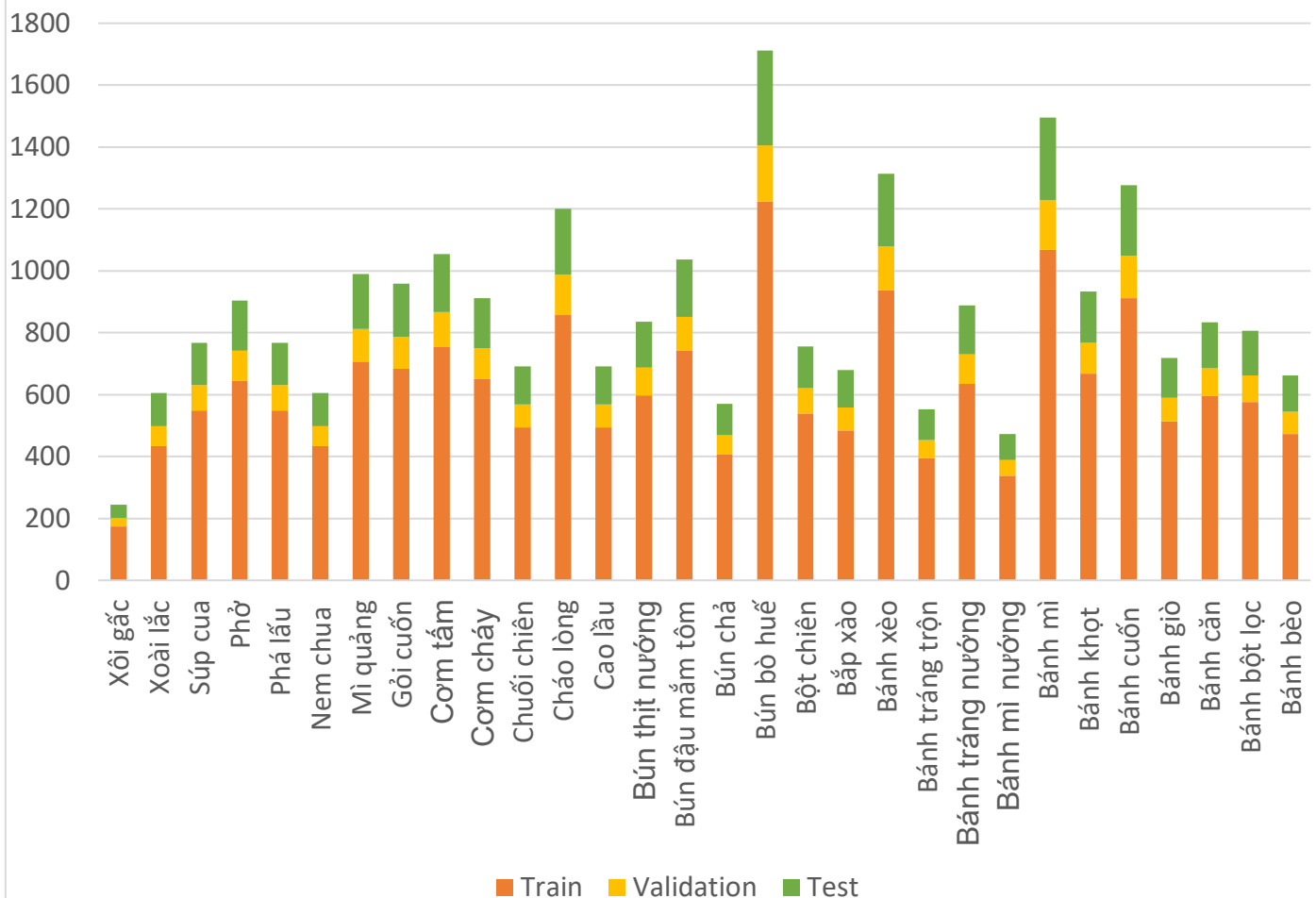
80% Train và 20% Test.

Đối với các kiến trúc deep learning trước khi train chúng em sẽ chia thêm tập **Validation** từ tập Train với tỉ lệ bằng 15% tập Train, khi đó ta được:

68% Train : 12% Validation : 20% Test

	SVM - HOG	Deep learning
Train	18552	15785
Validation		2767
Test	4626	4626

Số lượng ảnh từng label, chia theo train/val/test



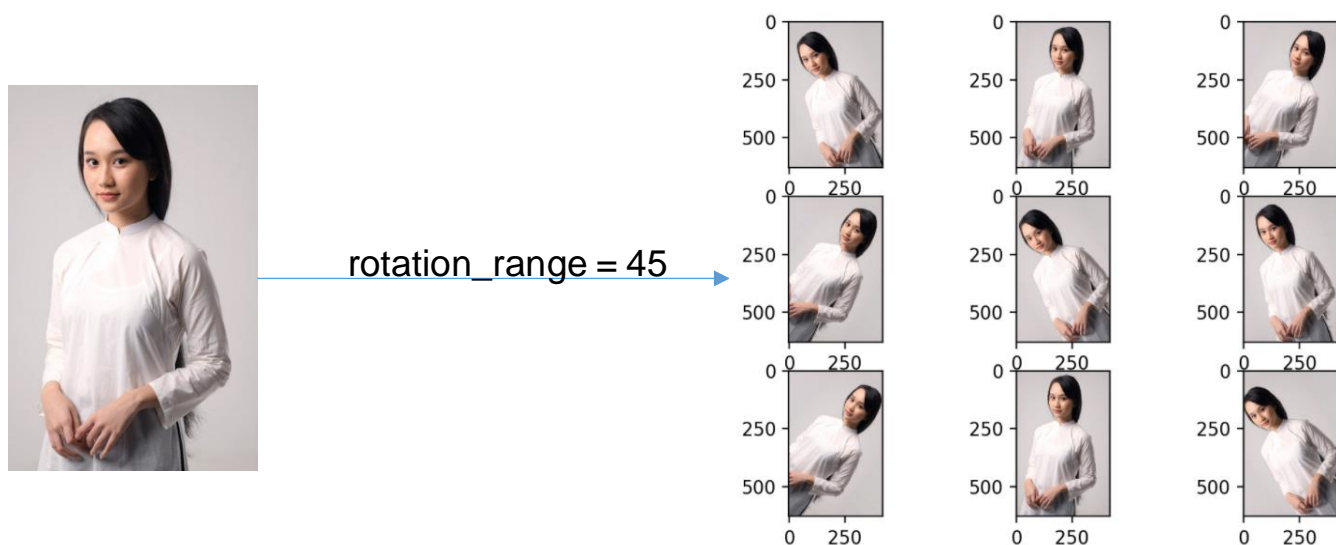
4. Tiền xử lý dữ liệu:

Do số lượng ảnh lớn và không đồng đều về số lượng cũng như kích thước và độ phân giải nên trước khi huấn luyện chúng ta cần phải tiền xử lý và tăng số lượng ảnh. Như đã nói ở trên chúng em sử dụng cả rút trích đặc trưng thủ công với HOG và sử dụng deep learning nên việc tiền xử lý ảnh cũng có sự khác biệt.

- Đối với việc sử dụng HOG chúng em chỉ chuyển kích thước tất cả ảnh về 224x224 sau đó rút trích đặc trưng bằng HOG.
- Với các thuật toán Deep learning ngoài việc resize chúng em còn thực hiện các biện pháp augmentation data để dữ liệu được đa dạng và tránh overfit. Chúng em sử dụng hàm ImageDataGenerator() của Keras để tiền xử lý dữ liệu cụ thể như sau:

```
train_datagen = (shear_range = 0.1,  
                 rotation_range = 90,  
                 brightness_range = [0.7, 1.1],  
                 validation_split = 0.15)
```

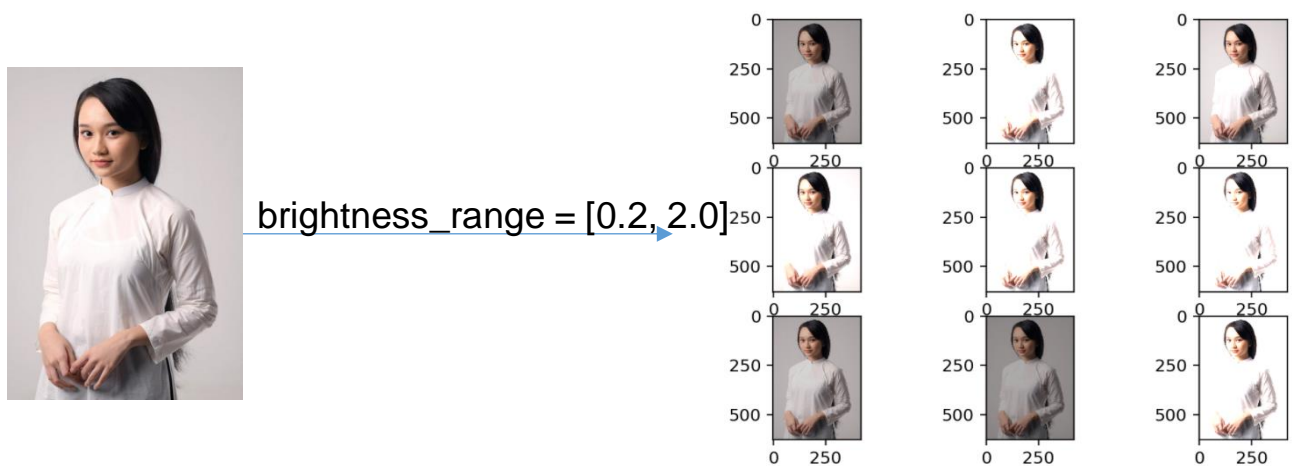
- **rotation_range**: xoay ảnh theo độ.



Chúng em sử dụng xoay với góc 90 độ, để những món có dạng hình trụ hoặc những hình dạng khi xoay sẽ được ảnh khác như bánh mì, chuối chiên...

- **validation_split**: dùng để chia data train và data validation. Như trình bày ở trên chúng em chia tập val = 0,15 tập train.

- **brightness_range**: tăng giảm độ sáng.



Chúng em sử dụng mức tăng giảm độ sáng $[0.7, 1.1]$ để có thể đa dạng hơn với nhiều điều kiện ánh sáng khác nhau tuy nhiên không dao động quá lớn để giữ được đặc trưng cho ảnh.

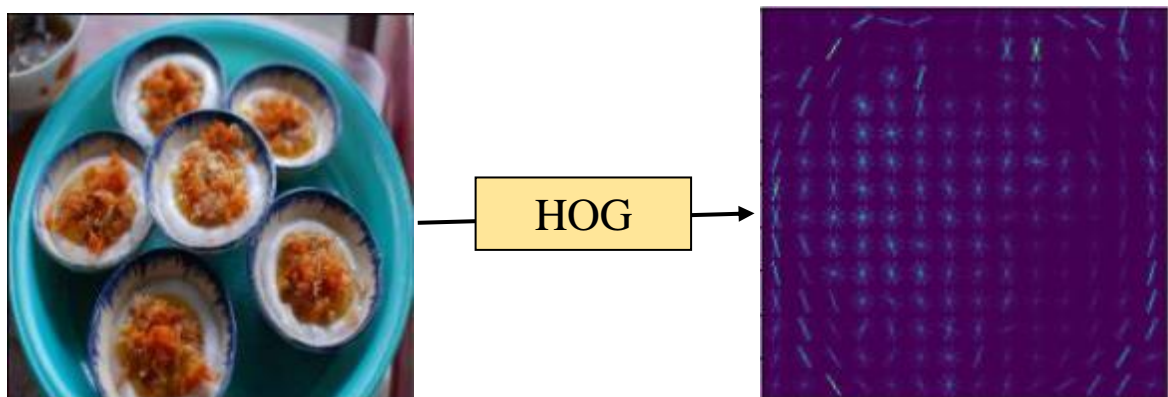
III. Đặc trưng của dữ liệu:

Chúng em sử dụng hai hình thức rút trích đặc trưng dữ liệu là:

- Cách thủ công khác nhau nhưng HOG cho kết quả tốt hơn.
- Các lớp Feature extractor của các mô hình Deep learning.

HOG (Histogram of oriented gradients):

- Mục đích của HOG là trích xuất ra những đặc trưng của ảnh và bỏ đi những chi tiết không hữu ích. Vì vậy HOG được sử dụng chủ yếu để mô tả hình dạng và sự xuất hiện của một đối tượng trong ảnh.
- Bản chất của HOG là sử dụng thông tin về sự phân bố cường độ gradient hoặc hướng biên để mô tả các đối tượng cụ thể trong ảnh.
- HOG chia nhỏ bức ảnh ban đầu thành các cells với mỗi cell trong cells, ta sẽ tính các hướng của gradients cho các điểm ảnh. Sau đó ghép tất cả các hướng này lại với nhau, ta được một biểu diễn cho bức ảnh đưa vào.
- Với ảnh $(224, 224, 3)$ sau khi rút trích theo tính chỉnh của nhóm thì thu được vector đặc trưng có 15488 phần tử

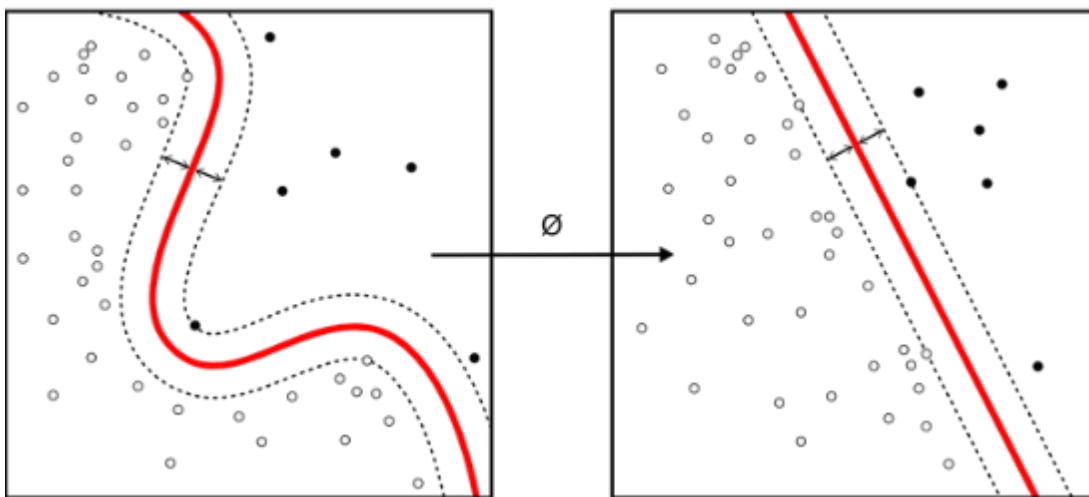


IV. Thuật toán và cài đặt, tinh chỉnh tham số:

1. SVM với rút trích đặc trưng HOG:

a. Lí do chọn SVM với rút trích HOG:

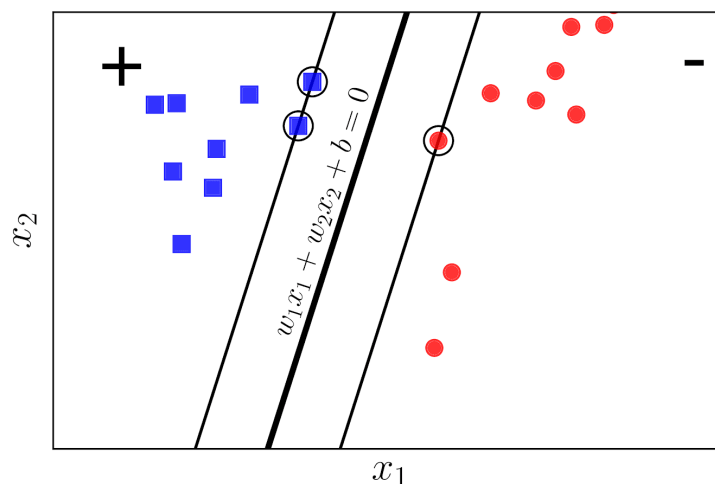
Thuật toán SVM là một trong những thuật toán cơ bản và phổ biến nhất với Machine learning trong giải quyết bài toán phân lớp classification. Với vector đặc trưng được rút trích bằng phương pháp HOG



b. Thuật toán SVM (Support vector machine):

Ý tưởng của SVM là tìm một siêu phẳng (hyper plane) để phân tách các điểm dữ liệu. Siêu phẳng này sẽ chia không gian thành các miền khác nhau và mỗi miền sẽ chứa một loại dữ liệu, sao cho;

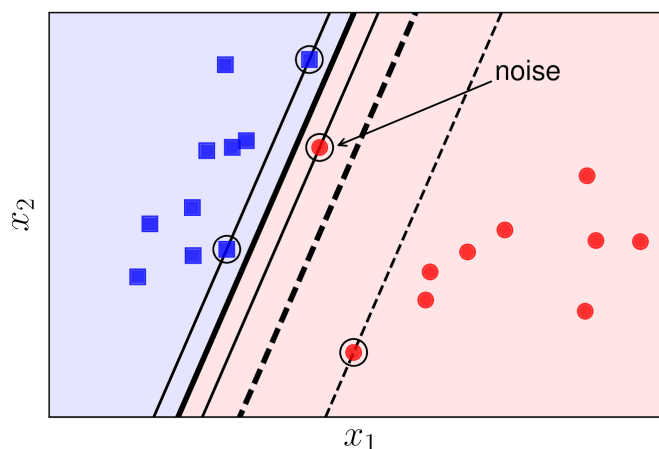
- Khoảng cách từ "siêu mặt phẳng" đến các điểm dữ liệu gần siêu mặt phẳng nhất của các lớp là bằng nhau và gọi là margin
- Giá trị margin là lớn nhất có thể.



Trong ví dụ trên, phương trình: $w_1x_1 + w_2x_2 + b = 0$ chính là một “siêu mặt phẳng” phân chia hai lớp dữ liệu với nhau.

Tuy nhiên, định nghĩa trên được định nghĩa cho **Hard margin support vector machine**. Trong thực tế, người ta thường sử dụng **Soft margin support vector machine** hơn vì dữ liệu thực tế ít khi nào được phân lớp rõ ràng như ví dụ trên.

Ví dụ:

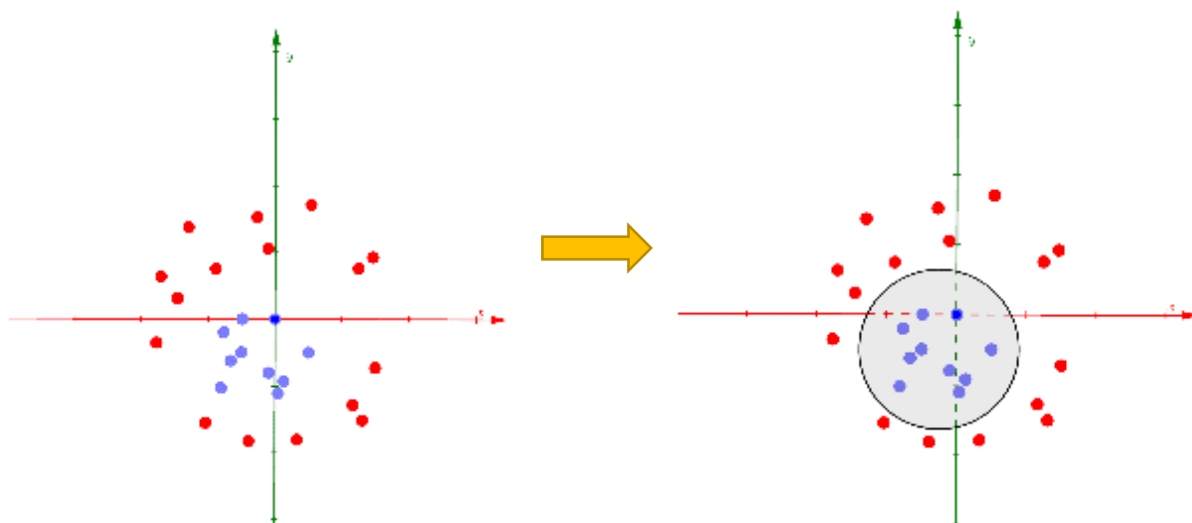


Ở ví dụ trên, nếu ta phân lớp theo **Hard margin support vector machine**, thì “siêu mặt phẳng” ta tìm được chính là đường thẳng đen đậm như trên. Tuy nhiên, nếu nhìn tổng quát dễ dàng thấy được đây không phải là “siêu mặt phẳng” tốt nhất, vì nó lệch về các điểm dữ liệu màu xanh nhiều hơn. Thay vào đó, đường màu đen nét đứt có vẻ là một “siêu mặt phẳng” tối ưu hơn.

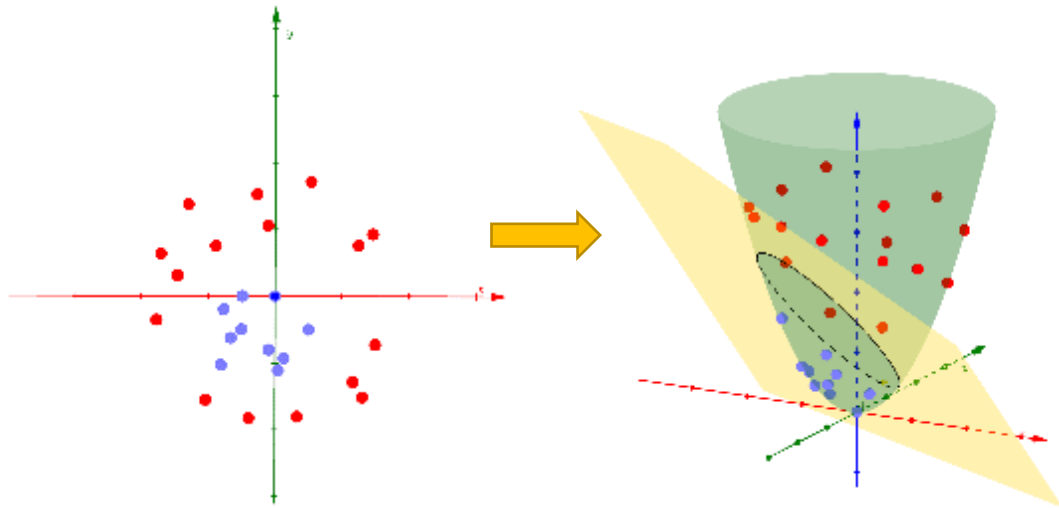
Theo đó, đường màu đen nét đứt chính là kết quả của thuật toán **Soft margin SVM**. Thay vì phân lớp một cách “cứng nhắc”, thuật toán này phân lớp một cách “mềm dẻo” hơn. Đối với những điểm dữ liệu gây nhiễu, không quá quan trọng như điểm “noise” màu đỏ trong hình, thuật toán sẽ bỏ qua nó và xét các điểm dữ liệu khác mang tính xu hướng của lớp đó hơn.

Kernel: Tuy nhiên, đối với trường hợp các điểm dữ liệu không phân biệt tuyến tính, nếu chỉ sử dụng SVM đơn thuần, “siêu mặt phẳng” có thể không được tìm thấy, hoặc nếu tìm được thì sẽ gây ra trường hợp overfitting.

Ví dụ:



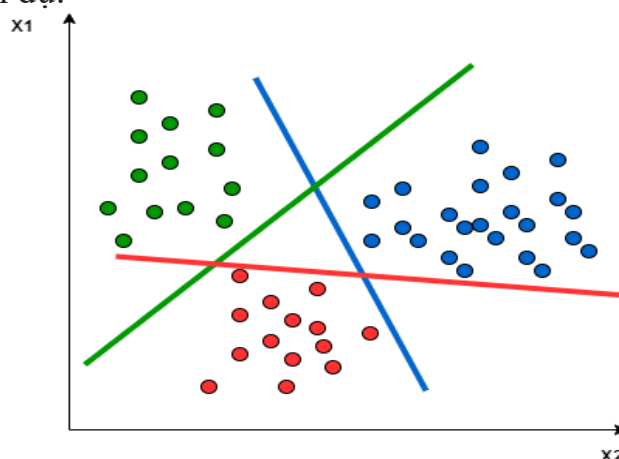
- Trong không gian hai chiều, các điểm dữ liệu trên không phân biệt tuyến tính, nếu chỉ sử dụng SVM đơn thuần phân lớp, ta sẽ được một “siêu mặt phẳng” phân biệt hai lớp như hình bên phải.
- Dễ dàng thấy được “siêu mặt phẳng” hiện tại đã gây ra tình trạng overfitting.
- Vậy để giải quyết trường hợp này, ta cần sử dụng hàm Kernel trong SVM để có thể phân lớp tối ưu hơn. Nói cách đơn giản, ta sẽ tăng chiều của dữ liệu lên, và sau đó phân lớp bằng SVM như sau:



- Như vậy, từ không gian hai chiều, ta đã biến đổi dữ liệu ban đầu lên thành ba chiều và mặt phẳng màu vàng chính là một “siêu mặt phẳng” phân hai lớp dữ liệu trên, và “siêu mặt phẳng” này tối ưu hơn.
- Một số hàm kernel thường được sử dụng trong SVM như: linear, polynomial, radial basic function, sigmoid.

Multiclass Classification:

- SVM đơn thuần thực tế chỉ hỗ trợ phân loại nhị phân, tức là phân tách hai lớp khác nhau., cách tiếp cận này gọi là **One-to-One**
- Trong trường hợp lớp dữ liệu nhiều hơn 2, ta sử dụng một cách tiếp cận khác gọi là **One-to-Rest**. Đối với cách tiếp cận này, khi xét một lớp dữ liệu, ta sẽ xem tất cả các lớp còn lại là một lớp chung và thực hiện phân loại nhị phân đối với hai lớp này. Sau đó tiếp tục xét các lớp còn lại.
- Ví dụ:



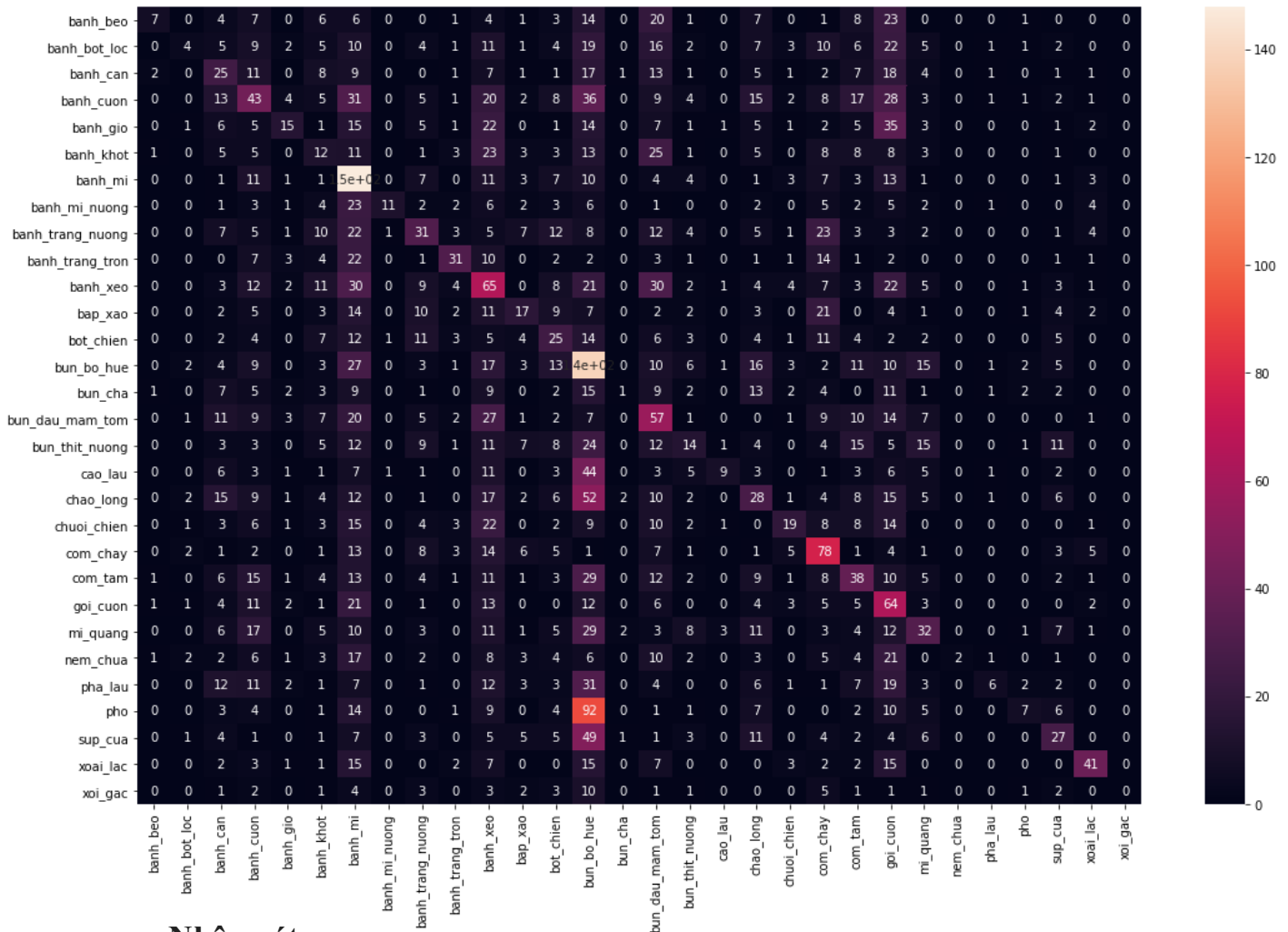
Trong ví dụ trên. Với mỗi lớp dữ liệu màu xanh dương, xanh lá, đỏ, ta sẽ tìm được các đường thẳng phân chia các lớp này với các lớp còn lại với màu tương ứng.

c. Nhận xét đánh giá:

Accuracy:

- Accuracy train: 0.7589796138496386
- Accuracy test: 0.2148403796376186

Confusion matrix:



Nhận xét:

- Dựa vào accuracy train và accuracy test, dễ dàng nhận thấy được mô hình đã gặp phải tình trạng overfitting nặng.
- Dựa vào confusion matrix và accuracy test, ta thấy model dự đoán sai rất nhiều, với kết quả rất thấp này ta không thể dùng được phương pháp này.
- Kết quả kém này có thể do chất lượng hình ảnh không đồng đều, lượng dữ liệu quá đa dạng phức tạp HOG trích xuất đặc trưng về hướng của cạnh trong khi các món ăn được bày trí vô cùng đa dạng.
- Nhận thấy rất khó để rút trích đặc trưng thủ công và các thuật toán Machine learning cơ bản để giải quyết vấn đề nên chúng em chuyển sang các kiến trúc Deep learning nổi tiếng khác.

2. VGG16:

a. Lí do chọn VGG16 với thuật toán tối ưu Adam:

Tại sao chọn VGG16:

- VGG16 là một kiến trúc NN nổi tiếng trong computer vision.
- VGG16 đã được train trên tập ImageNet với số lượng ảnh vô cùng lớn, ta có thể sử dụng pretrained model của VGG16 để transfer learning.

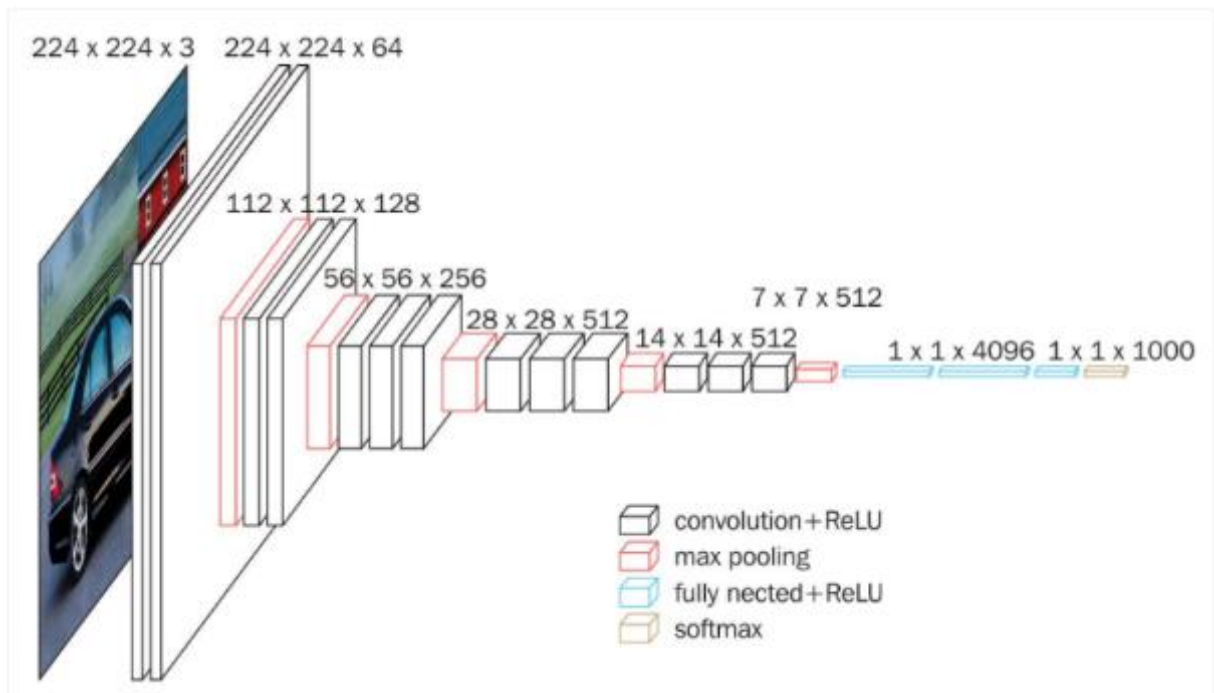
Tại sao lại chọn thuật toán tối ưu Adam:

- Adam chính là sự kết hợp của Momentum và RMSprop, vì vậy nó sẽ tận dụng được những ưu điểm của hai thuật toán.
- Adam giúp model tránh phải tình trạng Overfitting và Local optimizer.

b. Kiến trúc mạng VGG16:

VGG16 là một kiến trúc mạng neural convolutional cổ điển. Nó dựa trên phân tích về cách tăng độ sâu của mạng. VGG16 sử dụng các kernel có kích thước nhỏ 3x3. Ngoài ra, mạng VGG16 được đặc trưng bởi sự đơn giản của nó: Các thành phần khác ngoài các Convolutional là pooling layers và fully connected layers.

Cấu trúc mạng VGG16:

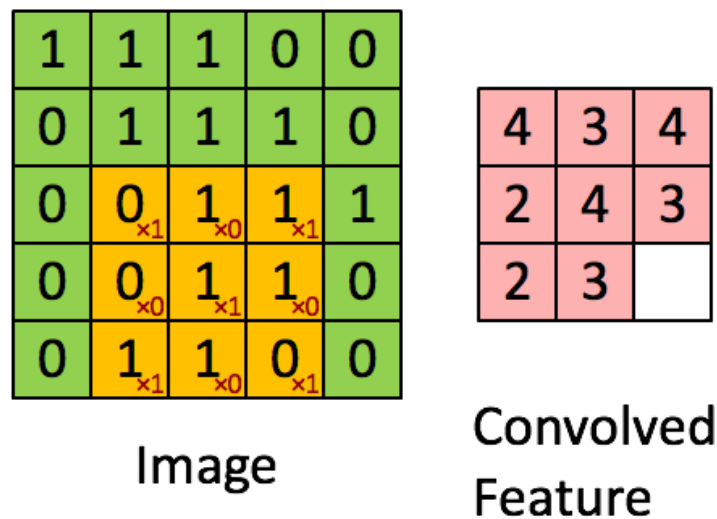


VGG16 Architecture

Như ta thấy, hình ảnh đầu vào của mạng có kích thước được chỉ định là (224,224,3)

Sau đó, hình ảnh ban đầu được chuyển đến một chồng các lớp convolutional với kích thước kernel là 3x3, stride được giữ cố định là 1 và max-pooling với kích thước là 2x2 và stride lúc này mang giá trị là 2

- **Kernel:** là một ma trận dùng để trích xuất đặc trưng như các cạnh từ hình ảnh, trong VGG16 thì ma trận này có kích thước 3x3. Kernel di chuyển trên dữ liệu đầu vào từ trái sang phải, từ trên xuống dưới, nhân các giá trị tương ứng của ma trận đầu vào mà kernel và tổng chúng lại, sau đó đưa qua activation function (ReLU đối với VGG16) và đưa ra output là một ma trận mới là các đặc trưng của đầu vào

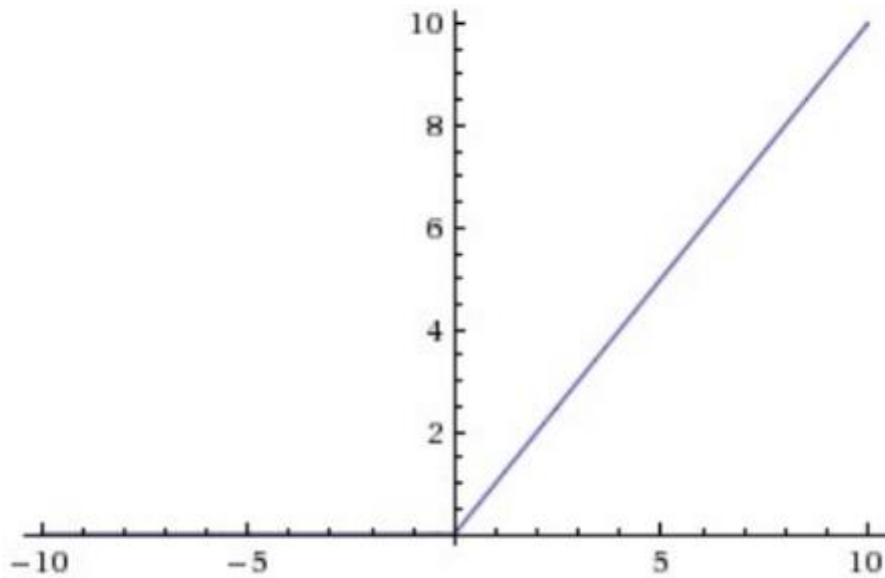


- **Stride** là giá trị mà Kernel di chuyển cách nhau stride pixel trên input đầu vào.
- **Max-pooling:** Khả tương tự như kernel, max-pooling di chuyển trên dữ liệu đầu vào với kích thước nhất định, và chọn ra pixel có giá trị lớn nhất sau đó, tổng hợp chúng thành một ma trận mới



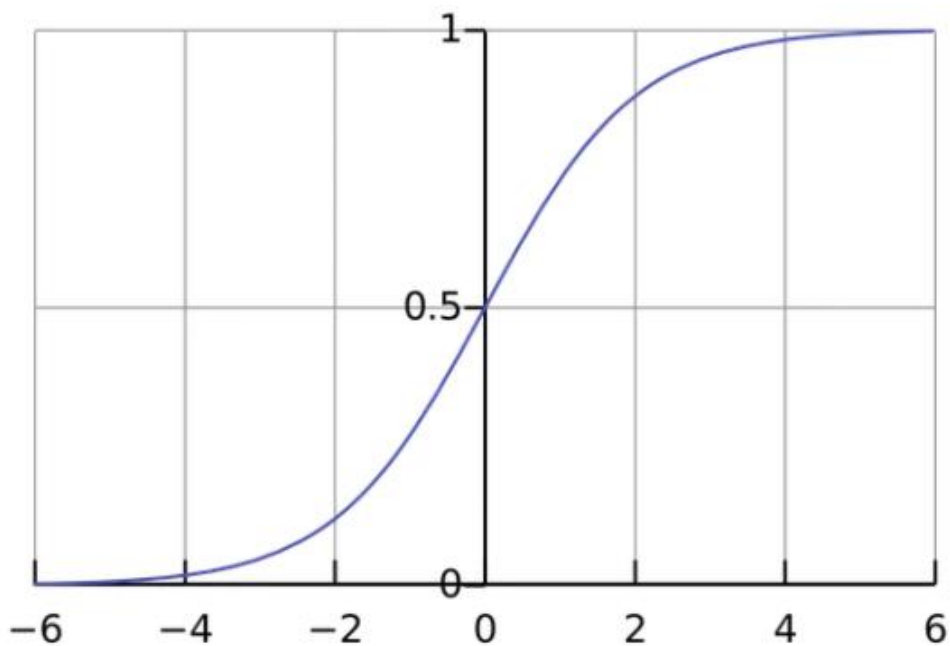
- **ReLU:** là một hàm kích hoạt có công thức:

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



- **Softmax:** là một hàm kích hoạt có công thức:

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)}, \quad \forall i = 1, 2, \dots, C$$



- Cuối cùng, thuật toán tối ưu được sử dụng là **thuật toán Adam**
 - + Adam là sự kết hợp của hai thuật toán: Momentum và RMSprop
 - + Tóm tắt thuật toán có thể biểu diễn như sau:

$$\mathbf{V}_{dw} = \mathbf{0}, \mathbf{S}_{dw} = \mathbf{0}, \mathbf{V}_{db} = \mathbf{0}, \mathbf{S}_{db} = \mathbf{0}$$

Với mỗi lần lặp t , ta có:

Tính \mathbf{dw} , \mathbf{db} ở mini_batch hiện tại.

$$\mathbf{V}_{dw} = \beta_1 * \mathbf{V}_{dw} + (1 - \beta_1) * \mathbf{dw}$$

$$\mathbf{V}_{db} = \beta_1 * \mathbf{V}_{db} + (1 - \beta_1) * \mathbf{db}$$

$$\mathbf{S}_{dw} = \beta_2 * \mathbf{S}_{dw} + (1 - \beta_2) * \mathbf{dw}^2$$

$$\mathbf{S}_{db} = \beta_2 * \mathbf{S}_{db} + (1 - \beta_2) * \mathbf{db}^2$$

$$\mathbf{V}_{dw} = \mathbf{V}_{dw} / (1 - \beta_1^t), \mathbf{V}_{db} = \mathbf{V}_{db} / (1 - \beta_1^t)$$

$$\mathbf{S}_{dw} = \mathbf{S}_{dw} / (1 - \beta_2^t), \mathbf{S}_{db} = \mathbf{S}_{db} / (1 - \beta_2^t)$$

$$\mathbf{W} = \mathbf{W} - \alpha *$$

$$\mathbf{b} = \mathbf{b} - \alpha *$$

Với các siêu tham số:

α : learning_rate

β_1 : thường được chọn là 0.9

β_2 : thường được chọn là 0.999

ϵ : siêu tham số này không quan trọng, chỉ để tránh chia cho 0, thường được chọn là 10^{-8}

c. Tinh chỉnh tham số:

Ban đầu, model sử dụng thuật toán Adam với learning_rate = 0.001. Tuy nhiên, learning_rate này quá lớn, khiến cho accuracy giảm và loss tăng qua mỗi epoch.

Sau đó, model tiếp tục sử dụng thuật toán Adam với learning_rate = 0.0001. Với learning_rate này, model chạy tốt, qua từng epoch, accuracy tăng và loss giảm. Tuy nhiên, model lại gặp vấn đề overfitting

Khi gặp vấn đề overfitting, model thêm kỹ thuật Dropout ở hai lớp FC đầu tiên. Vấn đề overfitting được cải thiện, tuy nhiên lại không đáng kể.

d. Đánh giá:

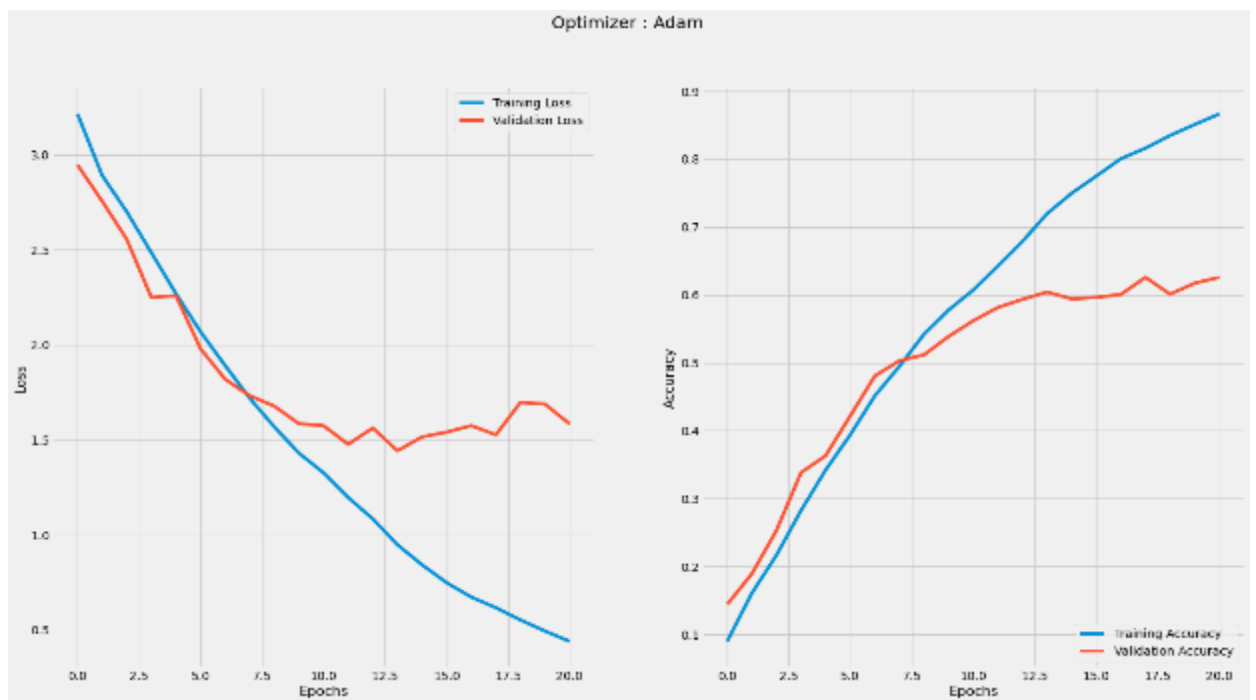
- Train, Test, Validation accuracy:

✓ Train accuracy: 0.8729806542396545

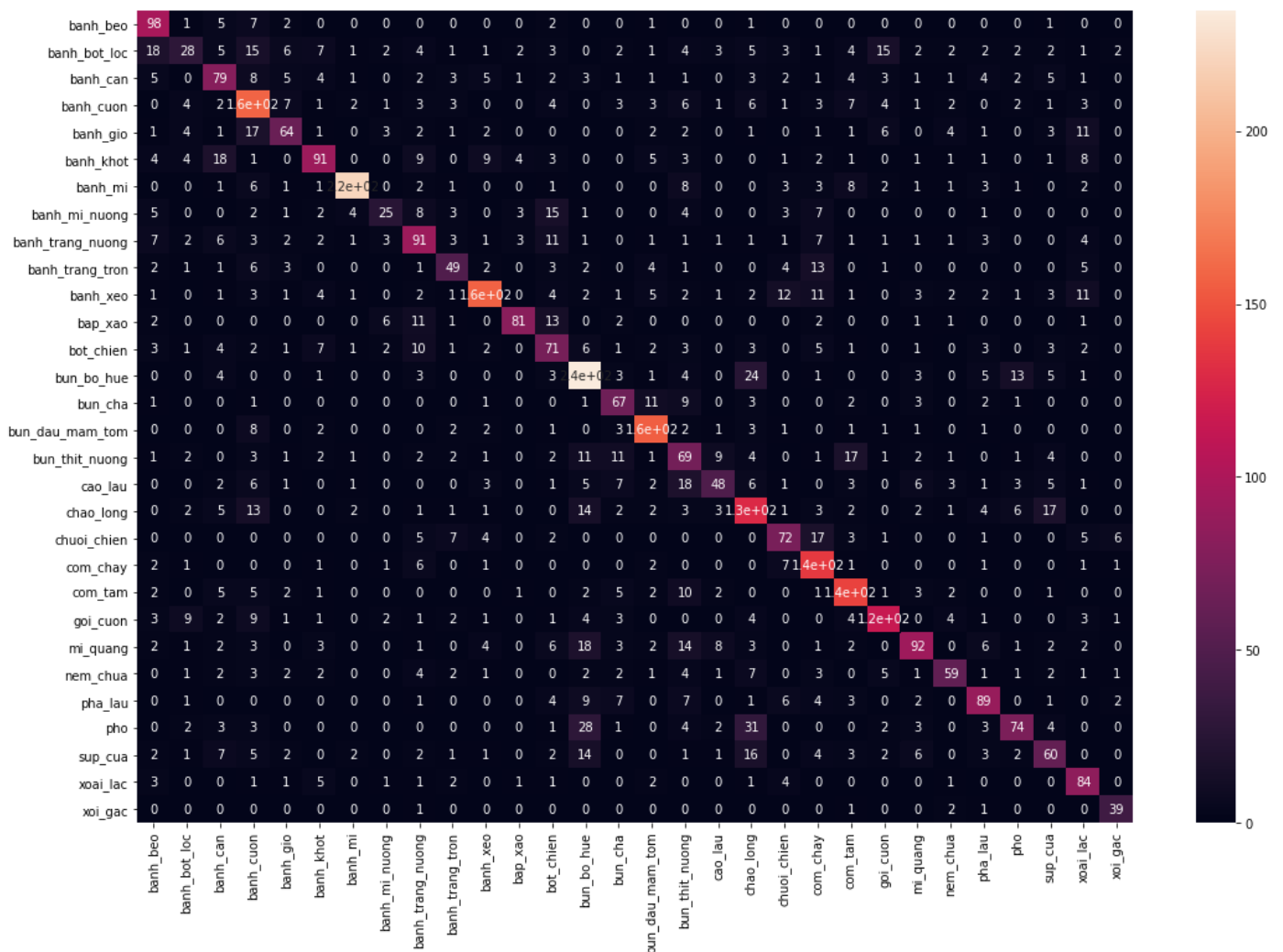
✓ Valid accuracy: 0.6208890676498413

✓ Test accuracy: 0.6232166290283203

- **Biểu đồ accuracy và loss của train và validation:**



- **Confusion matrix:**



- **Classification report:**

	precision	recall	f1-score	support
banh_beo	0.60	0.83	0.70	118
banh_bot_loc	0.43	0.19	0.27	144
banh_can	0.51	0.53	0.52	148
banh_cuon	0.55	0.69	0.61	228
banh_gio	0.62	0.50	0.55	128
banh_khot	0.66	0.54	0.60	167
banh_mi	0.93	0.83	0.88	267
banh_mi_nuong	0.54	0.30	0.38	84
banh_trang_nuong	0.53	0.57	0.55	159
banh_trang_tron	0.57	0.50	0.53	98
banh_xeo	0.79	0.67	0.73	234
bap_xao	0.84	0.67	0.75	121
bot_chien	0.46	0.53	0.49	135
bun_bo_hue	0.66	0.77	0.71	306
bun_cha	0.54	0.66	0.59	102
bun_dau_mam_tom	0.75	0.84	0.80	185
bun_thit_nuong	0.38	0.46	0.42	149
cao_lau	0.59	0.39	0.47	123
chao_long	0.51	0.60	0.55	214
chuoi_chien	0.59	0.59	0.59	123
com_chay	0.60	0.85	0.70	163
com_tam	0.67	0.76	0.71	188
goi_cuon	0.72	0.67	0.69	171
mi_quang	0.68	0.52	0.59	176
nem_chua	0.67	0.55	0.60	108
pha_lau	0.64	0.65	0.64	137
pho	0.67	0.46	0.55	161
sup_cua	0.50	0.44	0.47	137
xoai_lac	0.57	0.78	0.66	108
xoi_gac	0.75	0.89	0.81	44
accuracy			0.62	4626
macro avg	0.62	0.61	0.60	4626
weighted avg	0.63	0.62	0.62	4626

Nhận xét:

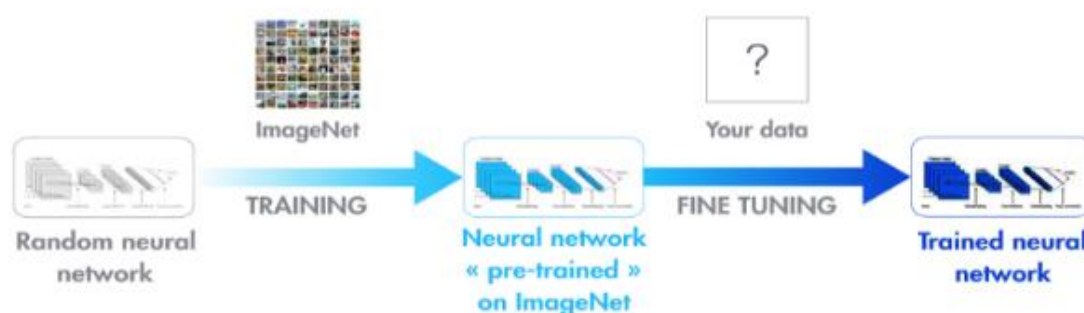
- **Trước tiên xét về accuracy** tổng quan sau quá trình train model ta thấy được rõ vấn đề overfit khi tập train có accuracy cao hơn hẳn so với tập validation và test, cụ thể qua các epoch:
 - ✓ Khi epoch chạy từ 0 đến 10, accuracy của tập train và valid đều tăng một cách đồng đều. Tương tự, loss của tập train và valid đều giảm một cách đồng đều với nhau
 - ✓ Tuy nhiên, khi bắt đầu từ epoch thứ 10 trở đi, mô hình bắt đầu gặp tình trạng overfitting. Diễn hình là khi ở epoch 20, acc_train ~ 0.87, loss_train ~ 0.4 acc_valid ~ 0.63, loss_valid ~ 1.6.

- **Xét về Confusion matrix và Classification report:** phần đường chéo của ma trận đậm hơn các phần còn lại tức lượng dự đoán đúng khá cao, tuy nhiên vẫn còn nhiều điểm dữ liệu ngoài đường chéo vẫn đậm màu tức dự đoán sai là khá nhiều, ví dụ:
 - ✓ **Bánh khọt với Bánh căn.**
 - ✓ **Phở, Mì Quảng với Bún bò Huế, Cháo lòng, Súp cua.**
 - ✓ **Bánh mì nướng với Bột chiên.**
 - ✓ **Bánh bột lọc với Gỏi cuốn, Bánh cuốn.**
 - ✓ **Bánh xèo với Cơm cháy.**
- **Tổng Kết:**
 - ✓ Với bộ dataset của nhóm thì việc train model VGG16 từ đầu là vô cùng mất thời gian và hiệu quả không cao, bên cạnh đó dù đã sử dụng nhiều biện pháp tinh chỉnh tuy nhiên hiện tượng overfit vẫn xảy ra.
 - ✓ Vì thế nhóm quyết định sẽ tiến hành Transfer learning với pretrained model VGG16 với weight thu được từ tập dữ liệu ImageNet.

3. VGG16 Transfer learning:

a. Transfer learning và Fine tuning;

Transfer learning: Là việc áp dụng những gì có sẵn, những tri thức có sẵn áp dụng vào model hiện tại. Thay vì phải train model lại từ đầu, cập nhật các trọng số lại từ đầu, ta có thể sử dụng kết quả của model khác đã train và có kết quả tốt, trong trường hợp này, ta sử dụng trong số của model có sẵn ImageNet và áp dụng train tiếp tục với data của mình và thêm, bớt một số layer mới.



Transfer learning – Fine tuning

Fine tuning: là một kiểu Transfer learning, khi lấy 1 pre-trained model, tận dụng 1 phần hoặc toàn bộ các layer, thêm/sửa/xoá 1 vài layer/nhánh để tạo ra 1 model mới. Thường các layer đầu của model được freeze (đóng băng) lại - tức weight các layer này sẽ không bị thay đổi giá trị trong quá trình train. Lý do bởi các layer này đã có khả năng trích xuất thông tin mức trừu tượng thấp, khả năng này được học từ quá trình training trước đó. Ta freeze lại để tận dụng được khả năng này và giúp việc train diễn ra nhanh hơn.

b. Thiết kế lại Pretrained model:

Để giảm overfit và tăng thời gian train model chúng em quyết định Transfer learning pretrained VGG16 model để tận dụng các trọng số đã train trên tập ImageNet.

Đầu tiên chúng em sẽ đồng bằng **n** layer top của pre-trained model.

Tiếp theo chúng em thiết kế lại các layer cuối:

- Thêm layer **flatten** để dằn phẳng dữ liệu, nối vào các lớp fully connect.
- Thay vì mô hình VGG16 thường dùng, ở các layer **FC** cuối, sẽ có thứ tự **FC4096 -> FC4096 -> FC30** thì ở model này, ta thay thế bởi **FC4096 -> FC1072 -> FC30**
- Thêm layer **Dropout** để bỏ qua ngẫu nhiên vài unit khi train giảm overfit
- Layer **Dense** cuối cùng chúng em sẽ dùng hàm kích hoạt **Softmax**.

flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102764544
dense_1 (Dense)	(None, 1072)	4391984
dropout (Dropout)	(None, 1072)	0
tf.convert_to_tensor (TFOpLa	(None, 1072)	0
tf.math.subtract (TFOpLambda	(None, 1072)	0
tf.math.truediv (TFOpLambda)	(None, 1072)	0
dense_2 (Dense)	(None, 30)	32190

c. Tinh chỉnh tham số:

Ban đầu, model được đặt giá trị **fine-tune = 8**, có nghĩa là 10 layers đầu của pre_trained model có trọng số không thay đổi trong quá trình train. Tuy nhiên, accuracy của model lại giảm mạnh, loss cũng tăng lên nhanh chóng. Nên model tạm dừng train để thay đổi các tham số khác

Tiếp theo, model được đặt lại giá trị **fine-tune = 4**. Tuy nhiên model lại gặp vấn đề tương tự, accuracy giảm và loss tăng theo thời gian

Sau đó, model được đặt lại giá trị **fine-tune = 2**. Ở giá trị này, model chạy ổn định, accuracy có xu hướng tăng và loss cũng có xu hướng giảm. Dù accuracy của bộ dữ liệu valid cao hơn so với model VGG16 cũ, tuy nhiên vẫn chưa cao hơn đáng kể

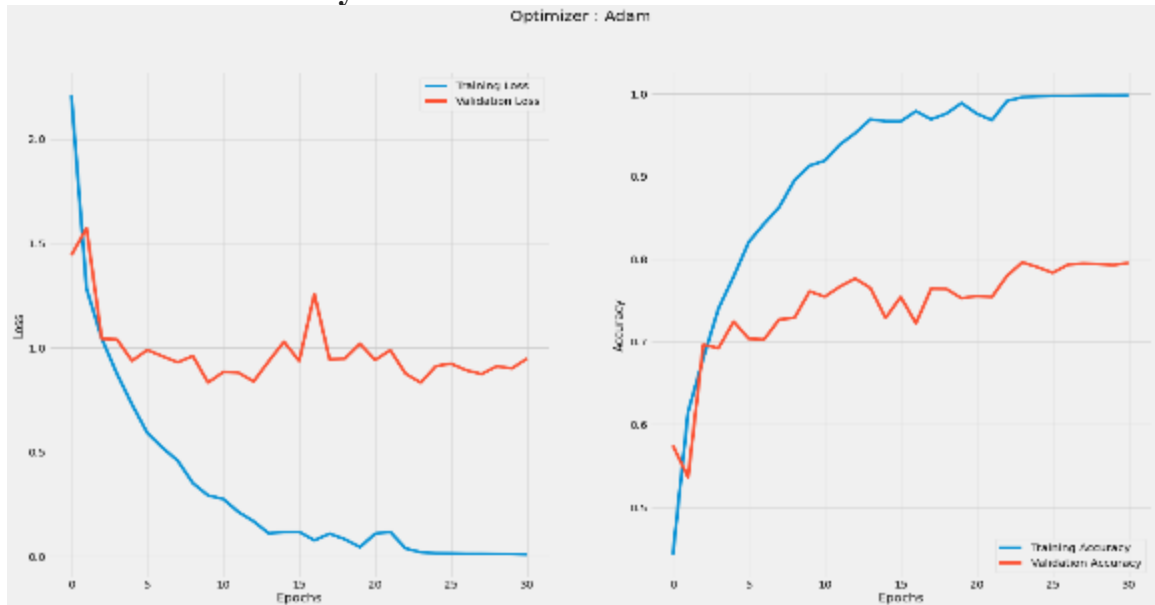
Cuối cùng, để tạo thành model như hiện tại, ta thêm normalization ở sau layer trước khi đưa vào hàm softmax, và kết quả là accuracy của bộ dữ liệu valid đã tăng đáng kể

d. Đánh giá:

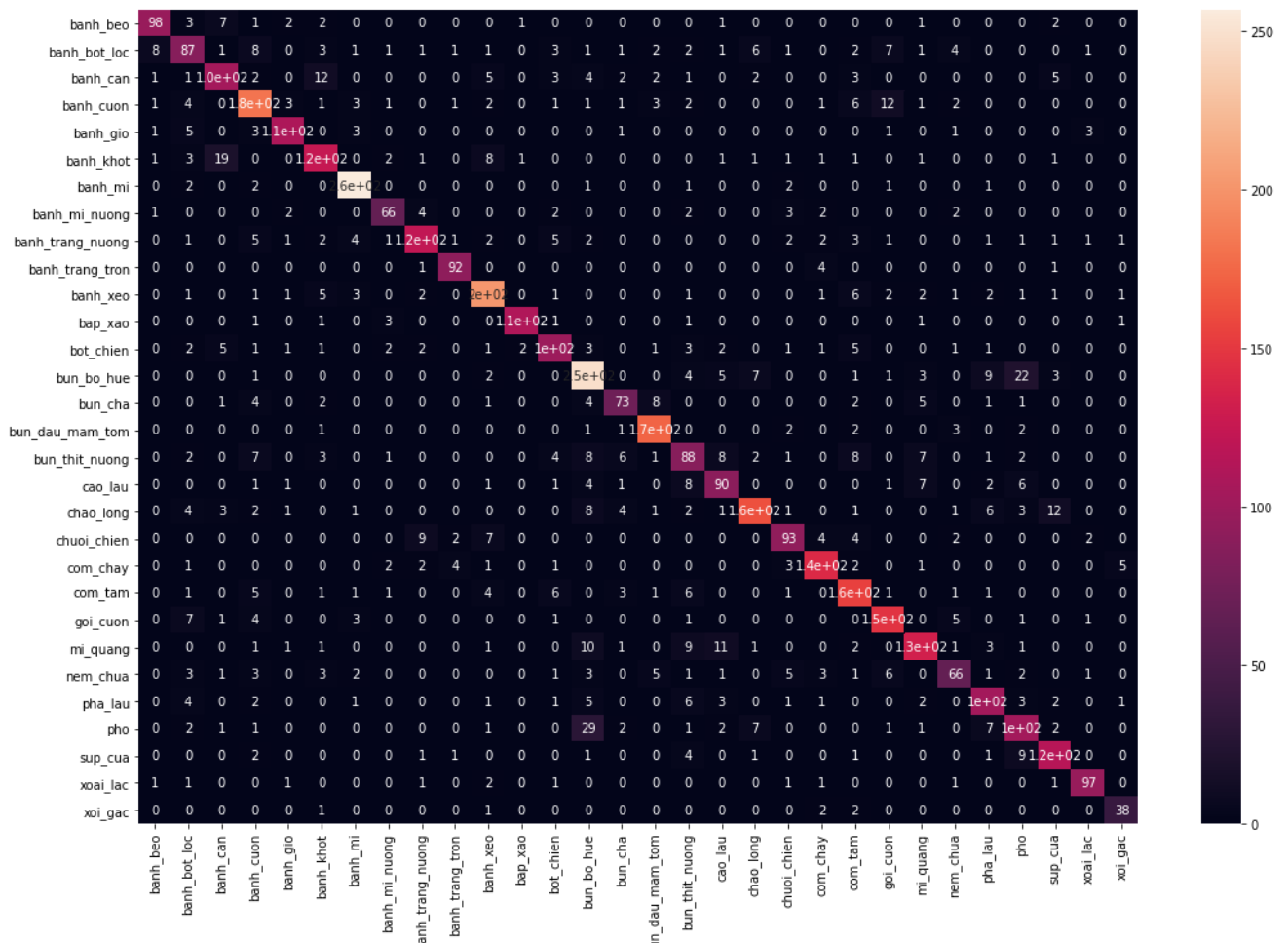
- Train, test, validation accuracy:

- ✓ Train accuracy: 0.9978460669517517
- ✓ Valid accuracy: 0.7864112854003906
- ✓ Test accuracy: 0.7970168590545654

- Biểu đồ accuracy và loss của train và validation:



- Confusion matrix:



- **Classification**

report:

	precision	recall	f1-score	support
banh_beo	0.88	0.83	0.85	118
banh_bot_loc	0.65	0.60	0.63	144
banh_can	0.73	0.71	0.72	148
banh_cuon	0.76	0.80	0.78	228
banh_gio	0.89	0.86	0.87	128
banh_khot	0.76	0.75	0.76	167
banh_mi	0.92	0.96	0.94	267
banh_mi_nuong	0.82	0.79	0.80	84
banh_trang_nuong	0.84	0.77	0.80	159
banh_trang_tron	0.90	0.94	0.92	98
banh_xeo	0.83	0.86	0.85	234
bap_xao	0.97	0.93	0.95	121
bot_chien	0.76	0.74	0.75	135
bun_bo_hue	0.74	0.81	0.78	306
bun_cha	0.76	0.72	0.74	102
bun_dau_mam_tom	0.88	0.94	0.91	185
bun_thit_nuong	0.62	0.59	0.60	149
cao_lau	0.71	0.73	0.72	123
chao_long	0.86	0.76	0.81	214
chuoai_chien	0.79	0.76	0.77	123
com_chay	0.86	0.87	0.86	163
com_tam	0.75	0.82	0.78	188
goi_cuon	0.81	0.86	0.84	171
mi_quang	0.80	0.76	0.78	176
nem_chua	0.73	0.61	0.66	108
pha_lau	0.74	0.76	0.75	137
pho	0.66	0.65	0.65	161
sup_cua	0.79	0.85	0.82	137
xoi_lac	0.92	0.90	0.91	108
xoi_gac	0.81	0.86	0.84	44
accuracy			0.80	4626
macro avg	0.80	0.79	0.79	4626
weighted avg	0.80	0.80	0.80	4626

Nhận xét:

- **Về Accuracy:** Dễ dàng nhận thấy được, accuracy của model hiện tại tốt hơn hẳn so với model trước, khi ta áp dụng các phương pháp mới vào việc xây dựng model
 - ✓ Khi model mới này sử dụng thêm các kỹ thuật như transfer learning, fine-tuning, normalization, tốc độ học tập của model đã tăng đáng kể. Cụ thể, từ epoch 0 đến epoch thứ 10, valid accuracy tăng nhanh, loss valid cũng giảm đáng kể.
 - ✓ Về vấn đề overfitting của model VGG16 cũ. Model mới vẫn gặp vấn đề tương tự, tuy nhiên đã cải thiện đáng kể, điển hình là valid accuracy đã có thể tăng lên đến 0.8
- **Về Confusion matrix:** So với model VGG16 trước, model hiện tại đã có sự cải biến và các giá trị dự đoán đã tốt hơn, biểu hiện ở việc đường những giá trị ngoài đường chéo của confusion matrix đã không còn phân bố rải rác các con số mang giá trị lớn nữa. Tuy nhiên model vẫn còn nhận nhầm ở một vài món có sự tương đồngng cao về ngoại hình như:
 - ✓ **Bánh căn** với **Bánh khọt**
 - ✓ **Bánh cuốn** với **Gỏi cuốn**
 - ✓ **Bún bò huế, Cao lầu** với **Phở, Mì Quảng.**
 - ✓ **Cháo lòng** với **Súp cua.**

- **Tổng kết:**

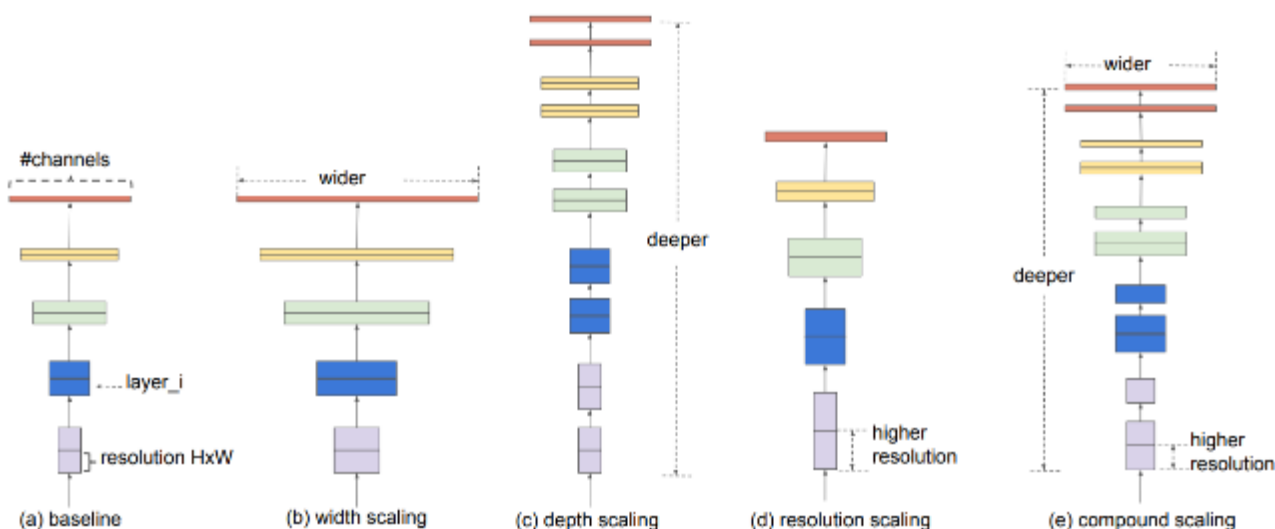
- ✓ So với model VGG16 cũ, model mới đã có sự cải thiện đáng kể về mọi mặt khi áp dụng các kĩ thuật tăng tốc độ học tập cũng như giảm overfitting
- ✓ Tuy nhiên, model này vẫn chưa tối ưu vì accuracy vẫn còn thấp, model vẫn gặp tình trạng overfitting và vẫn xòn bị nhầm lẫn các món ăn có sự tương đồng với nhau về mặt hình ảnh.
- ✓ Để cải thiện chúng em chuyển sang các mô hình mới hơn tối ưu hơn như VGG19 và EfficientNet, tuy nhiên, VGG19 kết quả không cải thiện.

4. EfficientNet Transfer learning:

a. Lí do chọn EfficientNet:

EfficientNet một kiến trúc, một cách tiếp cận mới về **Model scaling** cho CNNs được đề xuất vào năm 2019. CNNs thường được phát triển với ngân sách tài nguyên cố định và sau đó được thu phóng để có độ chính xác tốt hơn nếu có nhiều tài nguyên hơn.

Vi thể nên nhóm tác giả **Mingxing Tan** và **Lê Viết Quốc** (computer scientist người Việt làm việc cho Google) đã nghiên cứu một cách có hệ thống và nhận thấy rằng để đạt được độ chính xác và hiệu quả tốt hơn, điều quan trọng là phải cân bằng tất cả các kích thước của chiều rộng, chiều sâu và độ phân giải mạng trong quá trình thu phóng quy mô ConvNet (**compound scaling**).



Từ đó nhóm tác giả đã phát triển một mạng cơ sở kích thước di động, được gọi là EfficientNet

Chúng em lựa chọn kiến trúc này vì đây là một kiến trúc hiện đại nhất, vô cùng linh động với độ accuracy trên tập dữ liệu ImageNet là rất cao, bên cạnh đó dữ liệu chúng em sử dụng cũng là dữ liệu hình ảnh và có chút tương đồng với tập dữ liệu này.

Chúng em sử dụng 2 phiên bản của EfficientNet là B0 và B4 để làm pre-trained model vì EfficientNet B0 là mô hình cơ sở để thực hiện compound scaling lên các phiên bản cao hơn và B4 là phiên bản cao mà thời train không quá cao như các phiên bản cao hơn, phù hợp cho GPU giới hạn của colab.

b. Kiến trúc mạng EfficientNet:

Phương pháp thu phóng phức hợp (compound scaling):

- Các tác giả đã đề xuất thu phóng phức hợp mô hình bằng cách sử dụng hệ số kép ϕ để thu phóng đồng nhất chiều rộng, độ sâu và độ phân giải của mạng theo cách có nguyên tắc:

$$d = \alpha^\phi$$

$$w = \beta^\phi$$

$$r = \gamma^\phi$$

$$s\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Trong đó:

- **d, w, r** lần lượt là độ rộng, độ sâu và độ phân giải của mạng
- **α, β, γ** là các hằng số có thể được xác định bằng small grid search.

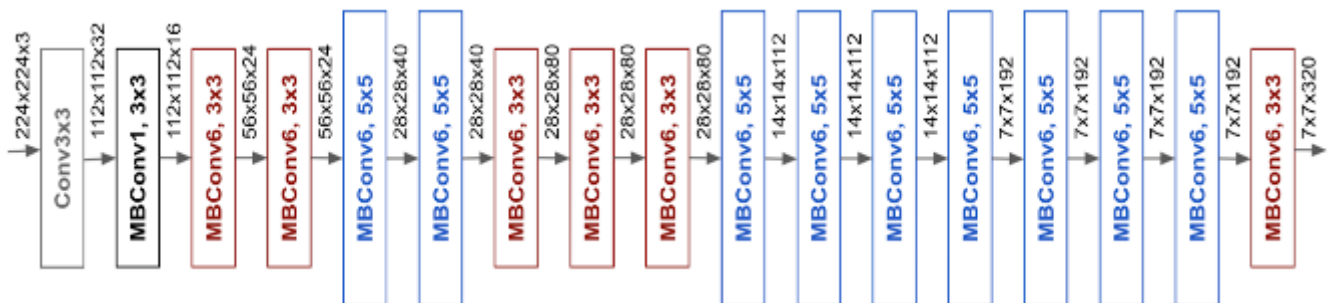
- Theo trực giác, ϕ là hệ số do người dùng chỉ định để kiểm soát số lượng tài nguyên khác có sẵn để thu phóng mô hình, trong khi **α, β, γ** chỉ định cách gán các tài nguyên bổ sung này cho độ rộng, độ sâu và độ phân giải của mạng tương ứng. Đáng chú ý, **FLOPS** của một op tích hợp thông thường tỷ lệ với **d, w^2, r^2** , tức là, độ sâu mạng tăng gấp đôi sẽ tăng gấp đôi FLOPS, nhưng tăng gấp đôi độ rộng hoặc độ phân giải của mạng sẽ tăng FLOPS lên bốn lần. Vì các hoạt động tích phân thường chiếm ưu thế trong chi phí tính toán trong ConvNets, nên việc thu phóng Mạng Conv với phương trình 3 sẽ làm tăng tổng FLOPS khoảng $(\alpha \cdot \beta^2 \cdot \gamma^2)^\phi$. Trong bài báo này, chúng tôi ràng buộc $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ sao cho với bất kỳ new mới nào, tổng FLOPS sẽ tăng xấp xỉ lên 2^ϕ

Kiến trúc mạng EfficientNet:

- Lấy cảm hứng từ các nghiên cứu trước, nhóm tác giả phát triển mạng cơ sở của mình bằng cách tận dụng tìm kiếm kiến trúc no-ron đa mục tiêu để tối ưu hóa cả độ chính xác và FLOPS. Cụ thể, nhóm tác giả sử dụng cùng một không gian với phương pháp được mô tả trong MnasNet: Platform-aware neural architecture search for mobile và sử dụng $ACC(m) \times [FLOPS(m)/T]^w$ làm mục tiêu tối ưu hóa, trong đó $ACC(m)$ và $FLOPS(m)$ biểu thị độ chính xác và FLOPS của mô hình m , T là mức FLOPS mục tiêu và $w = -0,07$ là siêu tham số để kiểm soát sự cân bằng giữa độ chính xác và FLOPS. Tuy nhiên không giống nghiên cứu trên, ở đây nhóm tác giả tối ưu hóa FLOPS thay vì độ trễ vì họ không nhắm mục tiêu bất kỳ thiết bị phần cứng cụ thể nào. Từ đó nhóm tác giả tạo ra một mạng hiệu quả, thứ được đặt tên đặt tên là **EfficientNet-B0**.

Table 1. EfficientNet-B0 baseline network – Each row describes a stage i with \hat{L}_i layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels \hat{C}_i . Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

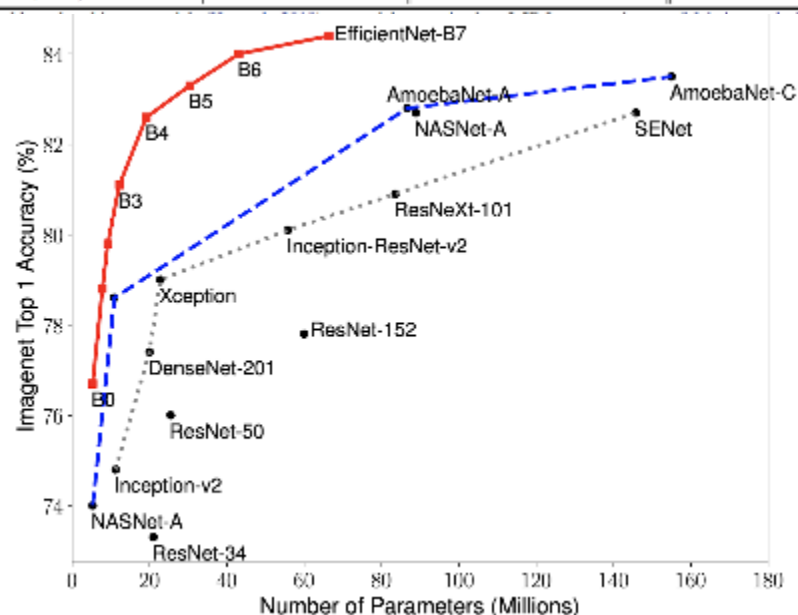


EfficientNet Architecture

- Bắt đầu từ mô hình cơ sở **EfficientNet-B0**, nhóm tác giả áp dụng phương pháp thu phóng phức hợp của mình để thu phóng quy mô với hai bước bao gồm:
 - ✓ **Bước 1:** Đặt cố định giá trị của ϕ bằng 1, nhóm tác giả thu được bộ giá trị tối ưu $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, theo ràng buộc $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$
 - ✓ **Bước 2:** Cố định α, β, γ dưới dạng các hằng số và thu phóng mạng cơ sở với các ϕ khác nhau từ đó thu được từ **EfficientNet-B1** đến **EfficientNet-B7**.

Hiệu suất của EfficientNet: Bảng sau cho thấy hiệu suất của tất cả các mô hình EfficientNet được thu phóng từ cùng một mô hình EfficientNet-B0 với các mô hình nổi tiếng khác khi huấn luyện trên ImageNet. EfficientNet vượt trội hơn hẳn.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	77.1%	93.3%	5.3M	1x	0.39B	1x
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	79.1%	94.4%	7.8M	1x	0.70B	1x
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	80.1%	94.9%	9.2M	1x	1.0B	1x
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	81.6%	95.7%	12M	1x	1.8B	1x
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	82.9%	96.4%	19M	1x	4.2B	1x
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	83.6%	96.7%	30M	1x	9.9B	1x
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	84.0%	96.8%	43M	1x	19B	1x
EfficientNet-B7	84.3%	97.0%	66M	1x	37B	1x
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-



c. EfficientNet B0 – B4 transfer learning và tinh chỉnh tham số:

Đây là mô hình mới và tương đối phức tạp trong quá trình tùy chỉnh để có thể tối ưu hơn nữa chúng em đã train rất nhiều cách fine tuning khác nhau và cuối cùng đây là hai cách fine tuning EfficientNet hiệu quả nhất:

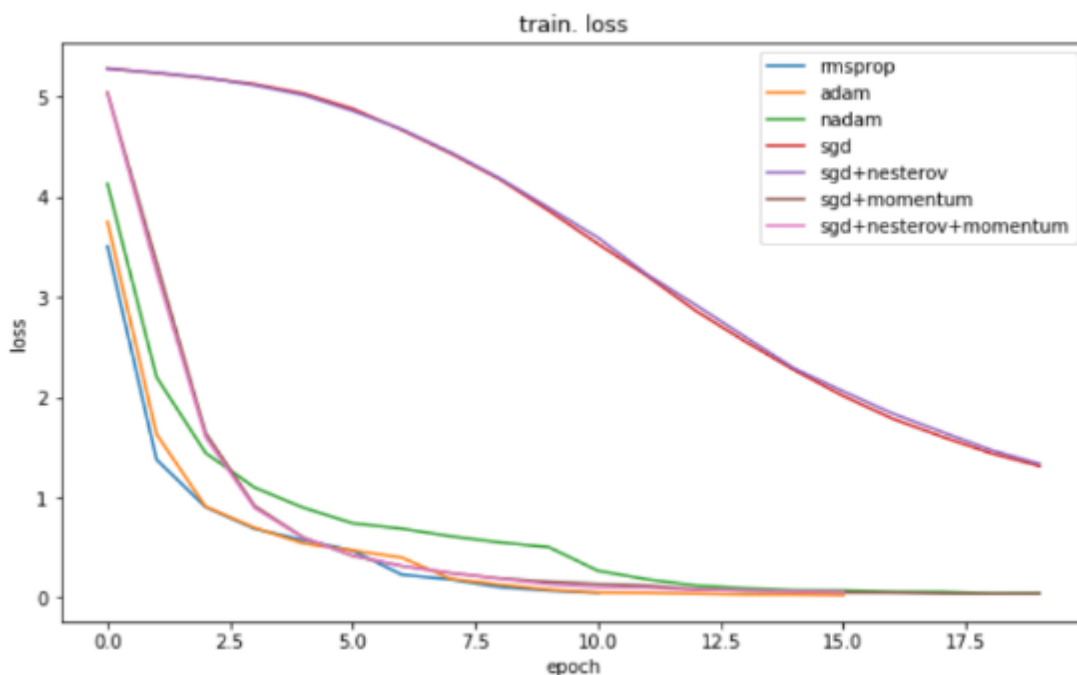
❖ Chiến lược fine tuning 1: Áp dụng cho EfficientNet B0 và B4:

- **Phase 1:** Đóng băng pretrained model và tinh chỉnh các lớp top layer.

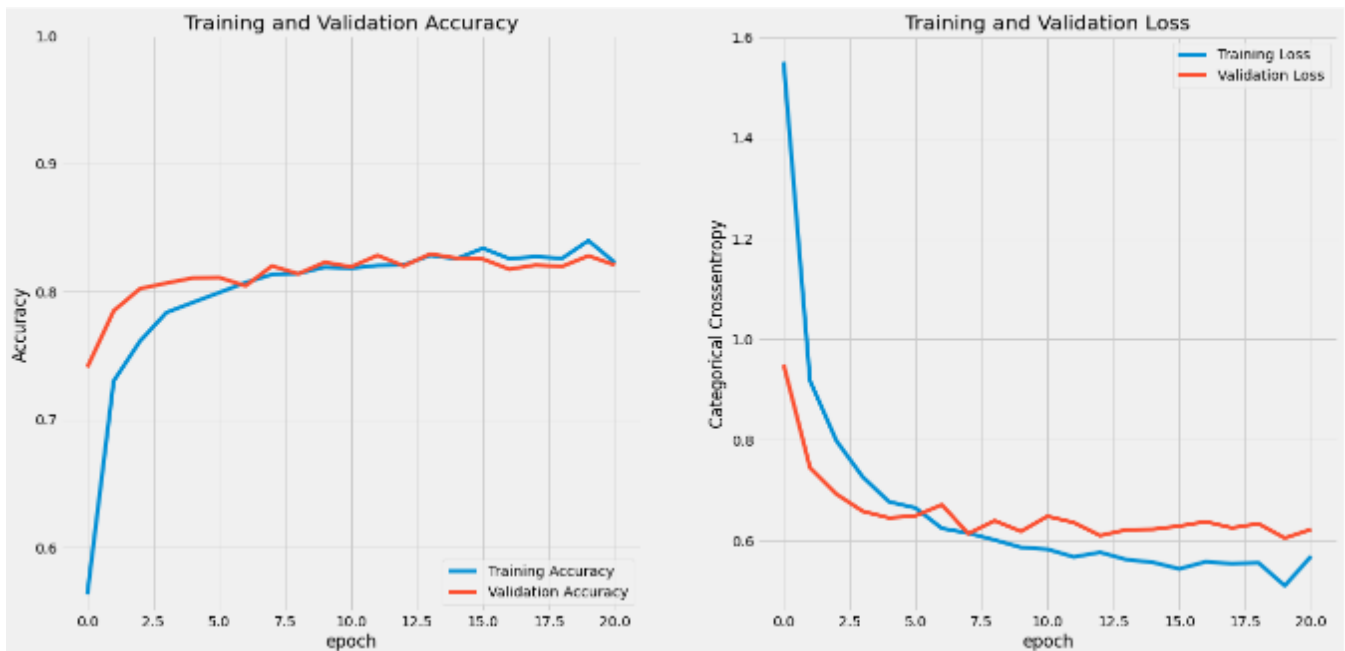
- ✓ Đầu tiên khởi tạo pretrained model với weight = ImageNet và không bao gồm các layer top.
- ✓ Đóng băng tất cả các layer để giữ lại kết quả tốt.
- ✓ Tiến hành tạo các layer top cho model gồm:
 - Thay vì làm dùng **flatten** để dẹt phẳng dữ liệu một cách bình thường chúng em sử dụng **GlobalAveragePooling2D** để có thể giảm số lượng phần tử trong vector mà vẫn giữ được đặc trưng.
 - Tiếp theo là lớp **BatchNormalization** để chuẩn hóa dữ liệu ở các layer theo batch về phân phối chuẩn để quá trình training hội tụ nhanh hơn.
 - Tiếp đến là **Dropout layer** để bỏ qua một vài unit ngẫu nhiên trong quá trình train để **giảm overfitting**.
 - Cuối cùng là **Dense** với hàm kích hoạt **Softmax**.

global_average_pooling2d_1 (Glo (None, 1280))	0	top_activation[0][0]
batch_normalization_2 (BatchNor (None, 1280))	5120	global_average_pooling2d_1[0][0]
top_dropout (Dropout)	(None, 1280)	0
dense_2 (Dense)	(None, 30)	38430
		top_dropout[0][0]

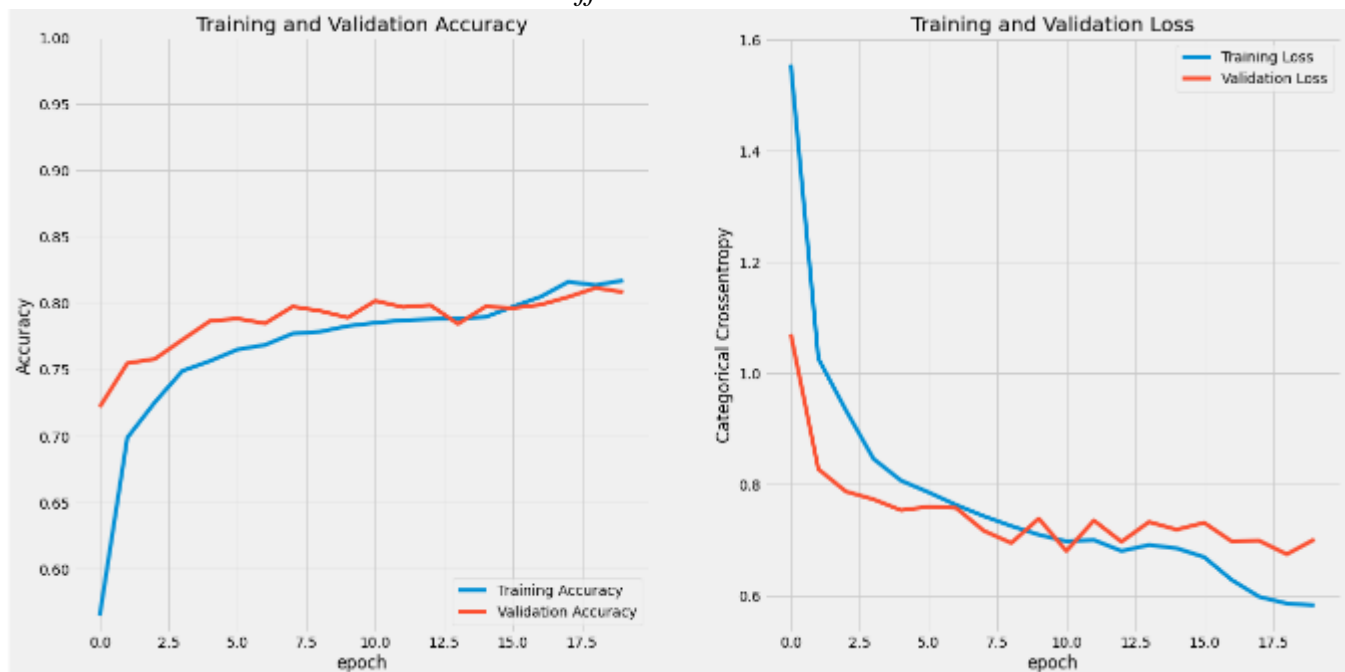
- ✓ Training với 20 epoch và thuật toán tối ưu Adam, vì theo tìm hiểu với EfficientNet thì thuật toán tối ưu tốt nhất là thuật toán Adam.



✓ Kết quả sau khi train:



EfficientNet B0

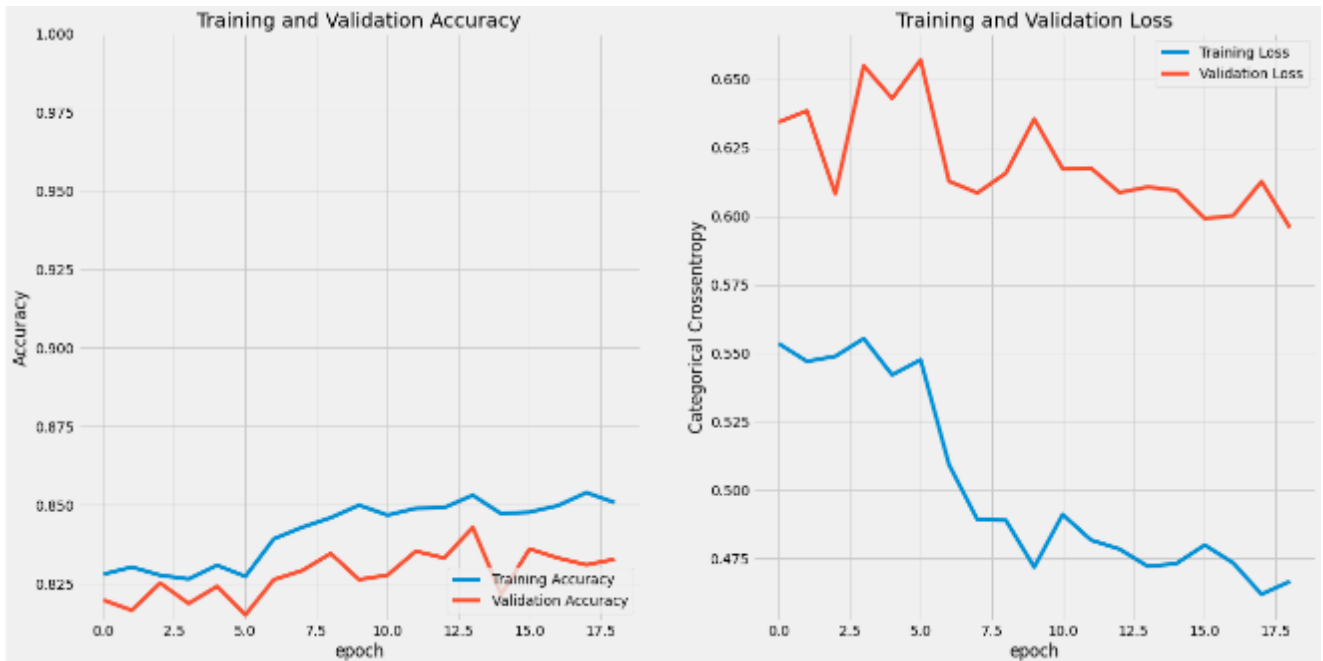


EfficientNet B4

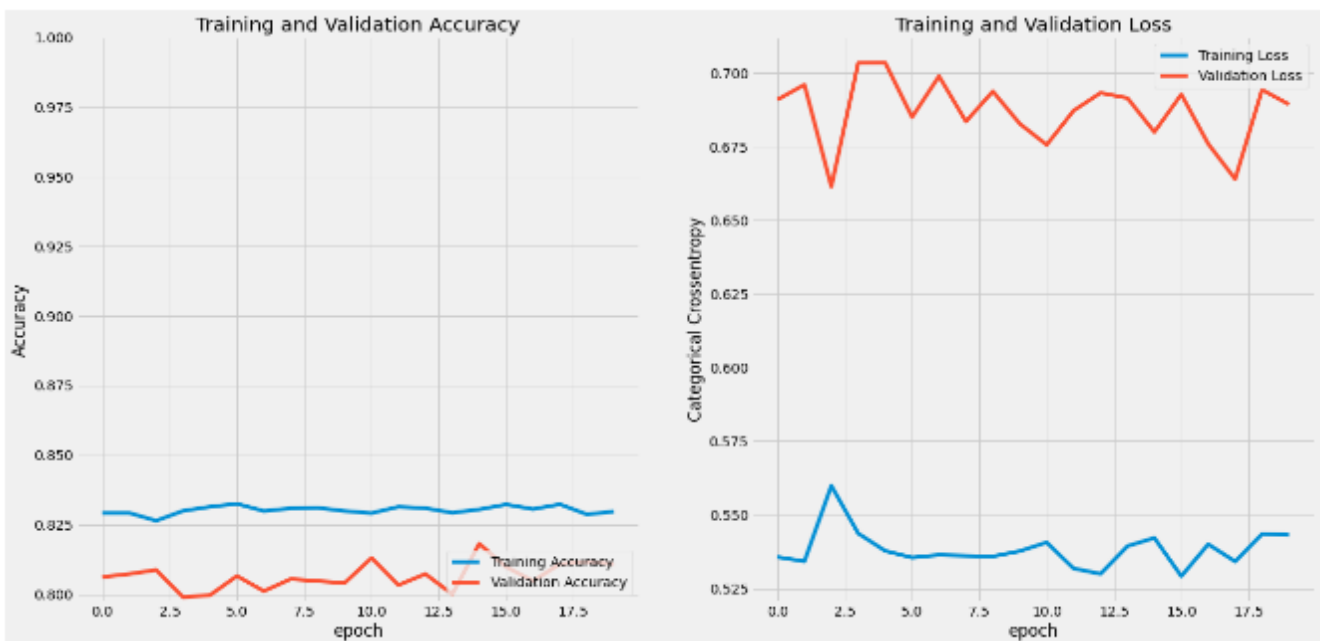
- Top 1 validation accuracy:
 - EfficientNet B0: 0.82942
 - EfficientNet B4: 0.81171

⇒ Cả hai model hội tụ rất nhanh và cơ bản được good fitting với accuracy cao hơn so với VGG16, accuracy của EfficientNet B4 thấp hơn so với B0 một chút nhưng thời gian train nhiều hơn rất nhiều.

- **Phase 2:** *Unfreeze các Multiply layers và tiếp tục train model*
 - ✓ Mở các lớp Multiply của model và tiếp tục train thêm 20 epoch.
 - ✓ Kết quả sau khi train:



EfficientNet B0



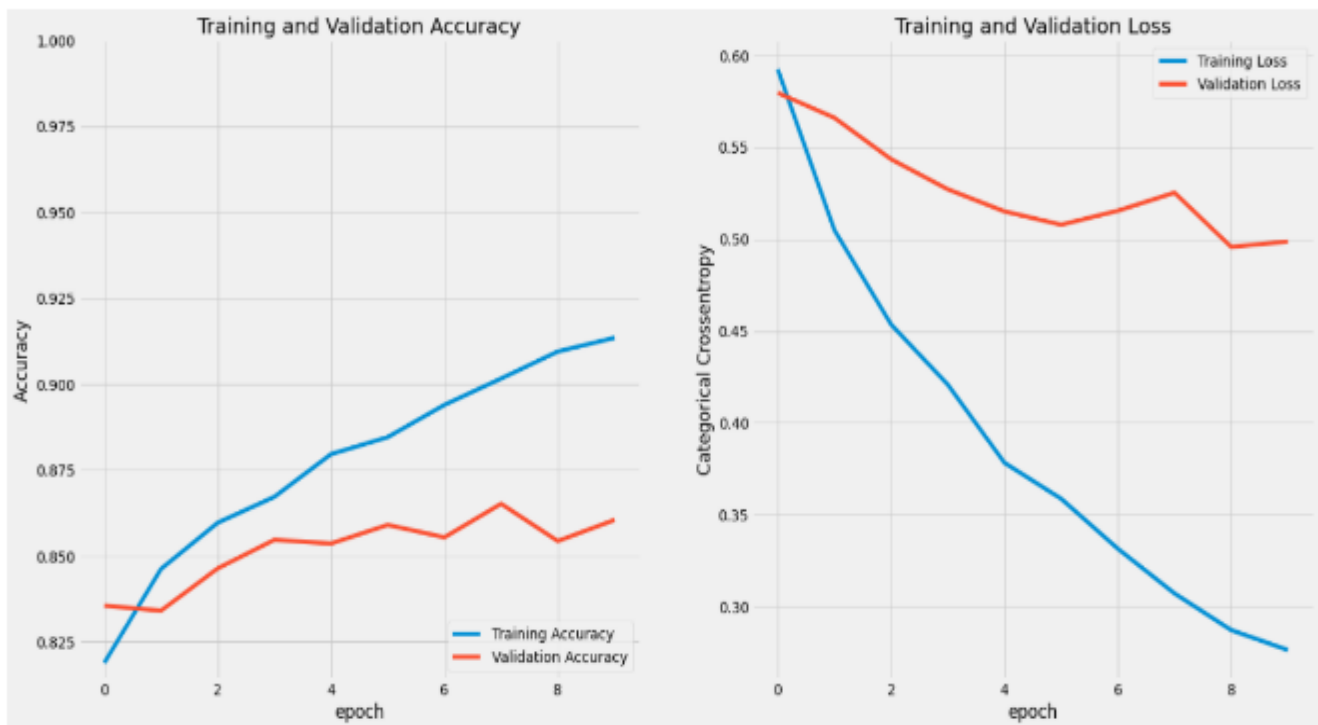
EfficientNet B4

- Top 1 validation accuracy:
 - EfficientNet B0: 0.84279
 - EfficientNet B4: 0.81821
- ⇒ Nhìn chung Phase 2 không cải thiện nhiều khi val_accuracy của 2 model tăng rất ít, overfit chưa đáng kể, kết quả của B4 thấp hơn B0.

❖ Chiến lược fine tuning 1: Áp dụng cho EfficientNet B0:

Từ kết quả của Phase 1 của chiến lược trước ta thấy các top layer được tinh chỉnh lại rất hiệu quả, tuy nhiên ở Phase 2 kết quả không cải thiện nhiều, bên cạnh đó kiến trúc B4 có kết quả thấp hơn so với B0 tuy nhiên thời gian train lại lâu hơn rất nhiều. Vì thế chiến lược này chúng em sẽ tập trung vào EfficientNet B0. Chiến lược fine tune này vẫn sử dụng lại Phase 1 của chiến lược 1 và thay đổi việc unfreeze các layer ở Phase 2:

- Thực hiện unfreeze **n** layer top rồi train thêm 10 epoch nữa với learning rate thấp.
- Sau nhiều lần thay đổi **n**, nhận thấy $n = 20$ tức unfreeze 20 layer top cho kết quả tốt hơn.
- Kết quả sau khi train 10 epoch với $n = 20$:



- Top 1 validation accuracy = 0.86520
- ⇒ Chiến lược này có phần tốt hơn so với chiến lược trước nhưng vẫn chưa quá nhiều, hiện tượng overfit bắt đầu xuất hiện nhưng chưa đáng kể do chỉ train 10 epochs.

d. Đánh giá:

- Train, Test, Validation accuracy:

✓ EfficientNet B0 v1:

- Train accuracy: 0.9130820631980896
- Validation accuracy: 0.8366461992263794
- Test accuracy: 0.847600519657135

✓ EfficientNet B4 v1:

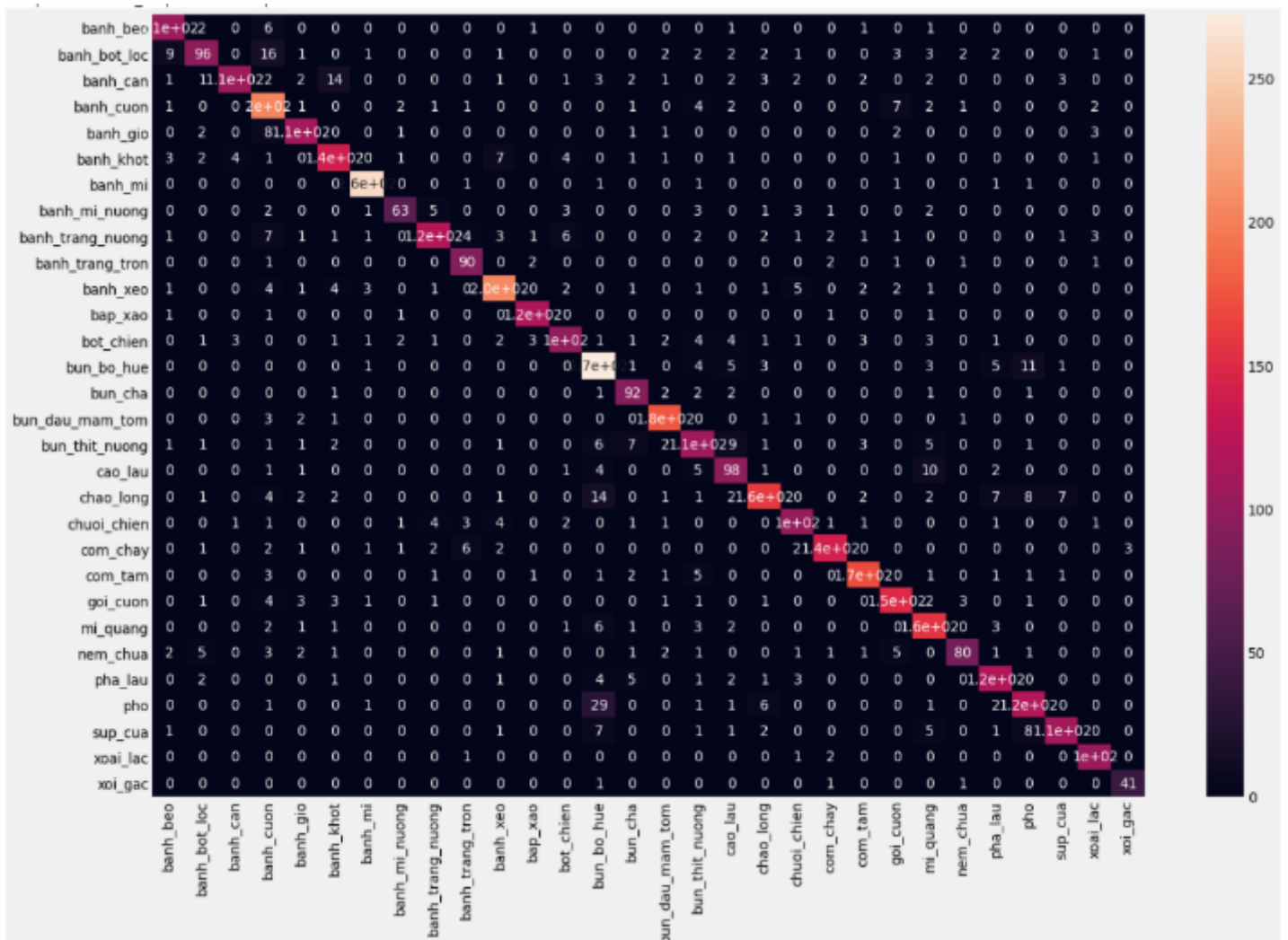
- Train accuracy: 0.8988913297653198
- Validation accuracy: 0.8117094039916992
- Test accuracy: 0.8357111811637878

✓ EfficientNet B0 v2:

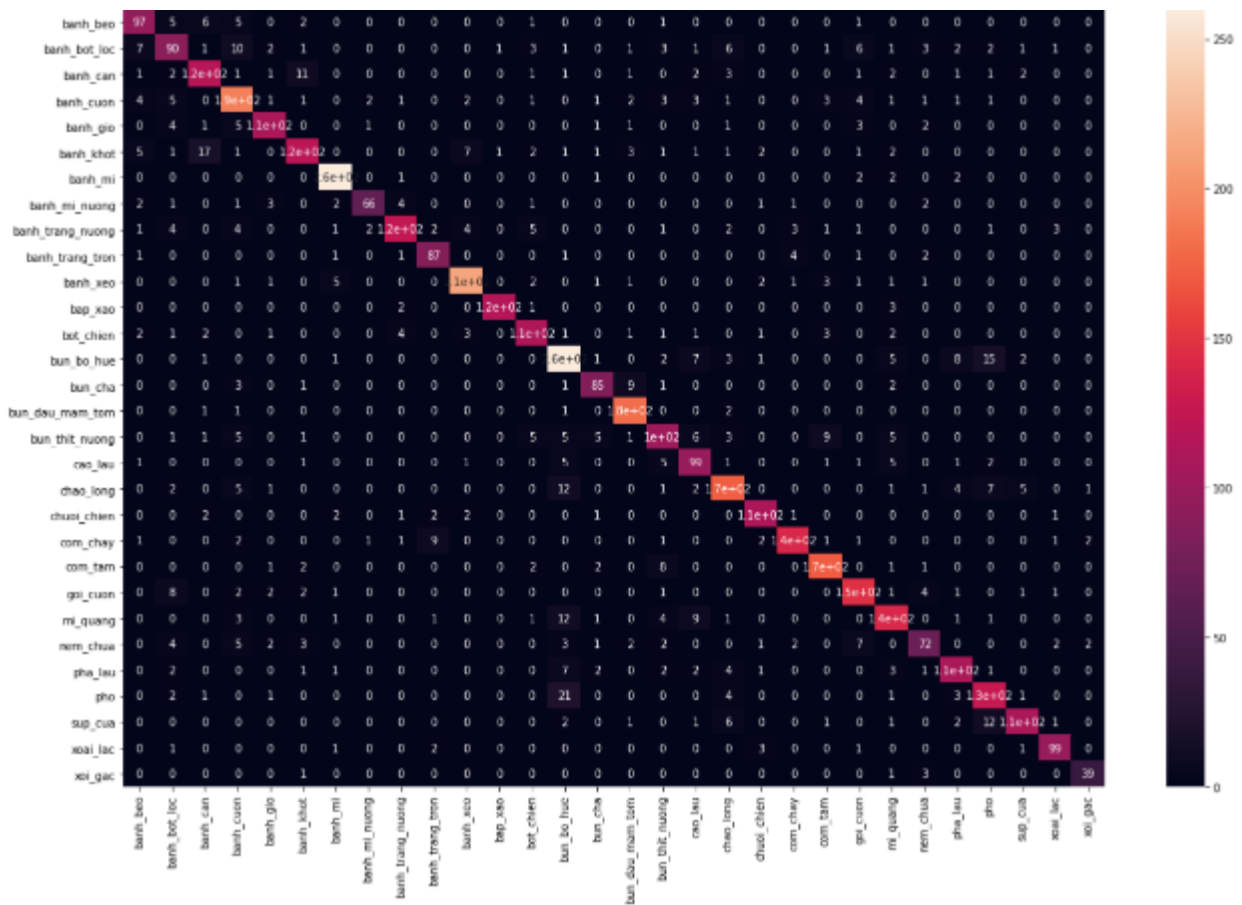
- Train accuracy: 0.9557808041572571
- Validation accuracy: 0.861944317817688
- Test accuracy: 0.8707306385040283

- Confusion matrix:

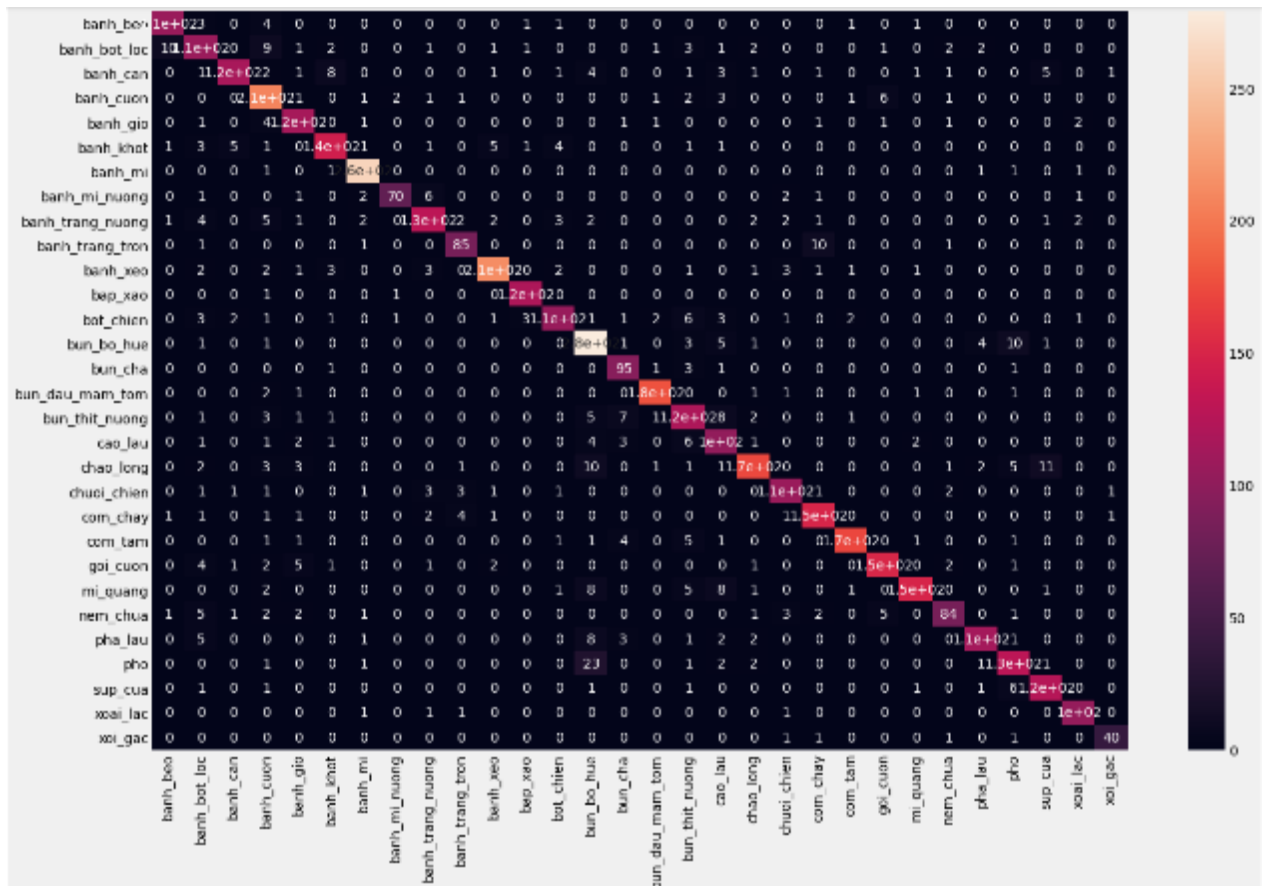
✓ EfficientNet B0 v1:



✓ EfficientNet B4 v1:



✓ EfficientNet B0 v2:



Nhân xét:

- **Về accuracy:** Với hai phương pháp fine tune ta thấy accuracy tăng nhẹ của phương pháp 2 so với phương pháp 1, và cao nhất so với các thuật toán trước. Tuy nhiên việc tìm ra phương pháp fine tune tốt nhất để tối ưu bài toán với EfficientNet là rất khó và tốn thời gian training.
- **Về Confusion matrix:** EfficientNet đã cải thiện được hơn so với kiến trúc VGG16 tuy nhiên các món ăn có ngoại hình rất giống nhau vẫn bị nhận nhầm, nhưng tỉ lệ nhận nhầm đã giảm nhiều.
- **Tổng kết:** EfficientNet là kiến trúc rất tốt khi nó tăng accuracy đáng kể so với VGG16 tuy nhiên rất khó trong quá trình fine tuning, đối với bài toán thì model EfficientNet B0 v2 là tốt nhất.

V. Đánh giá kết quả:

Accuracy Model	Train	Validation	Test
SVM với HOG	0.759	-	0.215
VGG16	0.873	0.621	0.623
VGG16 Transfer	0.998	0.786	0.797
EfficientNet B0 v1	0.913	0.837	0.848
EfficientNet B4 v1	0.899	0.812	0.836
EfficientNet B0 v2	0.956	0.862	0.871

- Qua quá trình xây dựng các model, model có kết quả tốt nhất là EfficientNet B0 với cách fine tuning thứ 2, model có kết quả kém nhất là SVM với rút trích đặc trưng HOG.
- Bước tiếp theo sẽ sử dụng kết quả của model EfficientNet B0 v2 để xây dựng một ứng dụng web có thể nhận dạng món ăn đường phố Việt Nam.

VI. Xây dựng web app:

1. Phân tích thị trường, đối tượng sử dụng:

Hiện nay, qua việc khảo sát nhóm nhận thấy có rất ít các ứng dụng nhận dạng món ăn và đặc biệt là món ăn Việt thì rất ít, mà đa số chỉ dừng lại ở việc nhận dạng được tên món chưa cung cấp cho người dùng thông tin cụ thể về món ăn, đặc biệt là bằng tiếng Anh phục vụ cho đối tượng du khách nước ngoài đến Việt Nam muốn tìm hiểu về ẩm thực Việt, hay đơn giản là muốn đọc tên món ăn và biết trong món ăn đó có thành phần nào gây dị ứng không.

2. Các chức năng của web-app:

Hướng tới đối tượng sử dụng là người nước ngoài, ứng dụng cho phép người dùng tải lên một tấm ảnh có thể được chụp từ smartphone hoặc url của tấm ảnh đó từ đó ứng dụng nhận dạng món ăn và xuất ra thông tin về món ăn đó gồm:

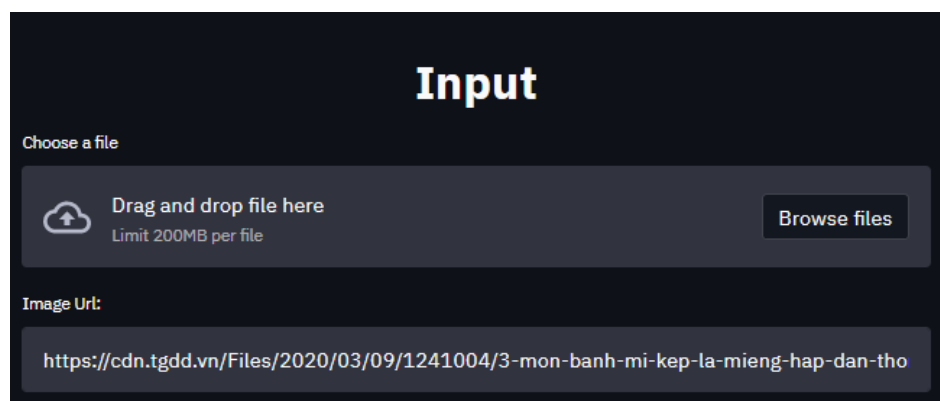
- Cách phát âm tên món đó trong tiếng Việt.
- Mô tả khái quát về món ăn.
- Các thành phần nguyên liệu trong món ăn.
- Mức giá trung bình.
- Các quán gốc, lâu đời có bán món ăn đó.
- 5 địa điểm nổi tiếng ở tp. Hồ Chí Minh có bán món ăn đó.

3. Thiết kế web-app:

Chúng em sử dụng chủ yếu ngôn ngữ Python với chủ yếu là thư viện **Streamlit** một công cụ mới được xem là một cuộc cách mạng trong việc tạo ra ứng dụng web vô cùng đơn giản và nhanh chóng.

Web app được thiết kế theo chiều dọc, tất cả thông tin và thao tác tập trung ở giữa điều này tiện cho việc sử dụng trên nền tảng mobile.

Ứng dụng vô cùng đơn giản với hai ô load input, một ô để người dùng có thể kéo thả ảnh vào, chọn ảnh trong máy hoặc chụp trực tiếp từ điện thoại, ô còn lại cho phép người dùng nhập vào url của ảnh. Từ đó tối ưu việc upload hình ảnh của người dùng.



Do model sử dụng hàm kích hoạt **softmax** ở layer cuối nên kết quả sẽ là phần trăm xác suất của các class nên khi thiết kế app chúng em đã cho thêm điều kiện là món ăn dự đoán là món có **phần trăm xác suất cao nhất** và phải **lớn hơn 70%**.

Về phần phát âm tên món ăn chúng em sử dụng thư viện **gTTS** của Google để tạo file audio phát âm tên từng món.

Về thông tin chúng em thu thập từ nhiều trang báo nổi tiếng về ẩm thực và có dẫn link khi người dùng bấm vào tên món, về phần các quán nổi tiếng chúng em thu thập theo thông tin đánh giá của Google và Foody các quán thuộc top có số lượng đánh giá nhiều và cao nhất, hoặc các quán có lịch sử lâu đời.

Input



Output

0:00 / 0:01

Bánh mì

Bánh mì is a short baguette with thin, crisp crust and soft, airy texture. It is often split lengthwise and filled with savory ingredients like a submarine sandwich and served as a meal. Plain bánh mì is also eaten as a staple food.

Ingredients:

- Bread
- Pork
- laksa leaves
- five-spice powder
- white wine
- cucumber
- seasoning: vinegar, ground pepper, sugar, salt, fish sauce, garlic, chili, cucumber, chili sauce

Famous places in HCM:

- Bánh mì Huỳnh Hoa: 26 Lê Thị Riêng St, Bến Thành Ward, District 1, HCM City
- Bánh mì Cô Diệp: 238 Võ Thành Trang St, Ward 11, Tân Bình District, HCM City
- Bánh mì cóc: 38 Nguyễn Thái Sơn St, Ward 3, Gò Vấp District, HCM City
- Bánh mì Bảy Hổ: 19 Huỳnh Khương Ninh St, Ba Kao Ward, District 1, HCM City
- Bánh mì thịt nướng Nguyễn Trác: 37 Nguyễn Trãi St, Bến Thành Ward, District 1, HCM City

Prices: 10,000- 50,000 VND

Probability: 99.99%

4. Chọn nền tảng deploy:

Để deploy ứng dụng chúng em chọn nền tảng **Heroku** một nền tảng đám mây dựa trên ứng dụng container, giúp chúng em có thể deploy miễn phí và nhanh chóng ứng dụng với source code từ Github mà không cần quan tâm việc mua tên miền, hay duy trì sự ổn định của máy chủ, phần cứng và cơ sở hạ tầng khác.



HEROKU

Link web: [Streamlit \(vietnamesefood-demo.herokuapp.com\)](https://streamlit(vietnamesefood-demo.herokuapp.com))

C. TÀI LIỆU THAM KHẢO

❖ SVM và HOG:

[1] Hải Hà – Viblo.asia - Tìm hiểu về phương pháp mô tả đặc trưng HOG (Histogram of Oriented Gradients)

Link: <https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-of-oriented-gradients-V3m5WAwXZ07>

[2] Scikit learn:

Link: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

[3] Machinelearningcoban.com – Thuật toán SVM

Link: <https://machinelearningcoban.com/2017/04/09/smv/#-xay-dung-bai-toan-toi-uu-cho-svm>

[4] Machinelearningcoban.com – Soft margin SVM

Link: <https://machinelearningcoban.com/2017/04/13/softmarginismv/>

[5] Baeldung.com - Multiclass Classification Using Support Vector Machines

Link: <https://www.baeldung.com/cs/svm-multiclass-classification>

❖ VGG16:

[6] neurohive.io - VGG16 Convolutional Network for Classification & Detection

Link: <https://neurohive.io/en/popular-networks/vgg16/>

[7] towardsdatascience.com - Step by step VGG16 implementation in Keras for beginners

Link: <https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c>

❖ VGG16 Transfer learning:

[8] machinelearningmastery.com - Transfer Learning in Keras with Computer Vision Models

Link: <https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/>

[9] learndatasci.com - Hands-on Transfer Learning with Keras and the VGG16 Model

Link: <https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/>

[10] tensorflow – Transfer learning and fine tuning

Link: <https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/>

❖ **EfficientNet:**

[11] Mingxing Tan, Quoc V. Le - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Link: <https://arxiv.org/pdf/1905.11946v5.pdf>

[12] Google Ai Blog - EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling

Link: <https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>

[13] Yixing Fu - Keras - Image classification via fine-tuning with EfficientNet

Link: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

[14] viblo.asia - EfficientNet: Cách tiếp cận mới về Model Scaling cho Convolutional Neural Networks

Link: https://viblo.asia/p/efficientnet-cach-tiep-can-moi-ve-model-scaling-cho-convolutional-neural-networks-Qbq5QQzm5D8#_huong-xu-ly-cua-cac-tac-gia-6

[15] kaggle.com - Fine-tuning EfficientNetB0 on CIFAR-100

Link: <https://www.kaggle.com/micajoumathematics/fine-tuning-efficientnetb0-on-cifar-100>