

**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

<b>CHƯƠNG 1: TỔNG QUAN.....</b>	<b>1</b>
<b>I.    Mô tả bài toán: .....</b>	<b>1</b>
1.    Ngữ cảnh ứng dụng: .....	1
2.    Bài toán: .....	2
<b>II.    Mô tả dữ liệu: .....</b>	<b>3</b>
<b>CHƯƠNG 2: CÁC NGHIÊN CỨU TRƯỚC.....</b>	<b>3</b>
<b>CHƯƠNG 3: XÂY DỰNG BỘ DỮ LIỆU.....</b>	<b>5</b>
<b>I.    Cách thức xây dựng bộ dữ liệu:.....</b>	<b>5</b>
1.    Thiết kế danh sách các món ăn đường phố Việt: .....	5
2.    Tiêu chí thu thập dữ liệu: .....	6
3.    Lưu ý về các món có ngoại hình giống nhau: .....	7
4.    Thu thập, lọc và gán nhãn dữ liệu: .....	8
<b>II.    Số lượng, độ đa dạng: .....</b>	<b>9</b>
<b>III.  Phân chia Train/Test/Validation: .....</b>	<b>10</b>
<b>IV.  Tiền xử lý dữ liệu: .....</b>	<b>11</b>
<b>V.    Đặc trưng dữ liệu: .....</b>	<b>12</b>
<b>CHƯƠNG 4: TRAINING VÀ ĐÁNH GIÁ. ....</b>	<b>13</b>
<b>I.    Thuật toán và quá trình training model: .....</b>	<b>13</b>
1.    SVM với rút trích đặc trưng HOG: .....	13
2.    VGG16: .....	17
3.    VGG16 Transfer learning: .....	23
4.    EfficientNet Transfer learning: .....	27
<b>II.    Đánh giá chung: .....</b>	<b>37</b>
<b>CHƯƠNG 5: ỨNG DỤNG VÀ HƯỚNG PHÁT TRIỂN. ....</b>	<b>38</b>
<b>I.    Xây dựng ứng dụng web: .....</b>	<b>38</b>
1.    Phân tích thị trường, đối tượng sử dụng: .....	38
2.    Các chức năng của web-app: .....	38
3.    Thiết kế web-app: .....	38
4.    Chọn nền tảng deploy: .....	39
5.    Thử nghiệm và nhận phản hồi: .....	40
<b>II.    Hướng phát triển: .....</b>	<b>41</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>42</b>

## CHƯƠNG 1: TỔNG QUAN.

### I. Mô tả bài toán:

#### 1. Ngữ cảnh ứng dụng:

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, 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.

## 2. 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:

### a. Input:

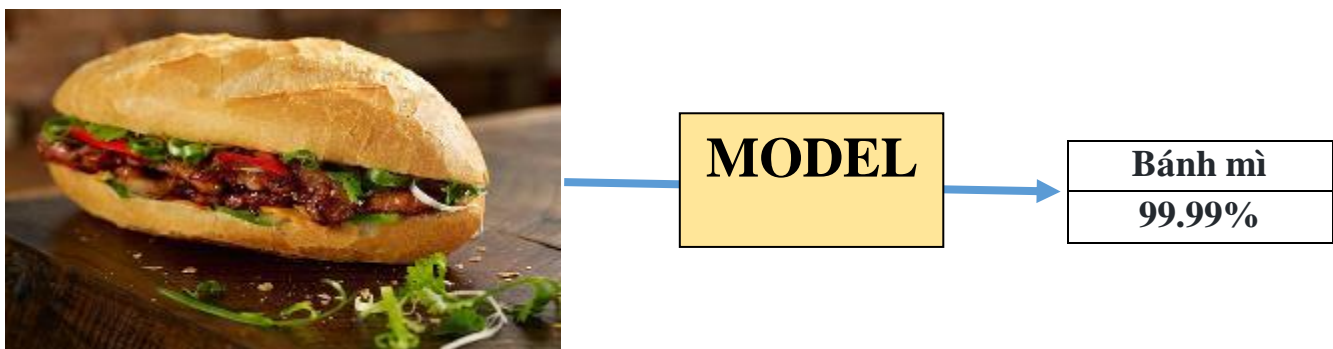
- Một bức ảnh chụp món ăn:
  - Được chụp bằng smartphone hoặc máy ảnh.
  - Góc chụp tùy ý đủ thấy toàn bộ món ăn.
  - Chỉ có một món ăn trong ảnh.

### b. Output:

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

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

Ví dụ:



⇒ Từ output trên, khi chúng ta nhận dạng được tên món ăn thì dễ dàng xuất ra thông tin về món ăn đó.

## II. Mô tả dữ liệu:

Từ việc xác định bài toán ở trên nhóm nhận thấy rằng dataset là hình ảnh các món ăn phải đa dạng về góc chụp độ phân giải và cũng phải phong phú về số lượng món và lượng hình ảnh của các món.

Đầu tiên, nhóm dự định sẽ chụp hình các món ăn đường phố. Tuy nhiên do tình hình dịch bệnh và giãn cách xã hội theo chỉ thị 16 nên việc ra đường chụp hình thực tế các món ăn là bất khả thi.

Vì lí do trên nhóm chuyển sang thu thập dữ liệu món ăn trên Internet. Các dataset có sẵn trên Internet về món ăn Việt rất ít và số lượng món ăn không nhiều, do đó nhóm quyết định sẽ tự xây dựng bộ dữ liệu bằng các thu thập hình ảnh từ các nền tảng tìm kiếm và chia sẻ hình ảnh như **Google Image, Microsoft Bing, Instagram, Foody**, ... Hình ảnh thu thập trên các nền tảng này vô cùng đa dạng về góc chụp cũng như độ phân giải vì thế khó khăn duy nhất của việc thu thập là làm sạch dữ liệu (*chi tiết ở chương 3*).

## CHƯƠNG 2: CÁC NGHIÊN CỨU TRƯỚC.

Sau quá trình khảo sát trên Internet nhóm nhận thấy chủ đề nhận dạng món ăn Việt khá ít người làm và bộ dataset khá nhỏ, cụ thể nhóm tìm được 3 bài sau:

### ❖ **Visual Recognition for Vietnamese Foods – Chris Tran (2020):**

- Link: <https://chriskhanhtran.github.io/posts/vn-food-classifier/>
- Tác giả xây dựng dataset từ việc thu thập ảnh món ăn chỉ trên **Bing** gồm 11 món ăn và nước uống với tổng số ảnh trên 6000 hình, lượng dữ liệu khá ít. Chỉ train bằng ResNet-50 pretrained model tuy nhiên với dataset ít và phân hóa rõ rệt nên accuracy khá cao ~ 94%.
- Có xây dựng web app tuy nhiên chưa hoàn thiện còn bị lỗi và chưa nêu được nhiều thông tin món ăn.

❖ **Vietnamese food recognition using convolutional neural networks - Thai Van Phat, Dang Xuan Tien, Quang Pham, Nguyen Pham, Binh T. Nguyen (2017):**

- Link: <https://sci-hub.se/https://ieeexplore.ieee.org/document/8119446>
- Mục đích của bài báo chủ yếu so sánh hai phương pháp traditional hand-crafted features và convolutional neural networks
- Tác giả thu thập dữ liệu cho 5 món ăn với hơn 2300 ảnh từ **Foody.com** và **Diadiemanuong.com** do mục đích so sánh nên số lượng món ăn quá ít các món có hình dáng rất khác nhau.

Method	Model	Train Set	Val Set	Test Set
HOG	ANN	70.4%	50.0%	50.0%
	SVM	92.14%	52.42%	47.39%
	XGBoost	99.62%	51.54%	44.35%
SIFT	SVM	61.51%	55.51%	50.43%
	XGBoost	89.67%	55.51%	43.48%
SURF	SVM	65.06%	58.14%	57.39%
	XGBoost	<b>100.0%</b>	56.82%	45.22%
HOG + SIFT	SVM	98.92%	61.67%	<b>62.61%</b>
	XGBoost	82.51%	60.0%	53.04%
HOG + SURF	SVM	85.84%	61.67%	56.08%
	XGBoost	<b>100.0%</b>	<b>62.11%</b>	53.04%

Models		lr	Train Set	Val Set	Test Set
GoogleNet	FineTune	0.0003	99.59%	98.97%	<b>97.39%</b>
		0.001	99.82%	<b>99.12%</b>	99.13%
		0.003	<b>99.98%</b>	98.83%	96.52%
	Scratch	0.0003	73.91%	77.67%	75.65%
		0.001	76.07%	78.71%	76.96%
		0.003	80.48%	82.23%	78.26%
AlexNet	FineTune	0.0003	99.08%	97.36%	<b>97.39%</b>
		0.001	99.62%	98.68%	97.82%
		0.003	97.85%	97.36%	92.61%
	Scratch	0.0003	63.94%	74.89%	64.35%
		0.001	66.74%	74.01%	66.96%
		0.003	69.69%	86.00%	68.69%

Bảng 1: Accuracy của traditional hand-crafted features và convolutional neural networks

❖ **30VNFoods – Quan Dang (2021):**

- Link: <https://github.com/18520339/30VNFoods>
- Bài nghiên cứu này diễn ra cùng thời gian với việc thực hiện đồ án của nhóm. Đây có lẽ là bài nghiên cứu có đề tài cũng như bộ dữ liệu gần với nhóm nhất, khi tác giả thu thập dữ liệu từ Internet với 30 món ăn và hơn 20000 hình, tuy nhiên ảnh vẫn chưa được lọc kỹ và hơn nữa số class (món ăn) của bài nghiên cứu này là khác so với nhóm.
- Tác giả sử dụng nhiều kiến trúc CNN khác nhau như: InceptionResNet (v2, v3), ResNet152v2, VGG19, Xception. Tuy nhiên accuracy chưa cao:

	Accuracy	Top 3 Accuracy	Top 5 Accuracy
<b>ResNet152V2</b>	0.778529	0.916103	0.958251
<b>InceptionV3</b>	0.747117	0.901789	0.951093
<b>Xception</b>	0.737177	0.901789	0.951491
<b>InceptionResNetV2</b>	0.724056	0.891451	0.943539
<b>VGG19</b>	0.702187	0.885089	0.934791

Bảng 2: Accuracy của tập Validation

	Accuracy	Top 3 Accuracy	Top 5 Accuracy
<b>ResNet152V2</b>	0.775397	0.921230	0.960714
<b>Xception</b>	0.730357	0.900198	0.947222
<b>InceptionResNetV2</b>	0.729167	0.901587	0.951389
<b>InceptionV3</b>	0.727183	0.895833	0.944841
<b>VGG19</b>	0.704762	0.879167	0.935119

Bảng 3: Accuracy của tập Test

- Validation accuracy cao nhất: 0.78 và Test accuracy cao nhất: 0.77
- Về phần thiết kế ứng dụng vẫn còn bị lỗi và không cung cấp thông tin món ăn ngoài tên món.



## CHƯƠNG 3: XÂY DỰNG BỘ DỮ LIỆU.

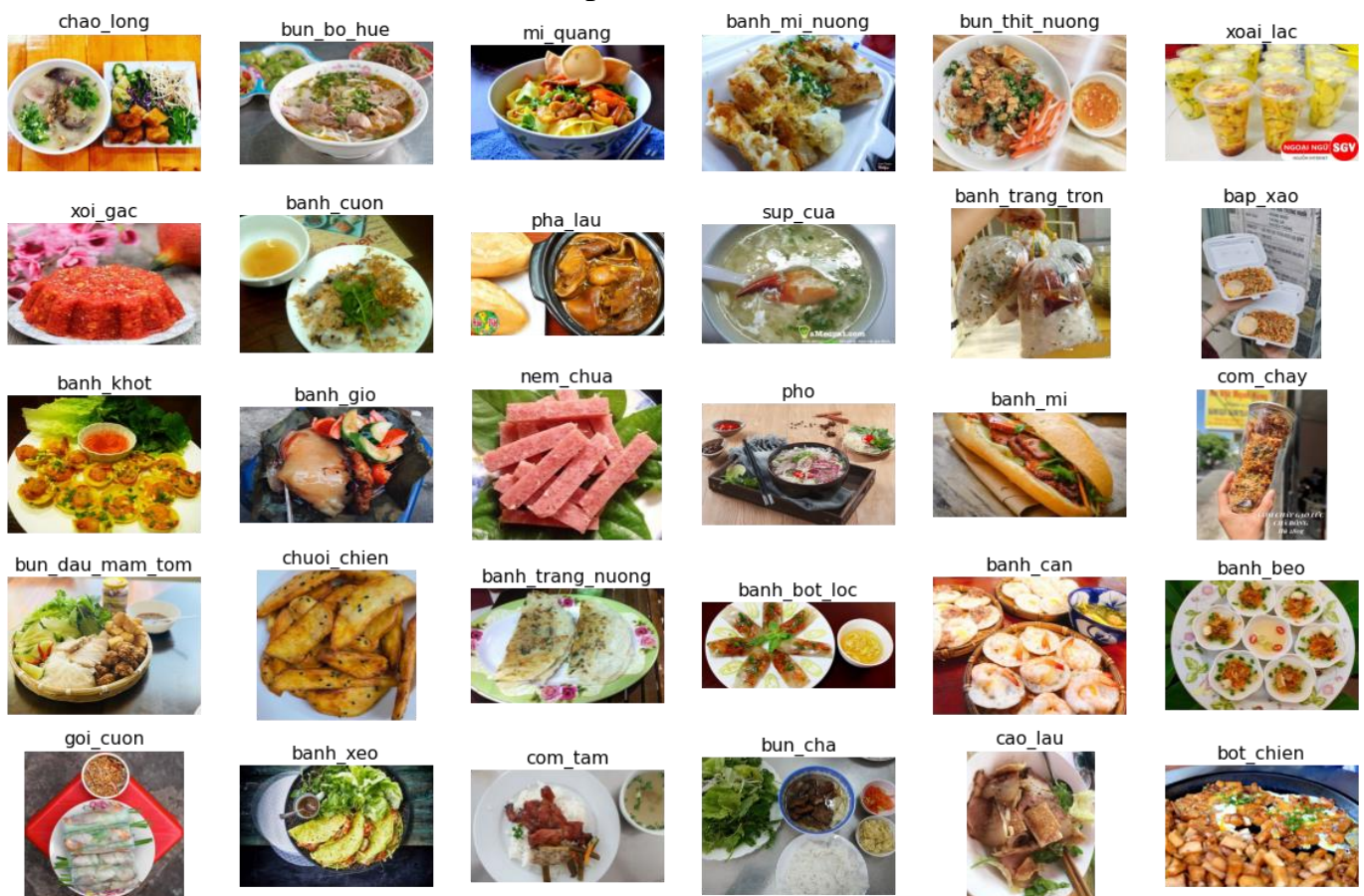
### I. Cách thức xây dựng bộ dữ liệu:

#### 1. 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       | – Phở                   | – 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    | – Bọt chiên             | – Bún bò Huế     |
| – Bánh mì         | – Cao lầu               | – Cháo lòng      |
| – Xôi gấc         | – Gỏi cuốn              | – Mì Quảng       |
| – Bún đậu mắm tôm | – Súp cua               | – Xoài lắc       |



Hình 1: Ảnh minh họa cho từng món ăn

## 2. Tiêu chí thu thập dữ liệu:

Do tình hình dịch bệnh nên không việc thu thập ảnh chụp món ăn thực tế là vô cùng 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, lấy 1 ảnh.
- Ảnh không liên quan đến món ăn.

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



*Một số ảnh không lấy:*



*Hình 2, 3, 4*



*Hình 5: Có nhiều món*



*Hình 6: Không rõ món ăn*



*Hình 7: Bị chữ che lấp*



*Hình 8: Ảnh bị trùng*



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

- **Bánh căn và Bánh khọt:** giống nhau vì gồm nhiều bánh hình tròn, dẹt.
  - **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.



Hình 9: Bánh căn



Hình 10: 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.



Hình 11: Phở



Hình 12: 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.



Hình 13: Bún thịt nướng



Hình 14: Bún chả

- 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 giống nhau.

#### 4. 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:



Hình 15: Các folder ảnh

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



Hình 16: Cách đặt tên ảnh

## II. Số lượng, độ đa dạng:

Sau khi mất 3 tuần để thu thập và label cho tất cả dữ liệu, kết quả thu được tổng cộng **23178** ảnh cho **30** classes 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

Bảng 4: Số lượng ảnh của từng Class

### Nhận xét:

- Do dữ liệu thu thập từ Internet nên số lượng giữa các class không cân nhau nhưng đa phần 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 có 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

### III. 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:

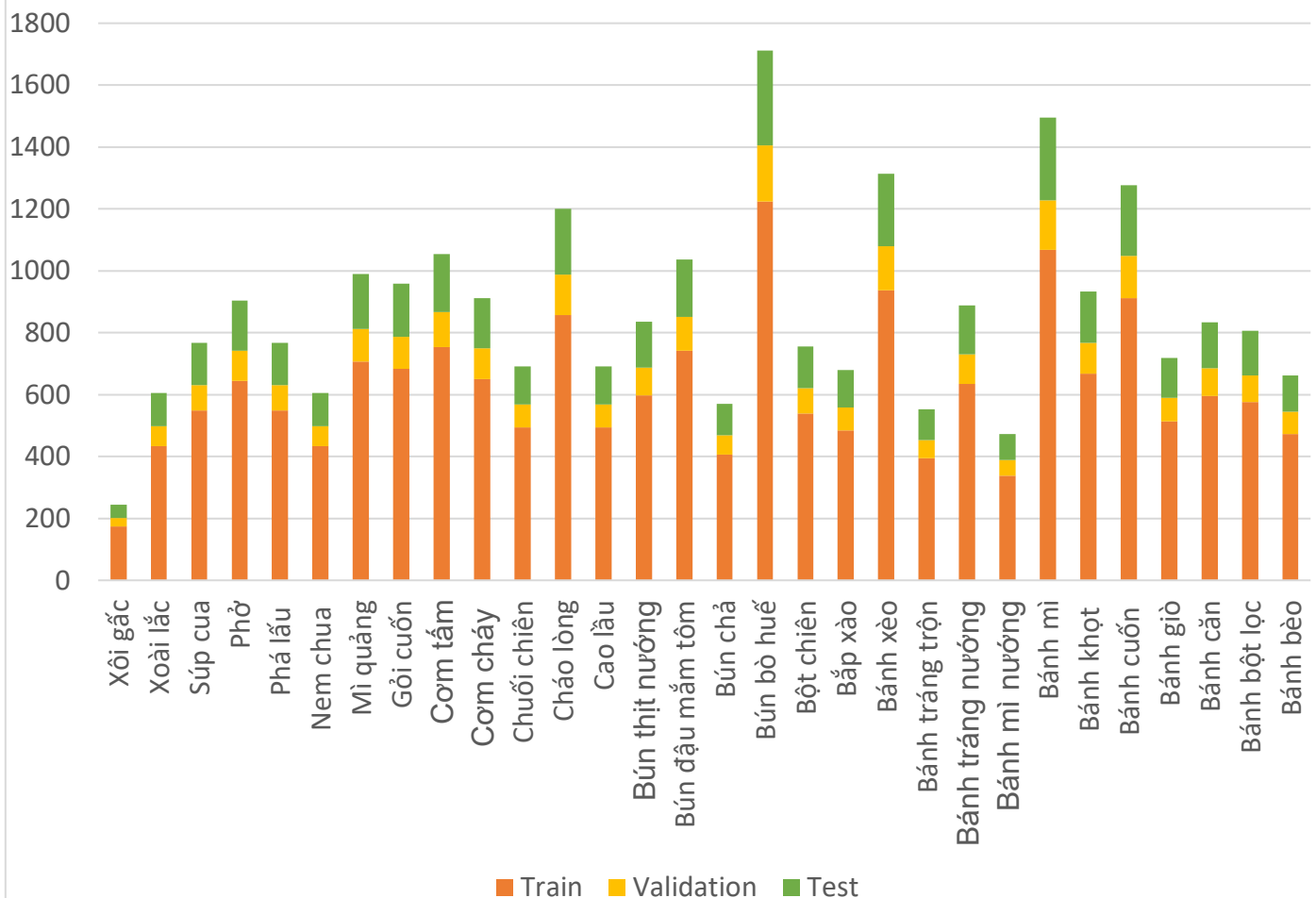
**68% Train : 12% Validation : 20% Test**

Chia như thế để tận dụng hết dữ liệu và đủ dữ liệu cho quá trình Train

	SVM - HOG	Deep learning
<b>Train</b>	18552	15785
<b>Validation</b>		2767
<b>Test</b>	4626	4626

Bảng 5: Tỉ lệ Train/ Validation/ Test

**Biểu đồ 1: Số lượng ảnh từng label, chia theo train/val/test**





#### IV. Tiền xử lý dữ liệu:

Do số lượng ảnh lớn và không đồng đều về số lượng, kích thước và độ phân giải nên trước khi huấn luyện 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 Deep learning nên việc tiền xử lý ảnh cũng có sự khác biệt.

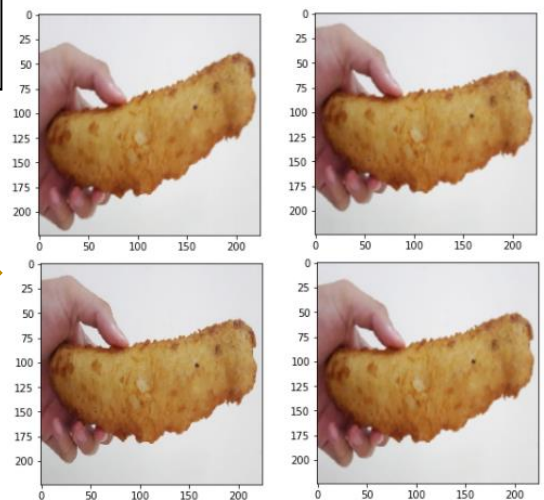
- Đối với việc sử dụng **HOG**: chúng em chỉ chuyển kích thước tất cả ảnh về **224x224** sau đó rút trích đặc trưng bằng HOG.
- Resize về **224x224** là do các **pretrained model** bên dưới đều train trên tập **ImageNet** với kích thước ảnh **224x224** và đảm bảo thời gian train.
- Với các thuật toán **Deep learning**: ngoài việc resize về **224x224** chúng em sử dụng hàm **ImageDataGenerator()** của Keras để tiền xử lý dữ liệu, như sau:

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

- **Shear\_range**: làm méo ảnh.



Shear\_range = 0.1

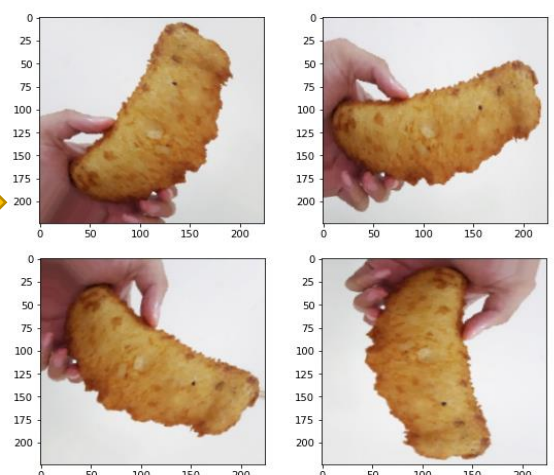


Hình 17: ví dụ hàm *shear\_range*

- **rotation\_range**: xoay ảnh theo độ.



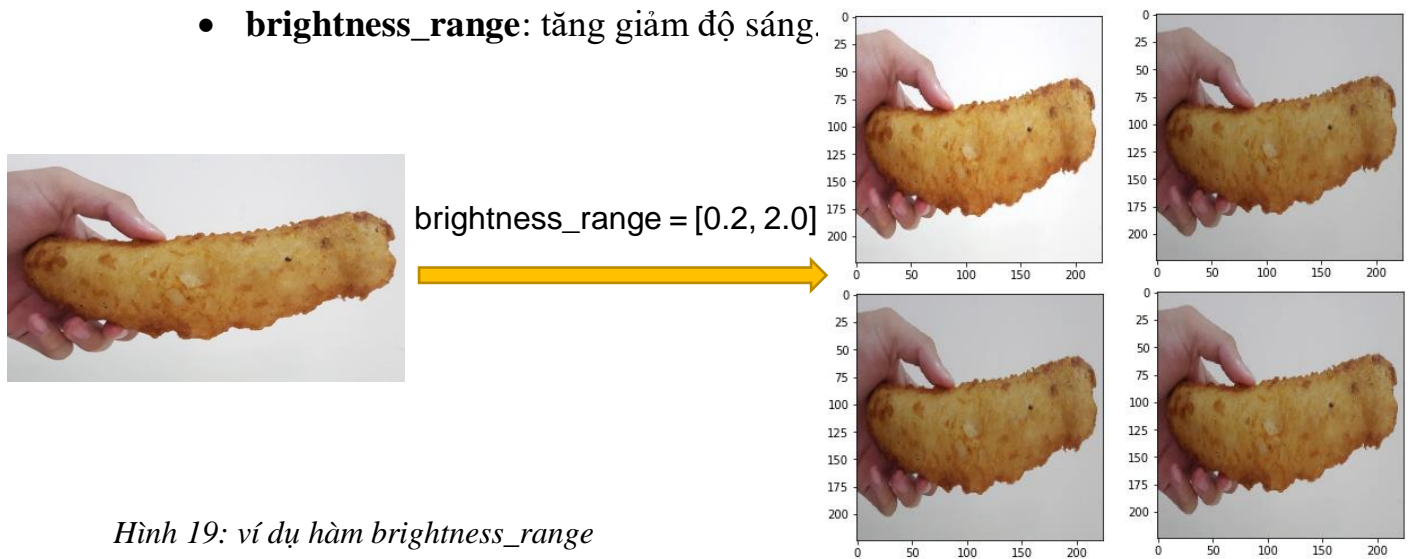
rotation\_range = 45



Hình 18: ví dụ hàm *rotation\_range*

Chúng em sử dụng xoay với góc 45 độ, để những món có dạng hình trụ hoặc những hình dạng khi xoay sẽ được ảnh khác như bánh mì, chuối chiên...

- **brightness\_range**: tăng giảm độ sáng.



Hình 19: ví dụ hàm *brightness\_range*

Mức tăng giảm độ sáng  $[0.7, 1.1]$  để 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.

- **validation\_split**: dùng chia train và validation với **val = 0,15 train**.

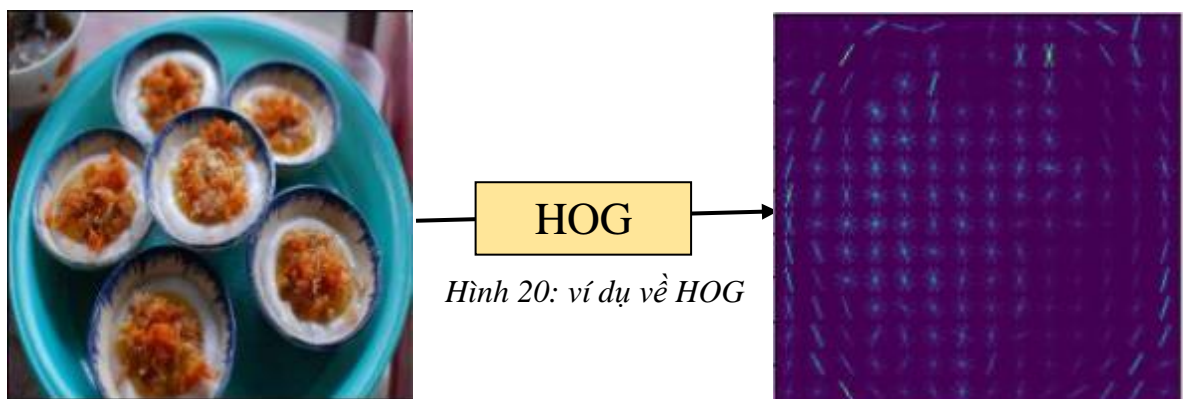
## V. Đặc trưng dữ liệu:

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

- Cách thủ công bằng rút trích đặc trưng HOG.
- Các lớp Feature extractor của các mô hình Deep learning.

### HOG (Histogram of oriented gradients):

- 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 tinh chỉnh của nhóm thì thu được vector đặc trưng có **15488** phần tử.



Hình 20: ví dụ về HOG

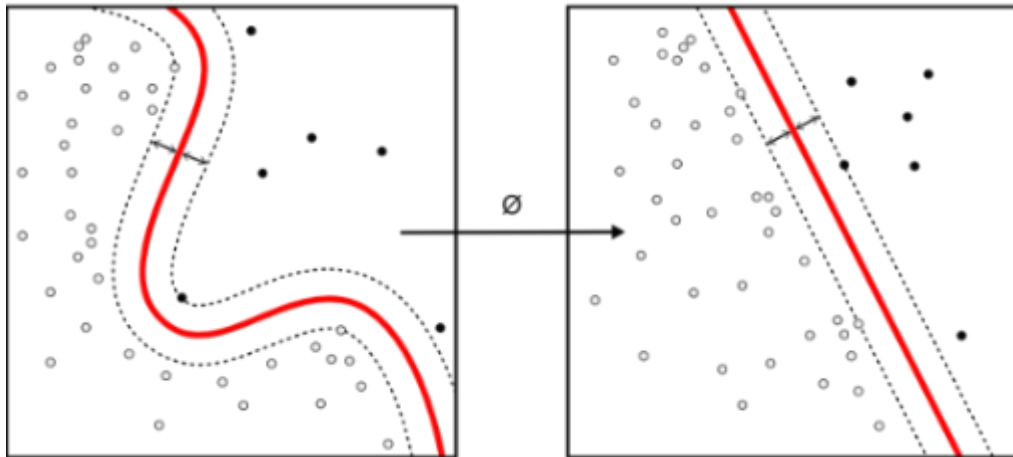
## CHƯƠNG 4: TRAINING VÀ ĐÁNH GIÁ.

### I. Thuật toán và quá trình training model:

#### 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

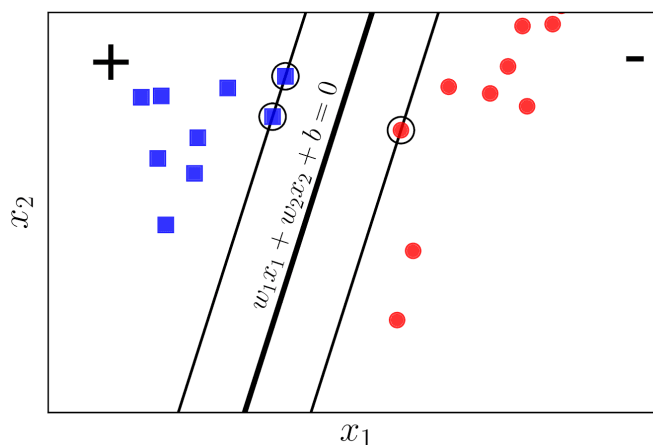


##### b. Thuật toán SVM (Support vector machine): *Hình 21: minh họa SVM*

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.

Ví dụ:

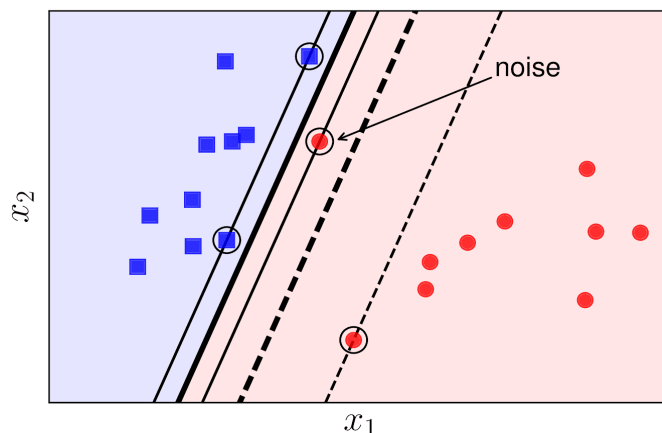


Hình 22

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:



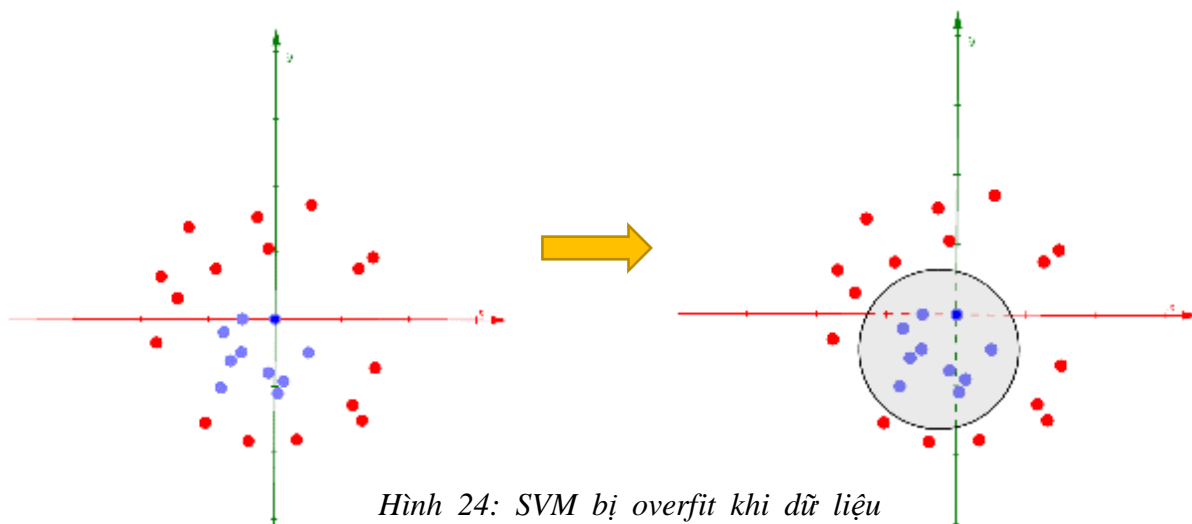
Hình 23

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.

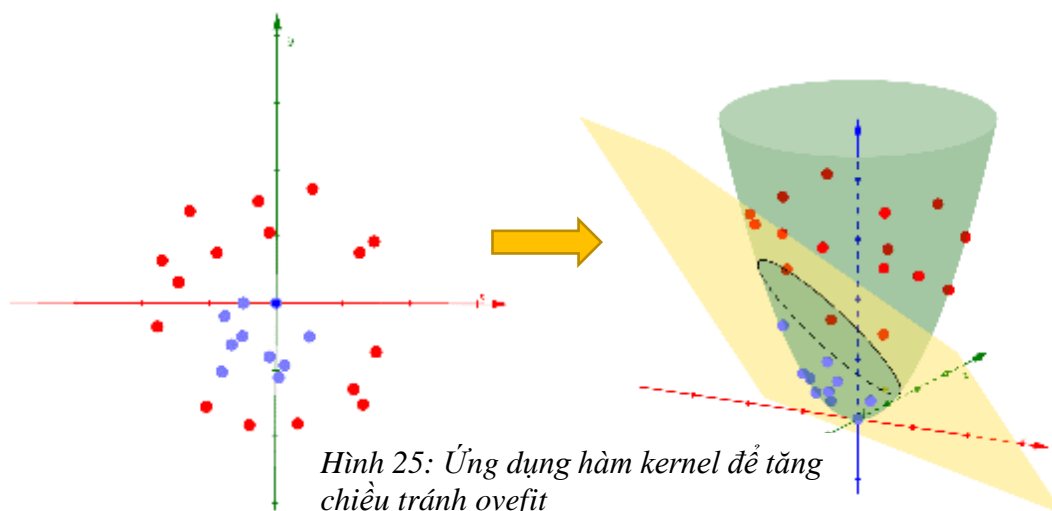
Ví dụ:



Hình 24: SVM bị overfit khi dữ liệu không phân chia tuyến tính



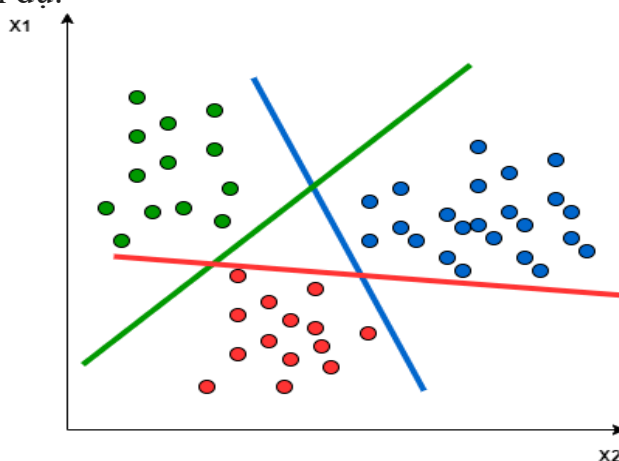
- Trong không gian hai chiều, các điểm dữ liệu trên không phân biệt tuyến tính, nếu chỉ sử dụng SVM đơn thuần phân lớp, ta sẽ được một “siêu mặt phẳng” phân biệt hai lớp như hình bên phải.
- Dễ dàng thấy được “siêu mặt phẳng” hiện tại đã gây ra tình trạng overfitting.
- Vậy để giải quyết trường hợp này, 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:



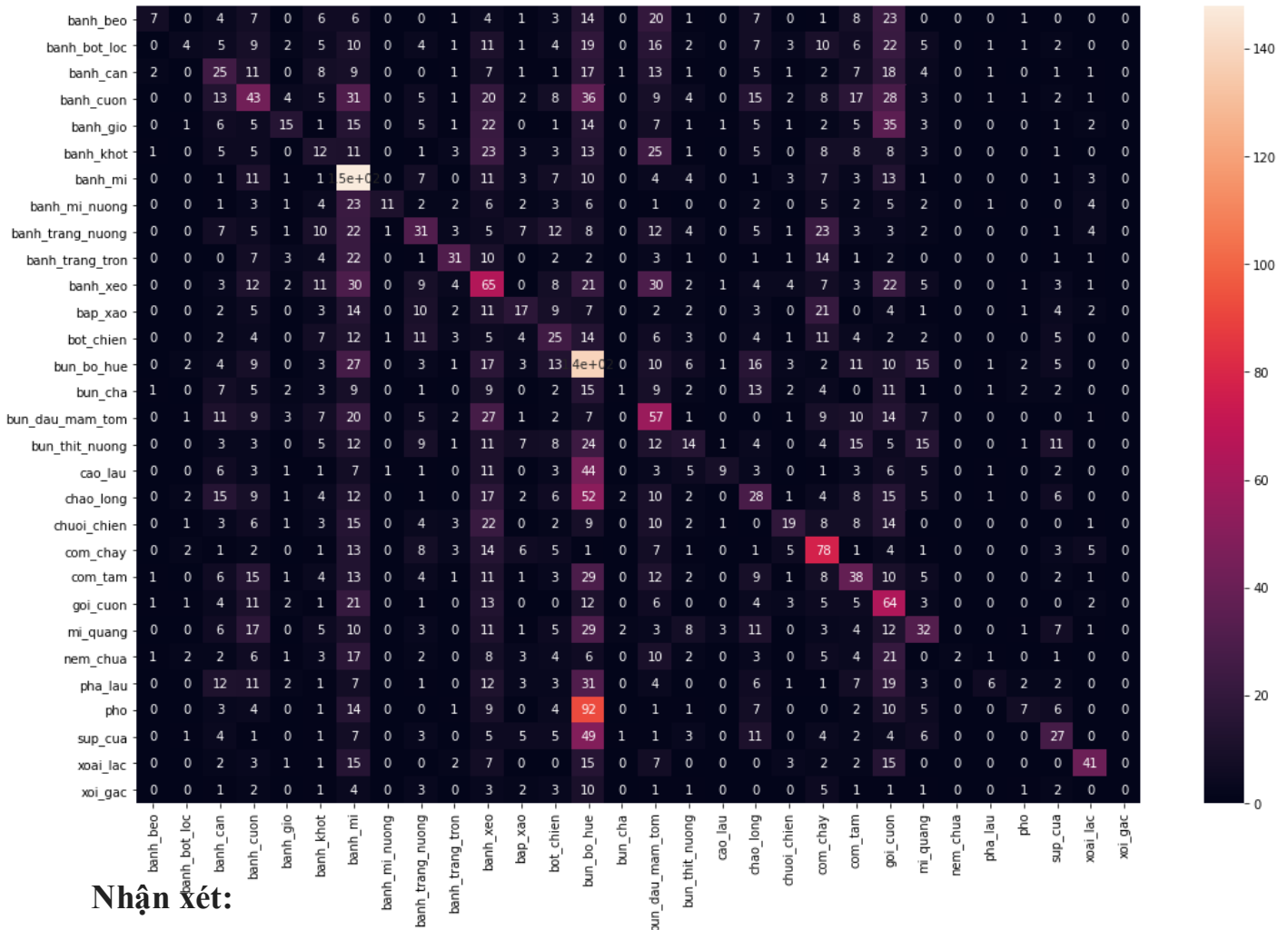
Trong ví dụ trên. Với mỗi lớp dữ liệu màu xanh dương, xanh lá, đỏ, ta sẽ tìm được các đường thẳng phân chia các lớp này với các lớp còn lại với màu tương ứng.

### c. Nhận xét đánh giá:

#### Accuracy:

- Accuracy train: 0.7589796138496386
- Accuracy test: 0.2148403796376186

#### Confusion matrix:



#### Nhận xét:

- Dựa vào **accuracy** dễ dàng nhận thấy được mô hình đã **overfitting nặng**.
- Dựa vào **confusion matrix** và **accuracy test**, ta thấy model dự đoán sai rất nhiều, nhiều class không dự đoán được.
- Kết quả kém này có thể do lượng dữ liệu quá đa dạng phức tạp HOG trích xuất đặc trưng về hướng của cạnh trong khi mỗi món ăn được bày trí nhiều kiểu khác nhau.
- Nhận thấy rút trích đặc trưng thủ công và các thuật toán Machine learning cơ bản để giải quyết vấn đề này rất tốn thời gian (train với Google colab không GPU thì **train** khoảng 8 giờ, **test** hơn 1 giờ) và kết quả thu được rất tệ, nên chúng em chuyển sang Deep learning mà không mất thời gian để tối ưu nữa.

## 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.

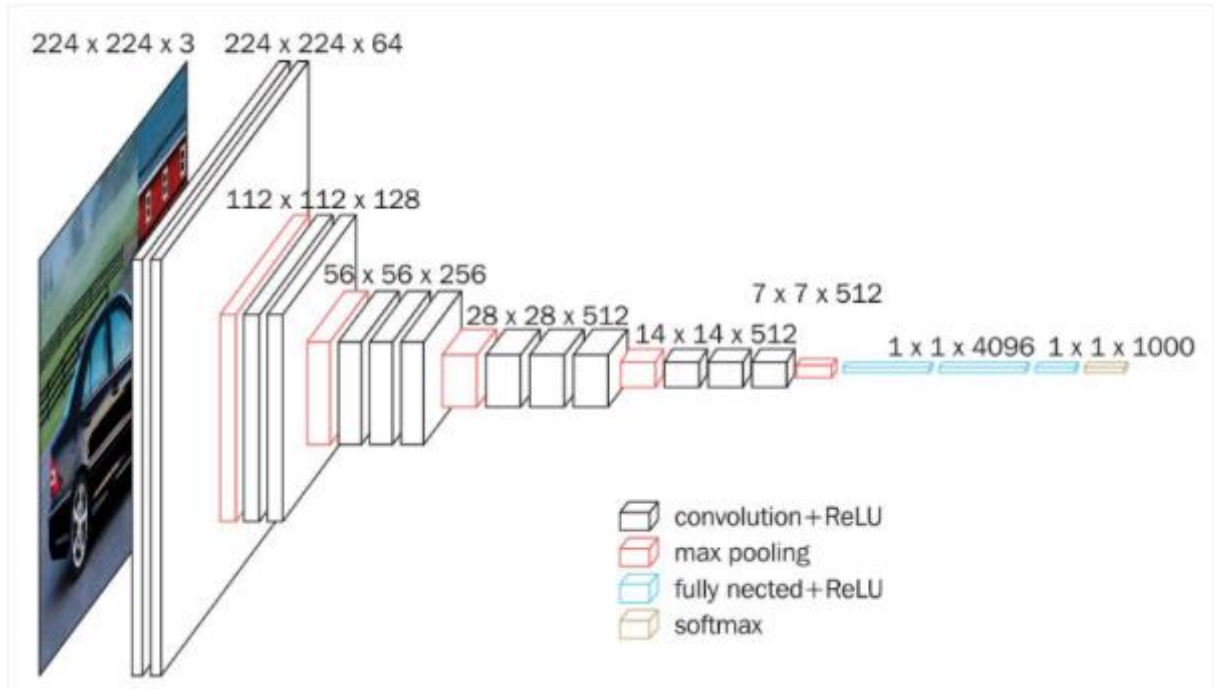
Tại sao lại chọn thuật toán tối ưu Adam:

- 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:

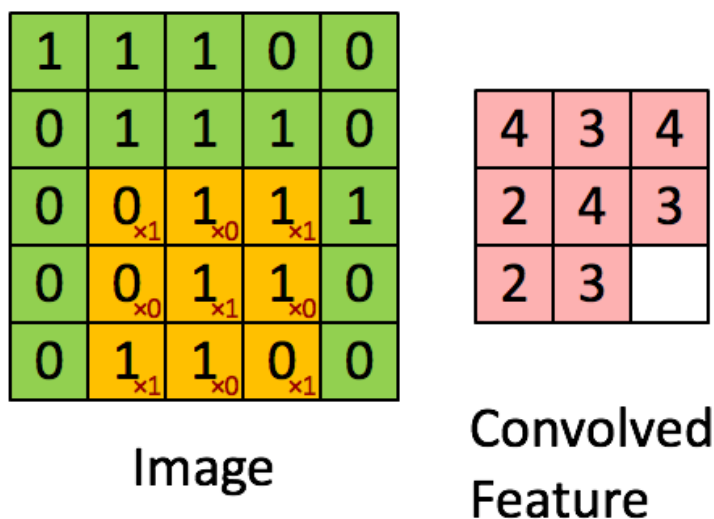


Hình 27: VGG16 Architecture

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

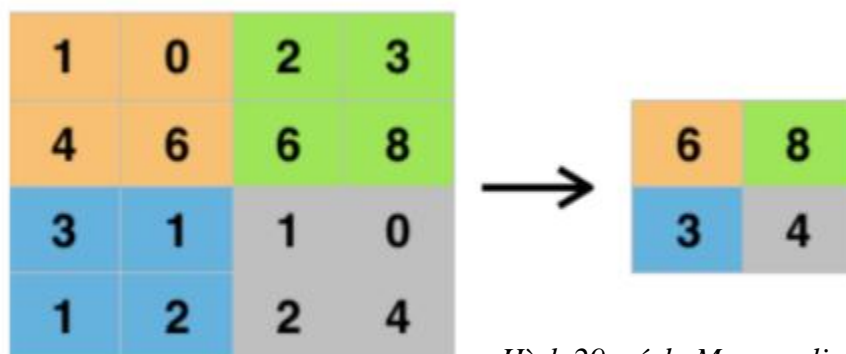
Sau đó, hình ảnh ban đầu được chuyển đến một chồng các lớp convolutional với kích thước kernel là 3x3, stride được giữ cố định là 1 và max-pooling với kích thước là 2x2 và stride lúc này mang giá trị là 2

- **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



Hình 28

- **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

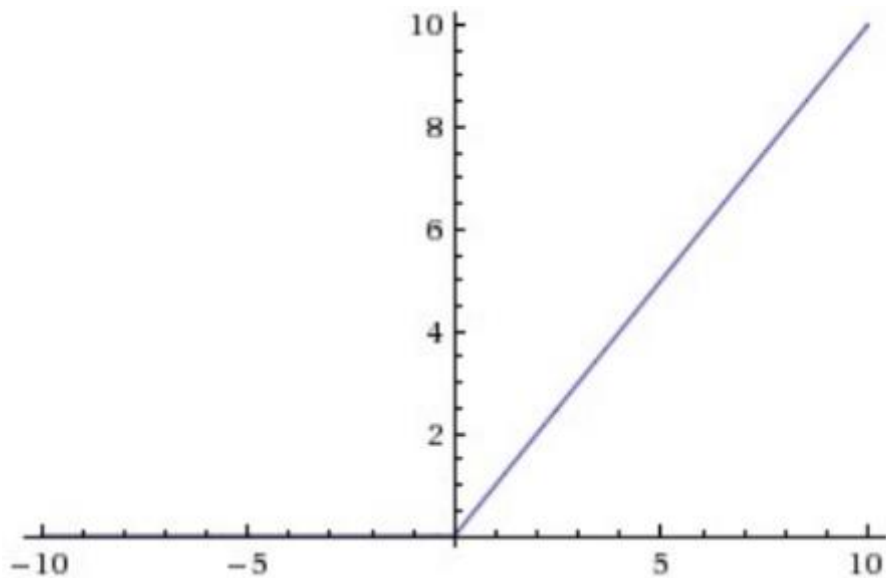


Hình 29: ví dụ Max-pooling



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

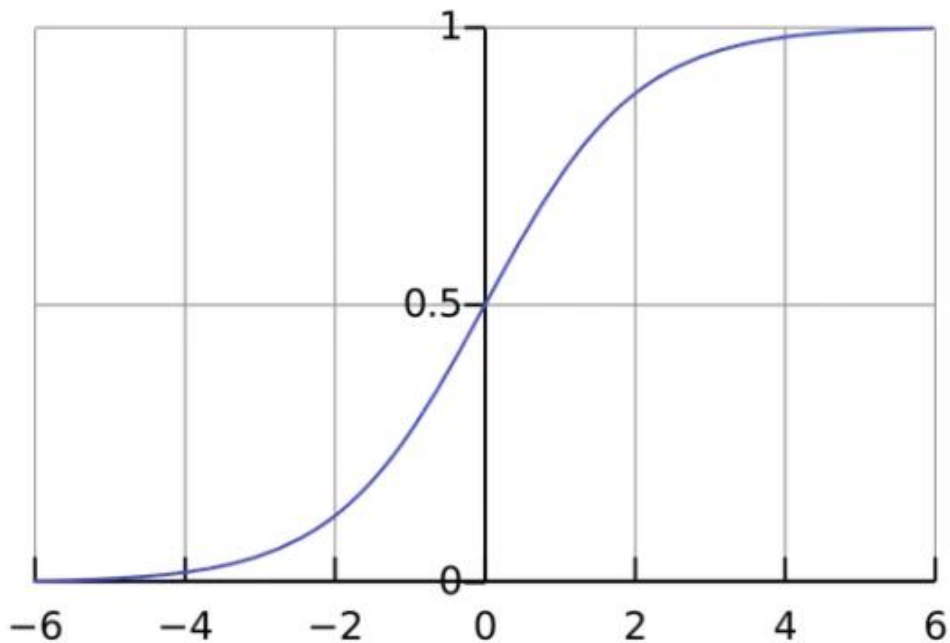
$$f(x) = \max(0, x).$$



Hình 30: Biểu diễn hàm ReLU

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

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



Hình 31: Biểu diễn hàm Softmax

- 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:

$$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$$

Với mỗi lần lặp  $t$ , ta có:

Tính  $dw, db$  ở `mini_batch` hiện tại

$$V_{dw} = \beta_1 * V_{dw} + (1 - \beta_1) * dw$$

$$V_{db} = \beta_1 * V_{db} + (1 - \beta_1) * db$$

$$S_{dw} = \beta_2 * S_{dw} + (1 - \beta_2) * dw^2$$

$$S_{db} = \beta_2 * S_{db} + (1 - \beta_2) * db^2$$

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

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

$$W = W - \alpha * (V_{dw} / \sqrt{S_{dw}} + \epsilon)$$

$$b = b - \alpha * (V_{db} / \sqrt{S_{db}} + \epsilon)$$

Với các siêu tham số:

$\alpha$ : learning\_rate

$\beta_1$ : thường được chọn là 0.9

$\beta_2$ : thường được chọn là 0.999

$\epsilon$ : 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.

Do train lại từ đầu cùng với việc tinh chỉnh tham số nhiều lần và giới hạn GPU của colab nên mất khá nhiều thời gian Train, cụ thể:

- **Training:** Mất hơn 20 giờ để train 100 epoch.
- **Testing:** Mất hơn 1 giờ nữa để test.

Thời gian tìm hiểu, xây dựng, tinh chỉnh và training model mất hơn 2 tuần.

### d. Đánh giá:

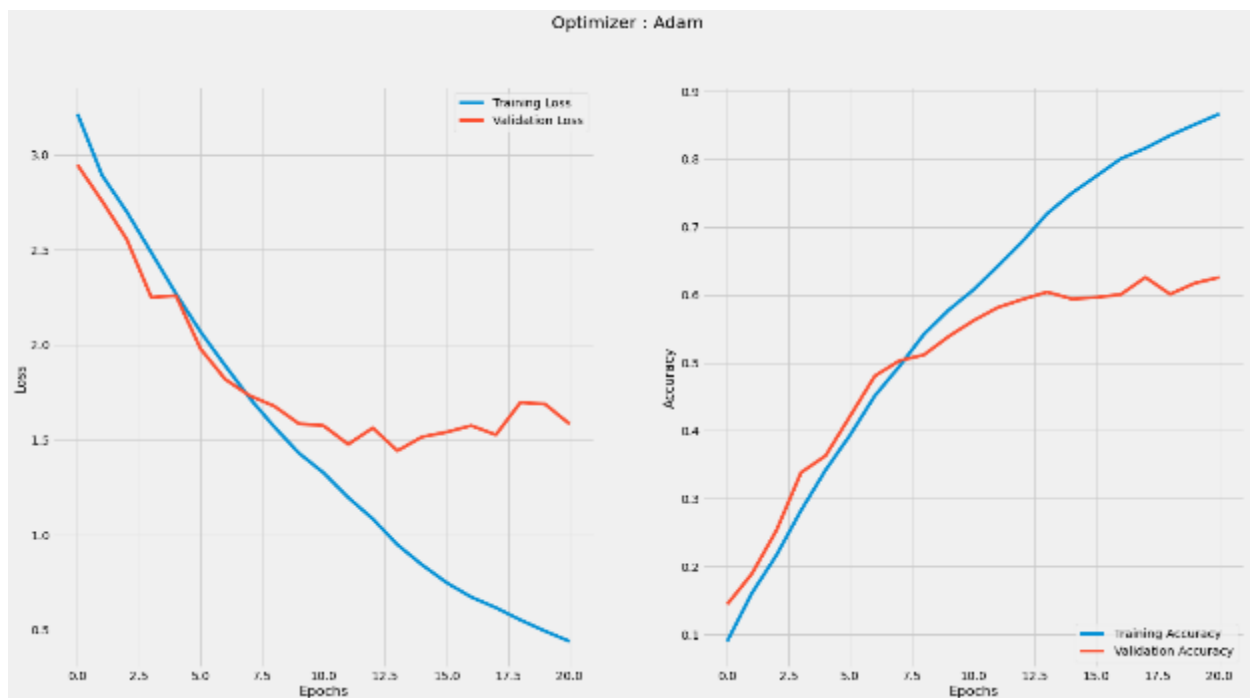
- **Train, Test, Validation accuracy:**

✓ Train accuracy: 0.8729806542396545

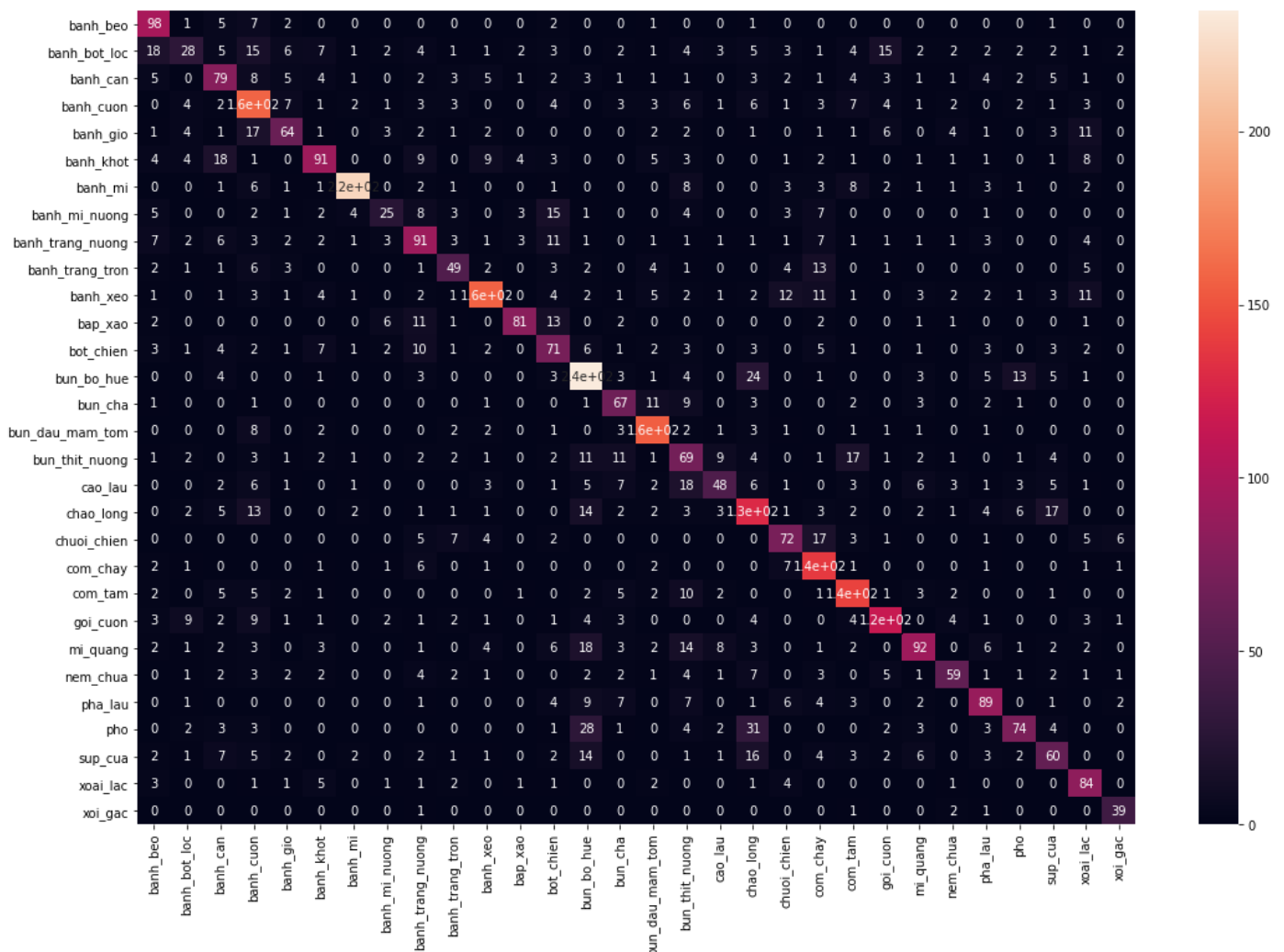
✓ Valid accuracy: 0.6208890676498413

✓ 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, nhầm lẫn các class với nhau, 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:**

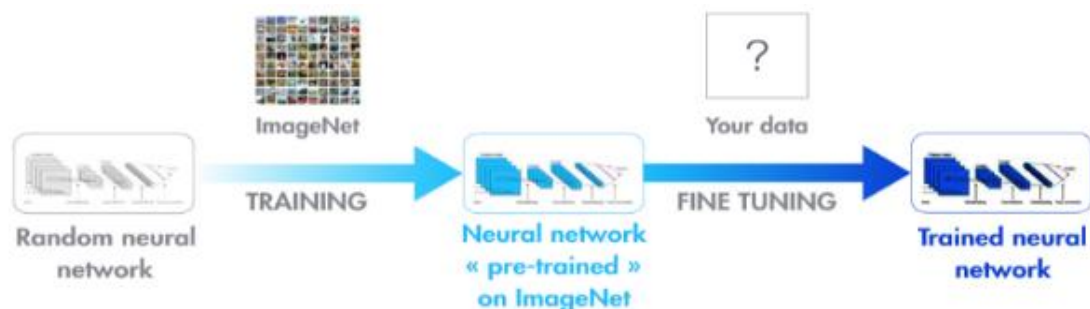
Hình 32: các hình VGG16 predict nhầm lẫn

- ✓ 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.



Hình 33: 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**. Mục đích để giảm số chiều giảm overfit
- Thêm layer **Dropout** để bỏ qua ngẫu nhiên vài unit khi train giảm overfit.
- Thêm layer **Normalization** để chuẩn hóa dữ liệu tăng tốc độ huấn luyện.
- 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. Và kết quả là accuracy của bộ dữ liệu valid đã tăng đáng kể so với VGG16 train ở trên.

Thời gian huấn luyện giảm đáng kể so với VGG16 train từ đầu, cụ thể:

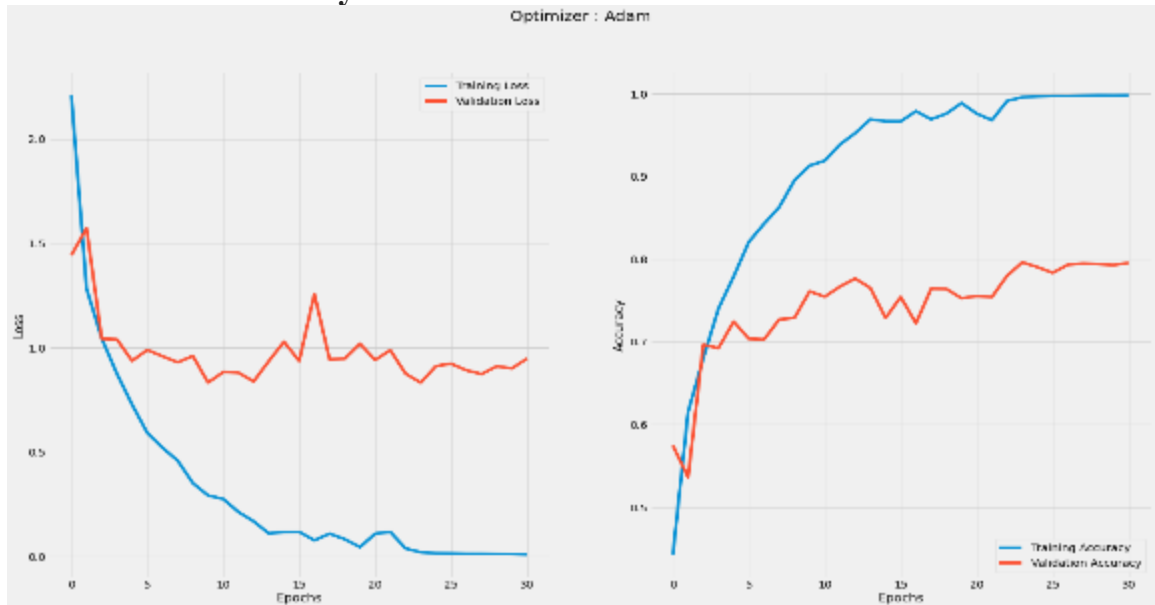
- **Training:** Mất gần 8 tiếng để train 30 epoch thì model hội tụ.
- **Testing:** Mất gần 45 phút.

#### d. Đánh giá:

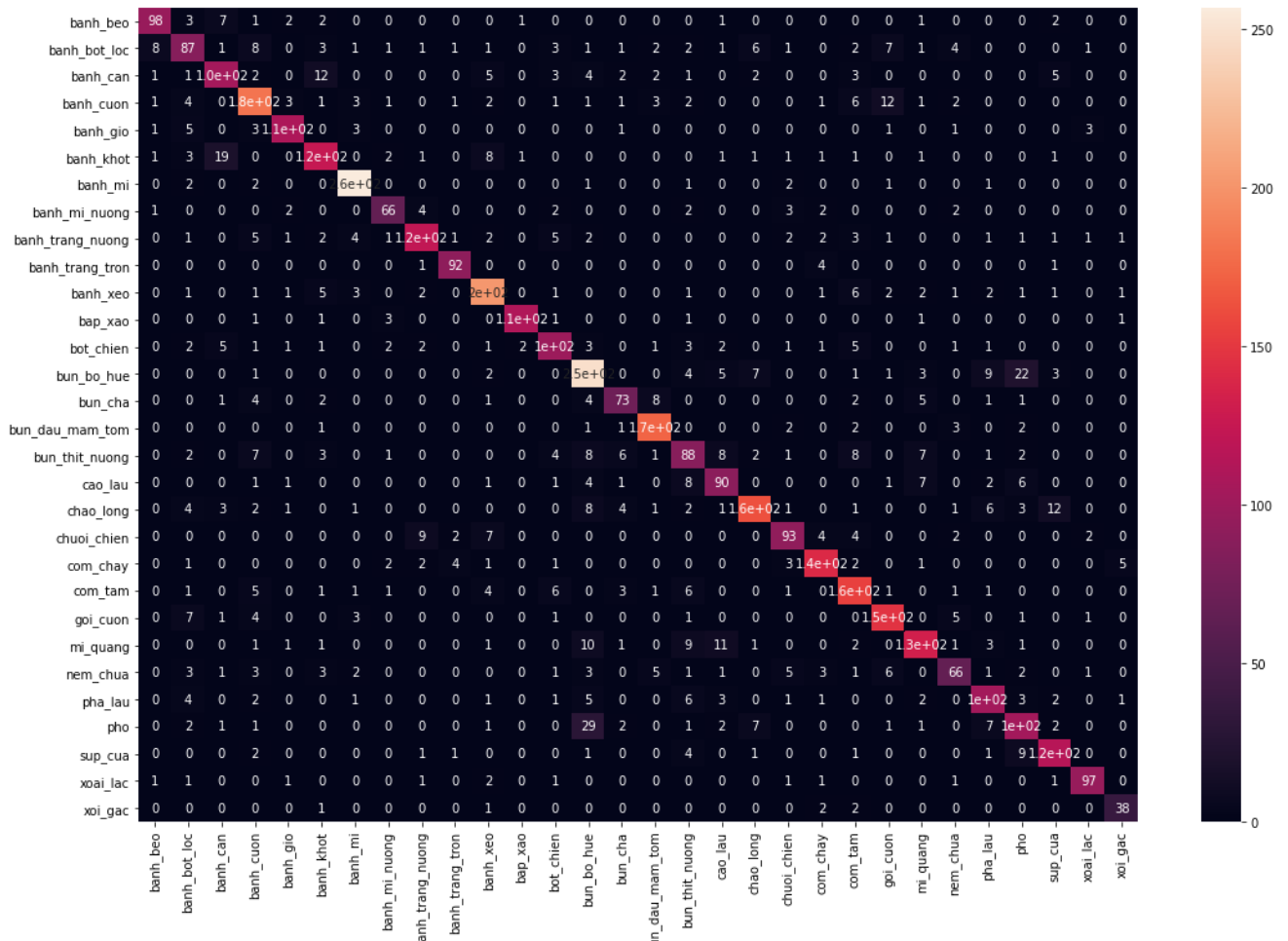
##### - Train, test, validation accuracy:

- ✓ 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
xoai_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:**

Hình 34: Các hình predict nhầm lẫn

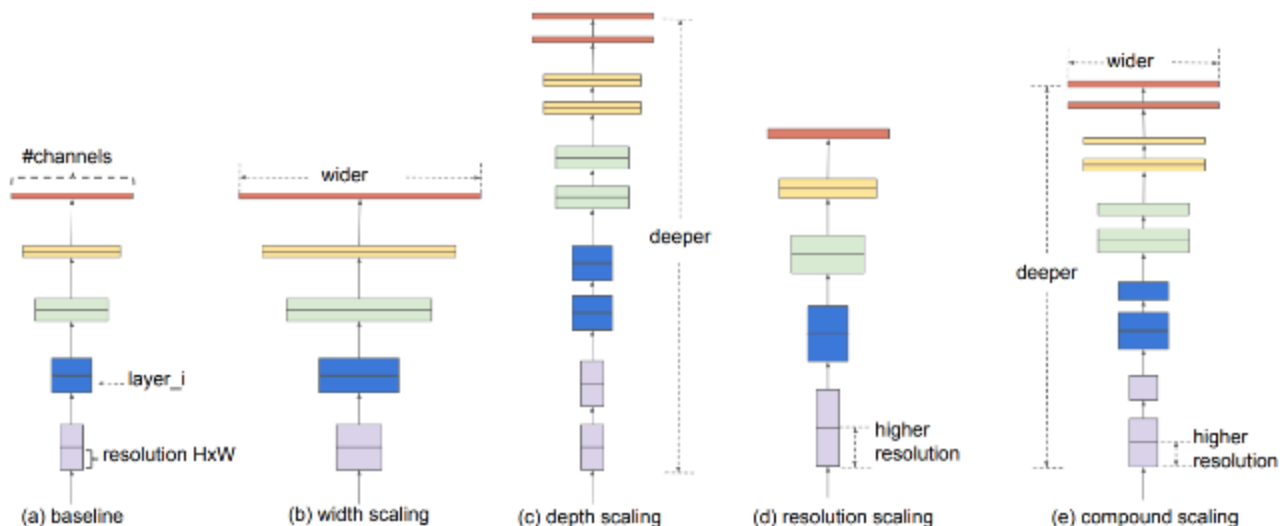
- ✓ So với model VGG16 cũ, model mới đã có sự cải thiện đáng kể về mọi mặt khi áp dụng các kỹ thuật tăng tốc độ học tập cũng như giảm overfitting
- ✓ Tuy nhiên, model này vẫn chưa tối ưu vì accuracy vẫn còn thấp, model vẫn gặp tình trạng overfitting và vẫn còn bị nhầm lẫn các món ăn có sự tương đồng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



Hình 35: Model 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 accuracy trên tập dữ liệu ImageNet là rất cao, bên cạnh đó dữ liệu chúng em sử dụng cũng là dữ liệu hình ảnh 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 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  $\phi$  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
- **$\alpha, \beta, \gamma$**  là các hằng số có thể được xác định bằng small grid search.

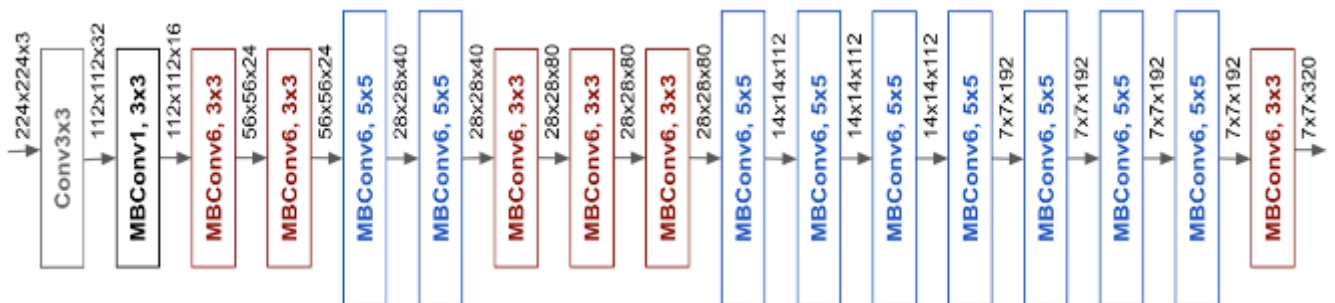
- Theo trực giác,  $\phi$  là hệ số do người dùng chỉ định để kiểm soát số lượng tài nguyên khác có sẵn để thu phóng mô hình, trong khi  **$\alpha, \beta, \gamma$**  chỉ định cách gán các tài nguyên bổ sung này cho độ rộng, độ sâu và độ phân giải của mạng tương ứng. Đáng chú ý, **FLOPS** của một op tích 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  $2^\phi$

### 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**.

Bảng 6: **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	$224 \times 224$	32	1
2	MBConv1, k3x3	$112 \times 112$	16	1
3	MBConv6, k3x3	$112 \times 112$	24	2
4	MBConv6, k5x5	$56 \times 56$	40	2
5	MBConv6, k3x3	$28 \times 28$	80	3
6	MBConv6, k5x5	$14 \times 14$	112	3
7	MBConv6, k5x5	$14 \times 14$	192	4
8	MBConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1

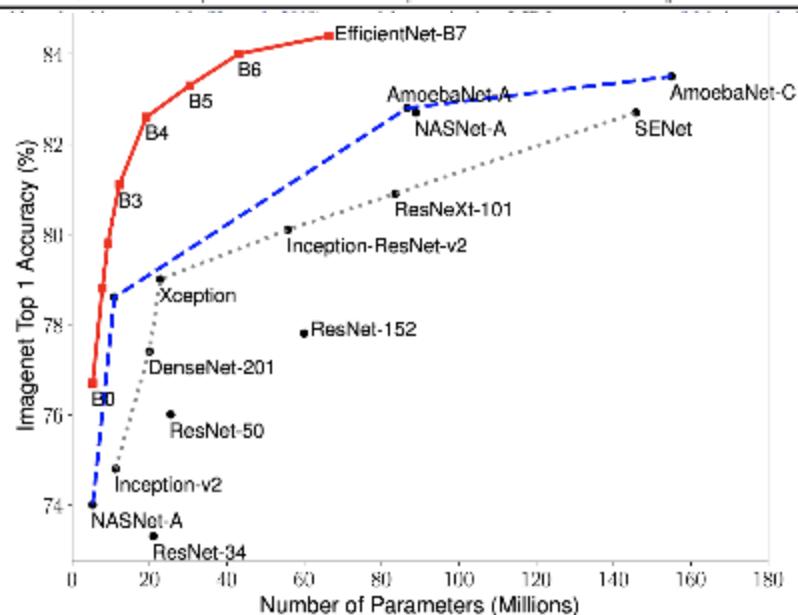


Hình 36: EfficientNet Architecture

- 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  $\phi$  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  $\alpha, \beta, \gamma$  dưới dạng các hằng số và thu phóng mạng cơ sở với các  $\phi$  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
<b>EfficientNet-B0</b>	<b>77.1%</b>	<b>93.3%</b>	<b>5.3M</b>	<b>1x</b>	<b>0.39B</b>	<b>1x</b>
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
<b>EfficientNet-B1</b>	<b>79.1%</b>	<b>94.4%</b>	<b>7.8M</b>	<b>1x</b>	<b>0.70B</b>	<b>1x</b>
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
<b>EfficientNet-B2</b>	<b>80.1%</b>	<b>94.9%</b>	<b>9.2M</b>	<b>1x</b>	<b>1.0B</b>	<b>1x</b>
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
<b>EfficientNet-B3</b>	<b>81.6%</b>	<b>95.7%</b>	<b>12M</b>	<b>1x</b>	<b>1.8B</b>	<b>1x</b>
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
<b>EfficientNet-B4</b>	<b>82.9%</b>	<b>96.4%</b>	<b>19M</b>	<b>1x</b>	<b>4.2B</b>	<b>1x</b>
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
<b>EfficientNet-B5</b>	<b>83.6%</b>	<b>96.7%</b>	<b>30M</b>	<b>1x</b>	<b>9.9B</b>	<b>1x</b>
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
<b>EfficientNet-B6</b>	<b>84.0%</b>	<b>96.8%</b>	<b>43M</b>	<b>1x</b>	<b>19B</b>	<b>1x</b>
<b>EfficientNet-B7</b>	<b>84.3%</b>	<b>97.0%</b>	<b>66M</b>	<b>1x</b>	<b>37B</b>	<b>1x</b>
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-



Bảng 7, Hình 37: So sánh accuracy của EfficientNet với các kiến trúc CNN khác

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

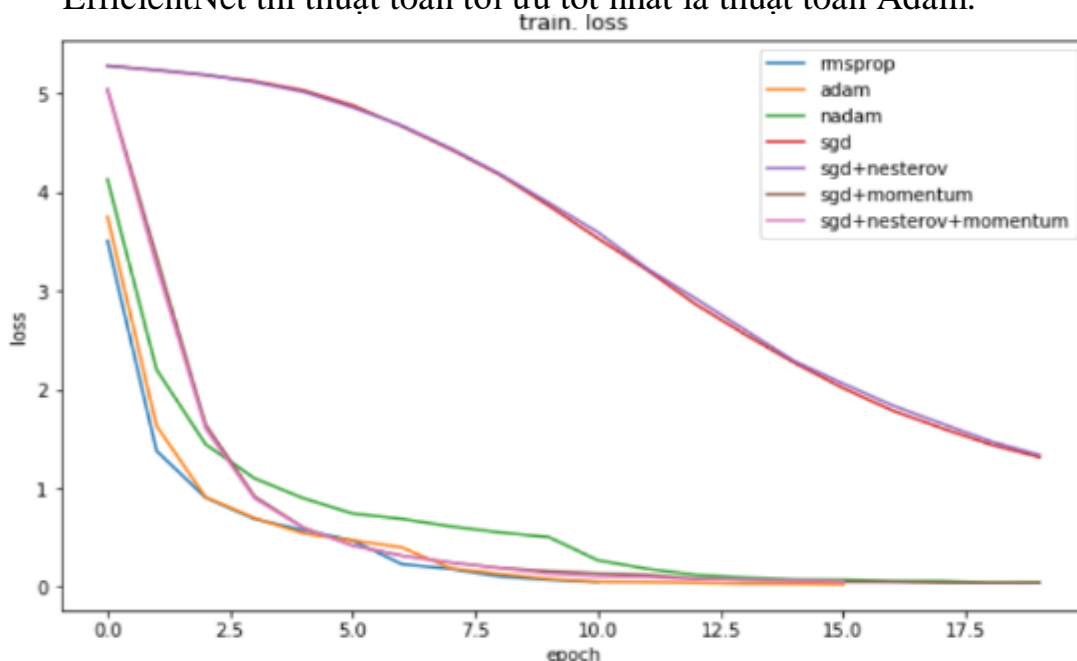
Đây là mô hình mới và tương đối phức tạp trong quá trình tùy chỉnh để có thể tối ưu hơn nữa chúng em đã train rất nhiều cách fine tuning khác nhau và cuối cùng đây là hai cách fine tuning EfficientNet hiệu quả nhất:

#### ❖ 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 dùng **flatten** để dẹt phẳng dữ liệu như VGG16 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 layer** với hàm kích hoạt **Softmax**.

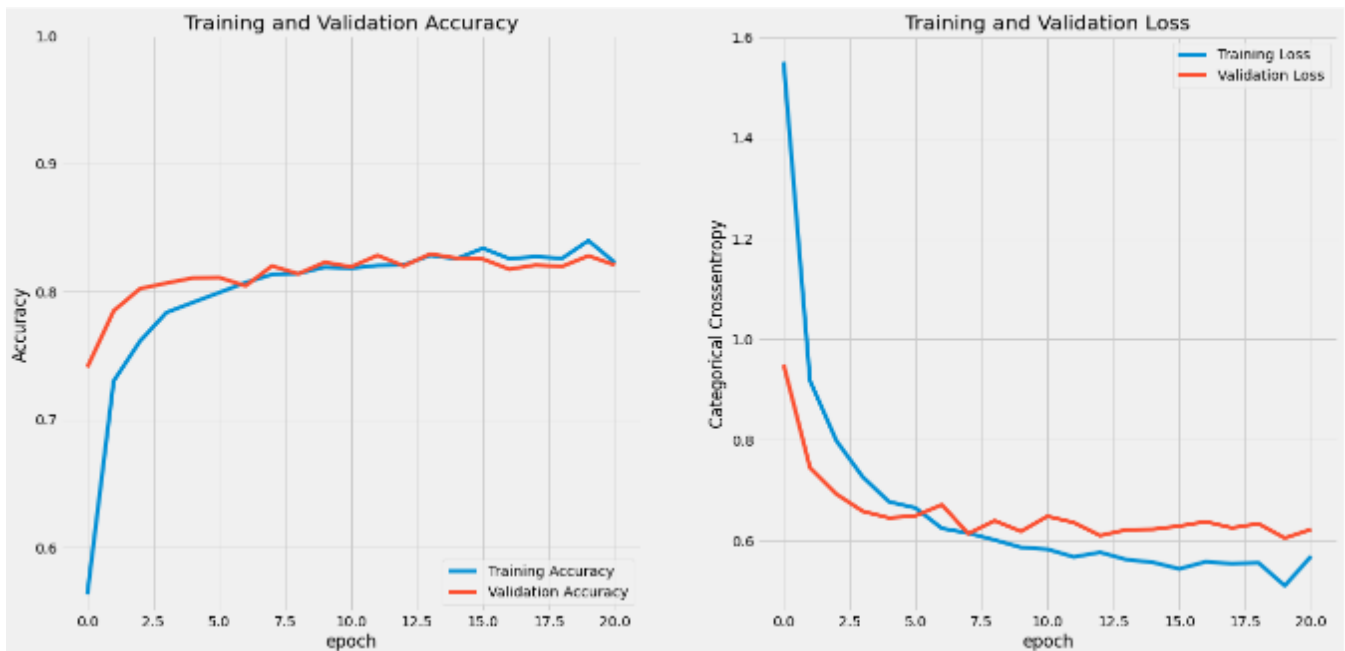
global_average_pooling2d_1 (Glo (None, 1280))	0	top_activation[0][0]
batch_normalization_2 (BatchNor (None, 1280))	5120	global_average_pooling2d_1[0][0]
top_dropout (Dropout)	(None, 1280)	0
dense_2 (Dense)	(None, 30)	38430
		top_dropout[0][0]

- ✓ 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.

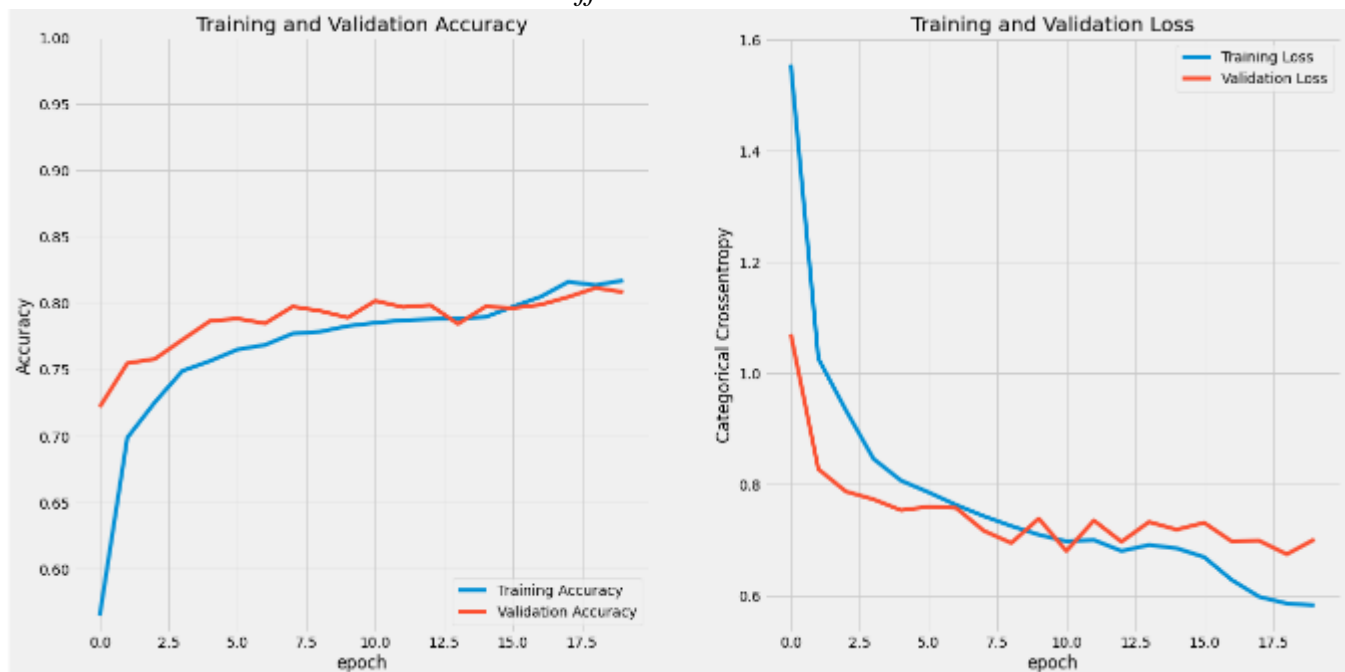


Hình 38: Biểu đồ train loss của EfficientNet với các thuật toán tối ưu khác nhau

✓ Kết quả sau khi train:



*EfficientNet B0*



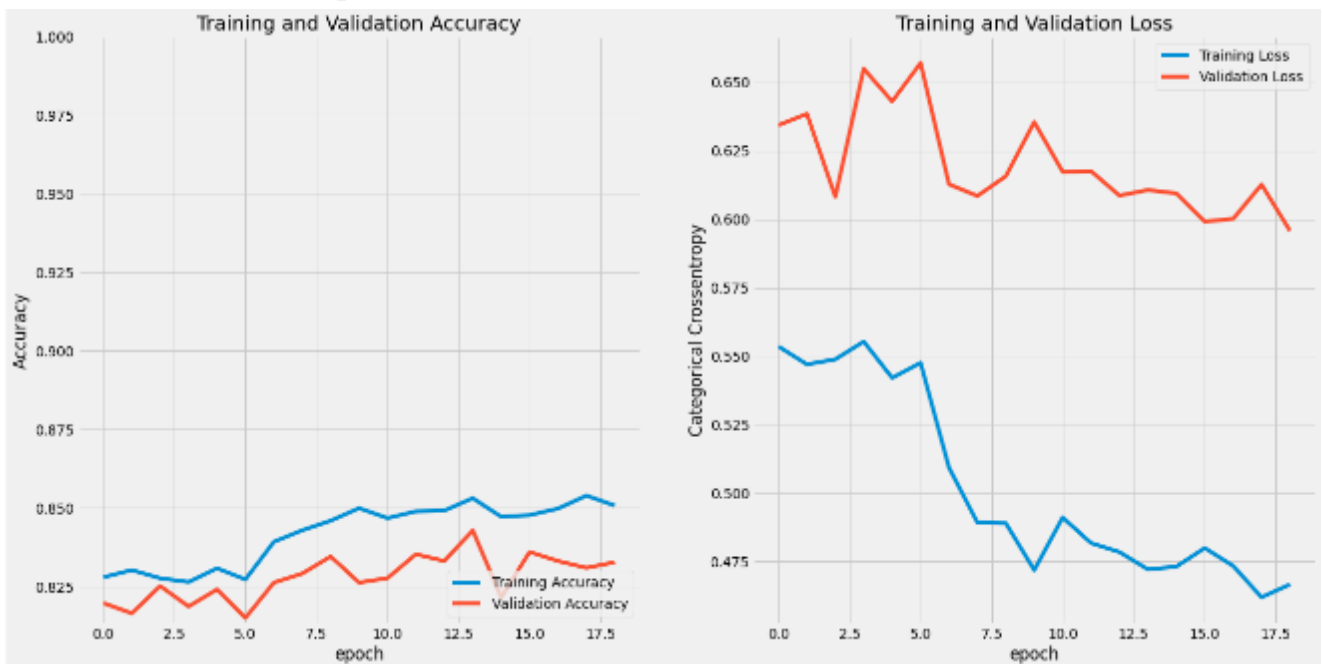
*EfficientNet B4*

- Validation accuracy:
  - EfficientNet B0: 0.82942
  - EfficientNet B4: 0.81171

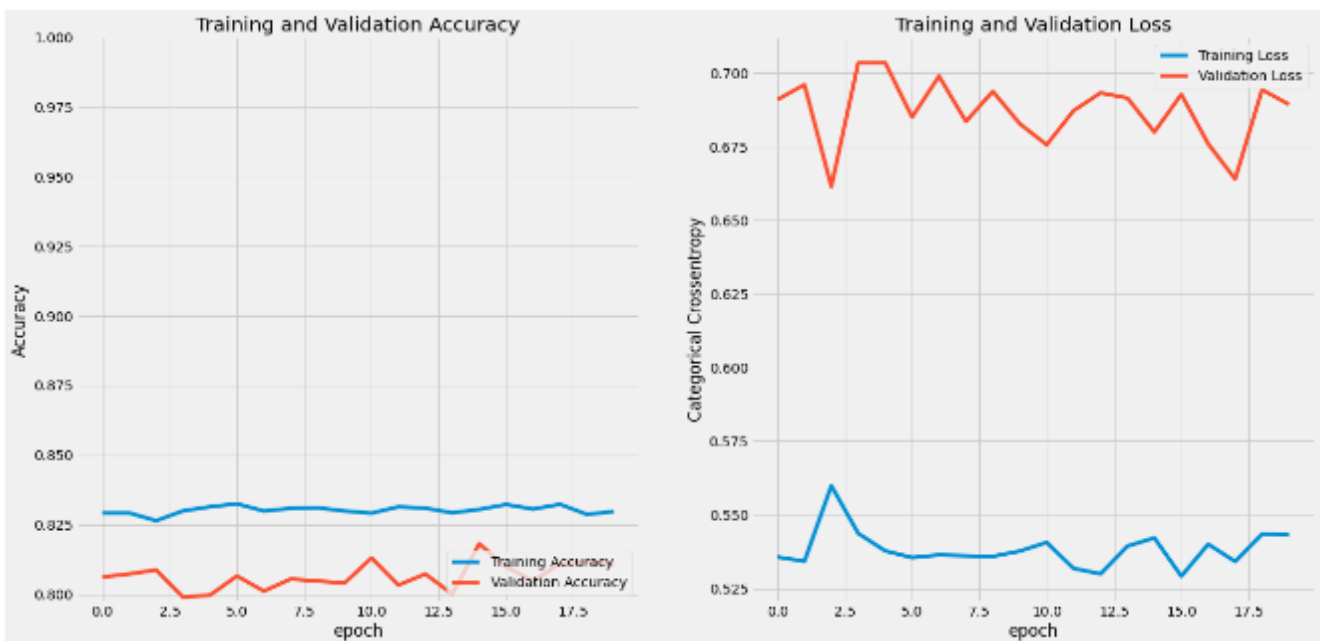
⇒ Cả hai model hội tụ rất nhanh và cơ bản gần được good fitting với accuracy cao hơn so với VGG16, accuracy của EfficientNet B4 thấp hơn so với B0 một chút nhưng thời gian train nhiều hơn rất nhiều.



- **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*

- Validation accuracy:
  - EfficientNet B0: 0.84279
  - EfficientNet B4: 0.81821

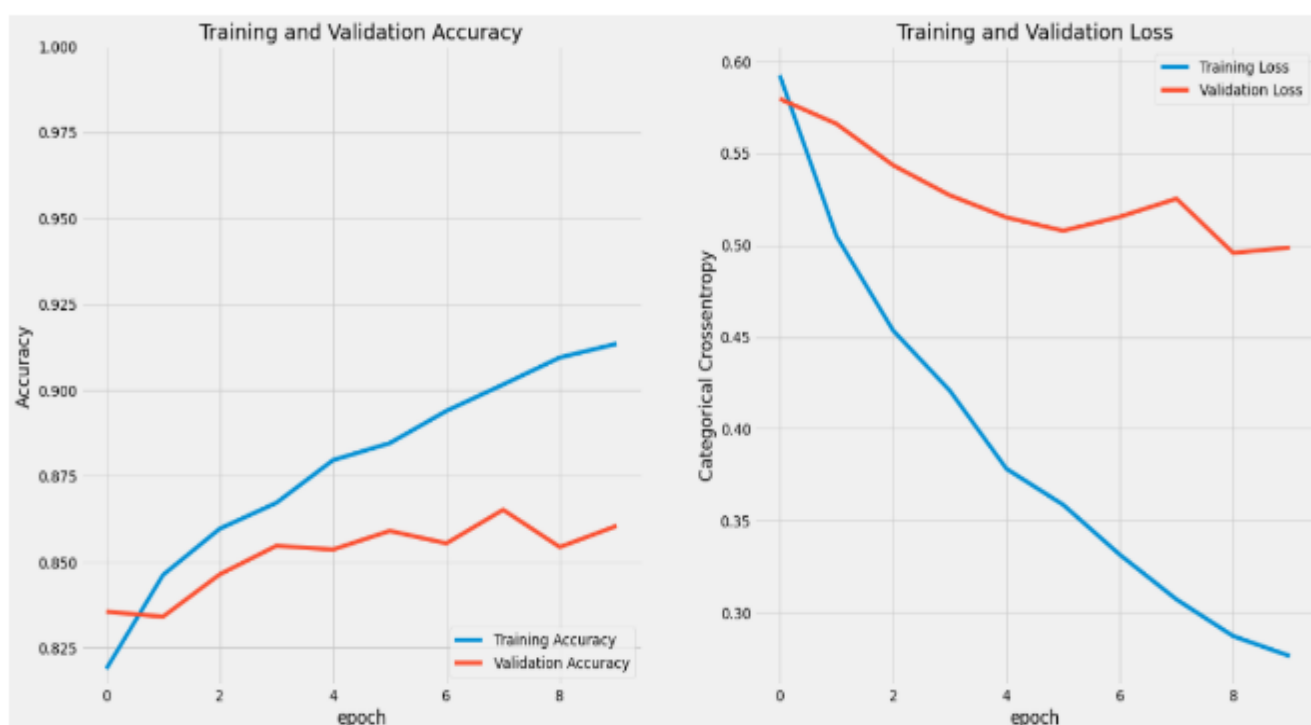
⇒ Nhìn chung Phase 2 không cải thiện nhiều khi val\_accuracy của 2 model tăng rất ít, overfit chưa đáng kể, kết quả của B4 thấp hơn B0.

### ❖ Chiến lược fine tuning 2: 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 và thời gian train lại lâu hơn rất nhiều. Vì thế chiến lược 2 này chúng em sẽ tập trung vào EfficientNet B0 vẫn sử dụng lại Phase 1 của chiến lược 1 và thay đổi cách unfreeze các layer ở Phase 2:

- Thực hiện unfreeze **fine-tune** layer top rồi train thêm 10 epoch nữa với learning rate thấp.
- Sau nhiều lần thay đổi **fine-tune**, nhận thấy fine-tune = 20 tức unfreeze 20-layer top và train là cho kết quả tốt nhất.

Kết quả sau khi train 10 epoch với fine-tune = 20:



- Validation accuracy = 0.86520

⇒ Chiến lược này tốt hơn so với chiến lược trước nhưng vẫn chưa quá nhiều, hiện tượng overfit bắt đầu xuất hiện nhưng chưa đáng kể vì chỉ train 10 epochs.

Chiến lược	Tên kiến trúc	Thời gian (giờ)	
		Training	Testing
Fine tune 1	EfficientNet B0	4.5	0.75
	EfficientNet B4	6	1
Fine tune 2	EfficientNet B0	5	0.75

Bảng 8: thời gian huấn luyện EfficientNet ước tính.

#### d. Đánh giá:

##### - Train, Test, Validation accuracy:

###### ✓ EfficientNet B0 v1:

- Train accuracy: 0.9130820631980896
- Validation accuracy: 0.8366461992263794
- Test accuracy: 0.8476005196571352

###### ✓ EfficientNet B4 v1:

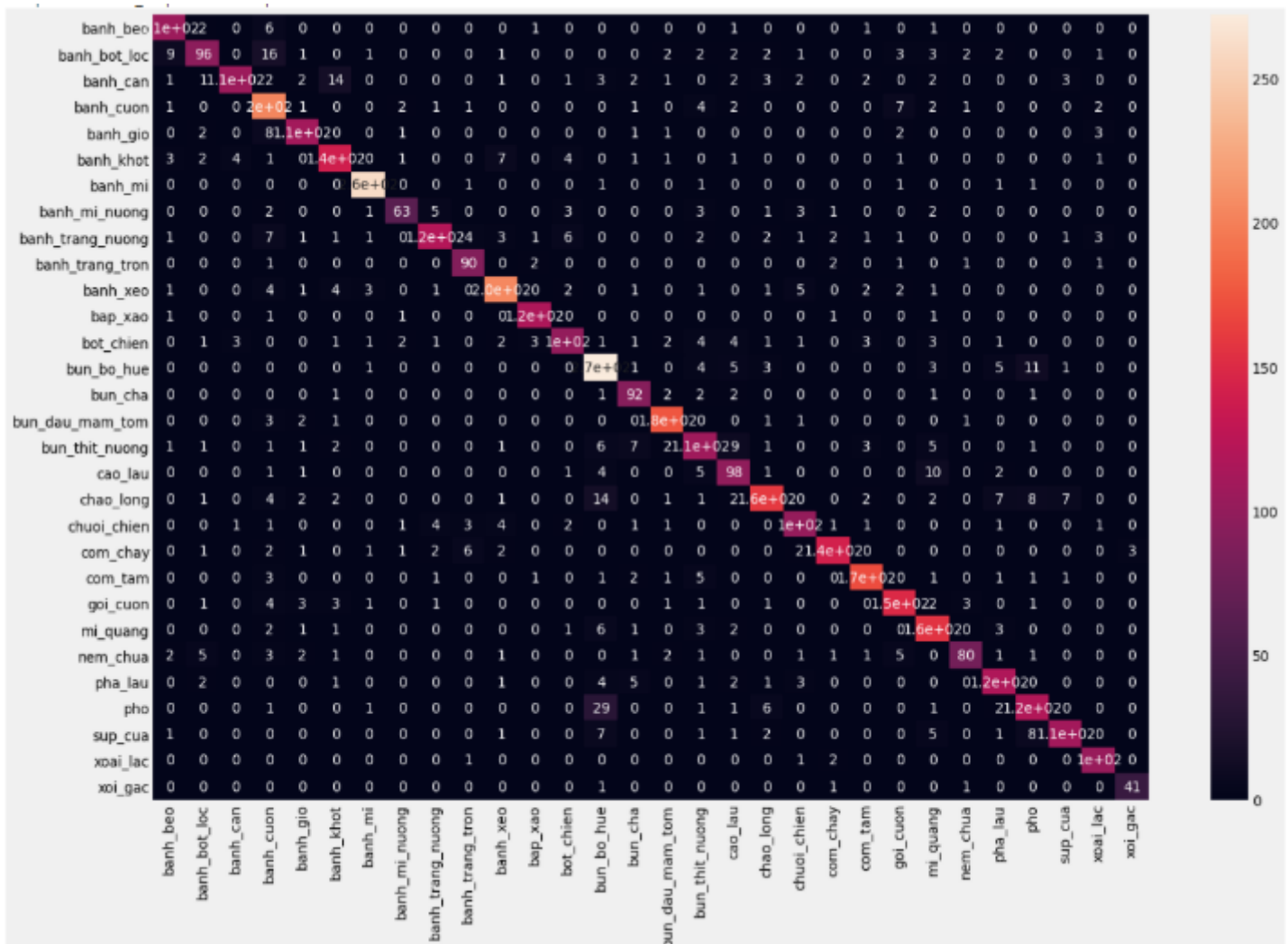
- Train accuracy: 0.8988913297653198
- Validation accuracy: 0.8117094039916992
- Test accuracy: 0.8357111811637878

###### ✓ EfficientNet B0 v2:

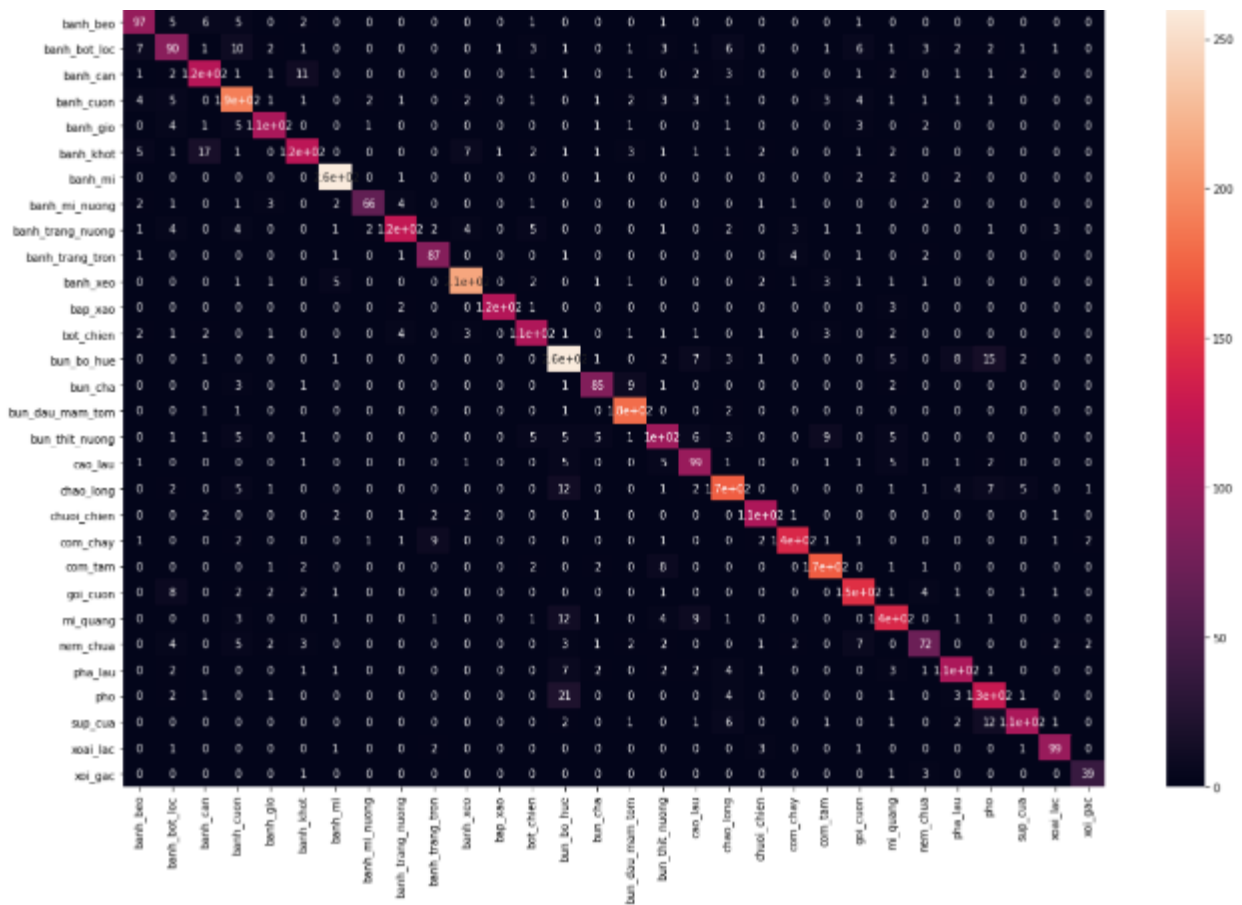
- Train accuracy: 0.9557808041572571
- Validation accuracy: 0.8619443178176882
- Test accuracy: 0.8707306385040283

##### - Confusion matrix:

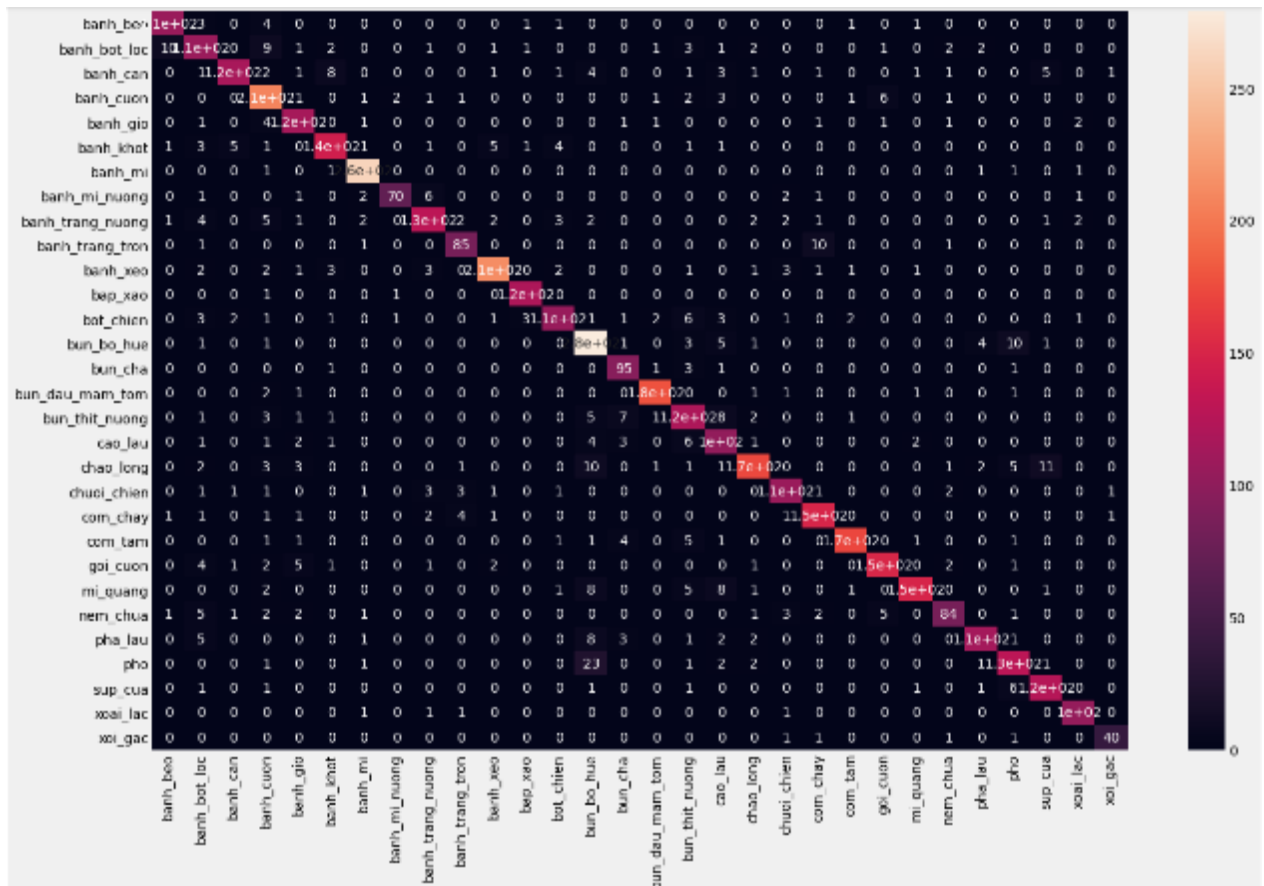
###### ✓ EfficientNet B0 v1:



✓ 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.
- Tổng thời gian tìm hiểu, xây dựng các chiến lược fine tune, tinh chỉnh và huấn luyện mất hơn 3 tuần.

## II. Đánh giá chung:

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

Bảng 9: Accuracy của các thuật toán

- 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.



## CHƯƠNG 5: ỨNG DỤNG VÀ HƯỚNG PHÁT TRIỂN.

### I. Xây dựng ứng dụng web:

#### 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.

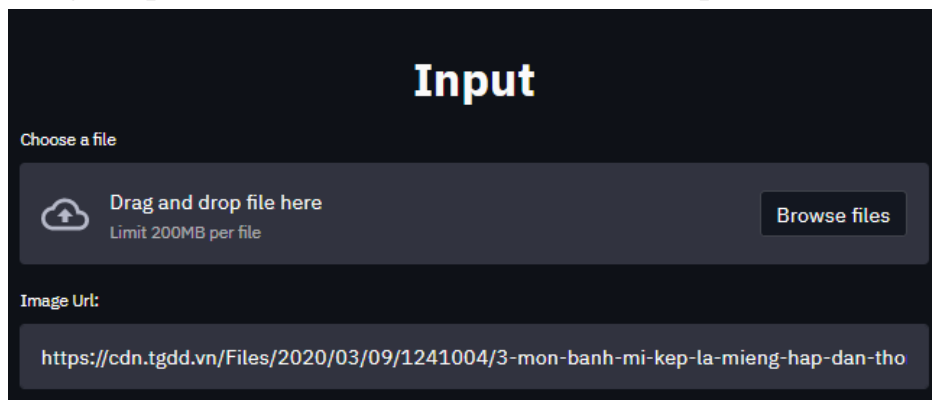
#### 3. Thiết kế web-app:

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.

Hình 39: ví dụ về các công upload ảnh



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



Hình 40: Mô tả input, output của web-app

### 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 banh mi 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 Bầy Hồ: 19 Huỳnh Khương Ninh St, Đa Kao Ward, District 1, HCM City
- Bánh mì thịt nướng Nguyễn Trãi: 37 Nguyễn Trãi St, Bến Thành Ward, District 1, HCM City

Price: 10.000-50.000 VND

Probability: 100.00%

## 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.



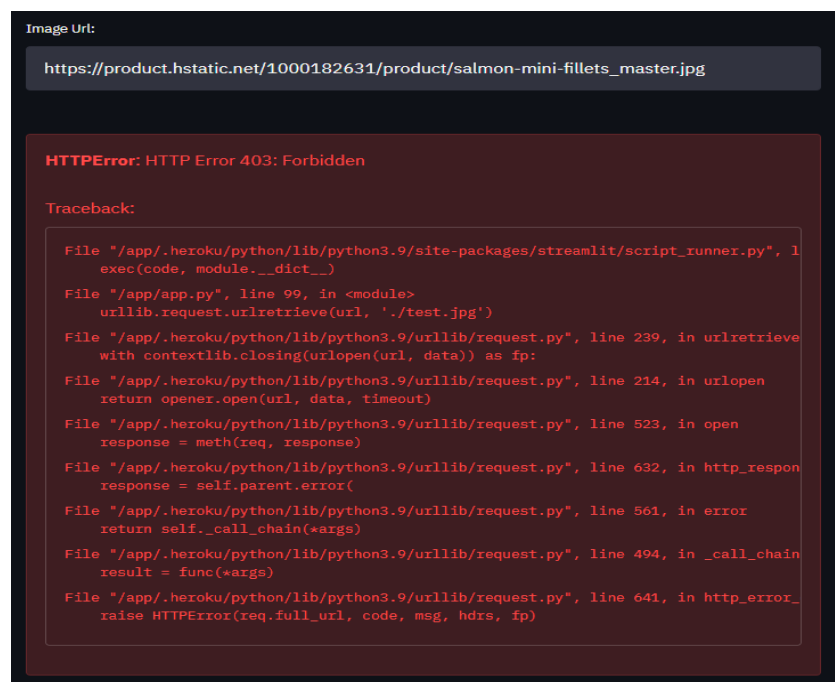
Hình 41: Logo Heroku

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

## 5. Thử nghiệm và nhận phản hồi:

Sau khi deploy ứng dụng chúng em tiến hành thử nghiệm với 30 tình nguyện viên (TNV) mỗi người đều thử truy cập vào webapp trên nhiều thiết bị, nền tảng, trình duyệt khác nhau và nhận dạng 20 ảnh bất kỳ họ tự chọn về các món ăn trong danh sách và 10 ảnh món ăn ngoài danh sách. Nhận được các phản hồi sau:

- Hơn 90% TNV cho rằng ứng dụng có thời gian truy cập lần đầu khá lâu và khác nhau giữa các trình duyệt, cụ thể:
  - Đối với **Google chrome** thời gian truy cập lần đầu lâu nhất khoảng 10-15 giây, các lần sau khoảng 2-3 giây;
  - **Microsoft Edge** thời gian truy cập lần đầu khoảng 5-7 giây, các lần sau khoảng 1-2 giây;
  - Tương tự với các thiết bị smartphone, tablet (Android & IOS).
- Hơn 15% TNV bị lỗi HTTP Forbidden ở một vài ảnh khi lấy url ảnh từ GG Chrome



Hình 42: Lỗi HTTP Forbidden

- **Về thời gian** kể từ khi paste url hoặc từ lúc đã upload xong ảnh tức không kể thời gian upload ảnh thì mất khoảng 3-5 giây để nhận dạng 1 bức ảnh.
- **Về kết quả nhận dạng:**
  - ✓ 90% TNV cho rằng app nhận dạng chính xác với tất cả hình của họ,
  - ✓ 10% còn lại cho biết ứng dụng nhận dạng sai và chủ yếu sai ở các hình ngoài danh sách 30 món ăn của nhóm.
- ❖ Mất hơn 3 tuần để tìm kiếm thông tin món ăn, tìm hiểu về làm web app, xây dựng, tinh chỉnh, tìm môi trường deploy, thêm 1 tuần nữa để thử nghiệm và nhận phản hồi, chỉnh sửa thì web app mới cơ bản hoàn thiện.

**Nhận xét:**

- Do chạy trên nền tảng web dạng dọc với server đám mây nên ứng dụng có thể chạy được trên đa dạng các thiết bị trình duyệt cũng như hệ điều hành khác nhau.
- Thời gian truy cập còn ảnh hưởng bởi tốc độ Internet và loại trình duyệt.
- Một số ảnh ngoài danh sách bị dự đoán sai. Bị lỗi ở một vài url cá biệt.
- Còn lại nhìn chung ứng dụng hoạt động ổn định, đáp ứng đủ tính năng đặc được đặc ra, tận dụng tốt model được huấn luyện.

**II. Hướng phát triển:**

Qua quá trình thực hiện đồ án với những kết quả đạt được cùng nhu cầu của thị trường em nhận thấy đề tài *Nhận dạng món ăn Việt* có khả năng phát triển cao.

Với đồ án trên chúng ta có thể tối ưu và mở rộng hơn bộ dữ liệu để có thể nhận dạng được nhiều món ăn hơn. Từ việc nhận dạng một ăn chúng ta có thể phát triển đề nhận dạng nhiều món hơn trong một bức ảnh.

Về phần ứng dụng tương đối ổn định nhưng cũng có thể phát triển thêm để ứng dụng ổn định hơn và thời gian tương tác với ứng dụng nhanh hơn. Hoặc có thể phát triển thành ứng dụng di động

## **TÀI LIỆU THAM KHẢO**

### ❖ **Thu thập data và tiền xử lý:**

#### **[1] TensorFlow - ImageDataGenerator**

Link: [https://www.tensorflow.org/api\\_docs/python/tf/keras/preprocessing/image/ImageDataGenerator?fbclid=IwAR1f6cxh-qCr1AMsr7nvy-Dmiewd0Y2OnXLERqZ5OWn5ZxdxILqzW6ndo7g](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator?fbclid=IwAR1f6cxh-qCr1AMsr7nvy-Dmiewd0Y2OnXLERqZ5OWn5ZxdxILqzW6ndo7g)

### ❖ **SVM và HOG:**

#### **[2] Hải Hà – 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-V3m5WAwXZO7>

#### **[3] SVC model - Scikit learn:**

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

#### **[4] Machinelearningcoban.com – Thuật toán SVM**

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

#### **[5] Machinelearningcoban.com – Soft margin SVM**

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

#### **[6] Baeldung.com - Multiclass Classification Using Support Vector Machines**

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

### ❖ **VGG16:**

#### **[7] neurohive.io - VGG16 Convolutional Network for Classification & Detection**

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

#### **[8] 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:**

#### **[9] 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/>

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

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



[11] tensorflow – Transferlearning and fine tuning

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

❖ **EfficientNet:**

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

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

[13] 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>

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

Link: [https://keras.io/examples/vision/image\\_classification\\_efficientnet\\_fine\\_tuning/](https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/)

[15] 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](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)

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

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

❖ **Adam với EfficientNet:**

[17] kaggle.com - Compare optimizer of efficientNet

Link: <https://www.kaggle.com/kimtaegwan/compare-optimizer-of-efficientnet>

[18] Phan Ngoc – Thuật toán tối ưu Adam

Link: <https://viblo.asia/p/thuat-toan-toi-uu-adam-aWj53k8Q56m>

❖ **Xây dựng web app với Streamlit và Heroku:**

[19] topdev.vn - Heroku là gì? Cách đưa ứng dụng lên Heroku

Link: <https://topdev.vn/blog/heroku-la-gi/>

[20] Bui Tien Tung - Xây dựng web app cực xịn xò cho project của bạn chỉ với 10 dòng code

Link: <https://viblo.asia/p/xay-dung-web-app-cuc-xin-xo-cho-project-cua-ban-chi-voi-10-dong-code-djeZ1wym5Wz>

[21] App demo - Chris Tran

Link: <https://github.com/chriskhanhtran/vn-food-app/blob/master/app.py>