

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



NHẬP MÔN HỌC MÁY

BÁO CÁO ĐỒ ÁN 02

PHÂN LỚP BẰNG MẠNG NƠ-RON

-- Giảng viên lý thuyết --

Bùi Tiến Lên

-- Giảng viên hướng dẫn đồ án 02 --

Lê Thanh Phong

TP. Hồ Chí Minh, tháng 6 năm 2021

MỤC LỤC

I-	THÔNG TIN NHÓM VÀ ĐÁNH GIÁ ĐỒ ÁN	3
1.	Thông tin thành viên, phân công công việc và đóng góp.....	3
2.	Đánh giá mức độ hoàn thành đồ án	3
II-	NỘI DUNG	3
1.	Phân tích bài toán	3
2.	Phân tích và tiền xử lý tập dữ liệu.....	4
2.1.	Phân tích tập dữ liệu	4
2.2.	Tiền xử lý tập dữ liệu.....	6
3.	Phân tích các kiến trúc neural network dùng để giải quyết bài toán.....	6
3.1.	Multi-layer Perceptron.....	6
3.2.	VGG-16 và VGG-19	8
3.3.	Residual Network (ResNet).....	10
3.4.	Inception Network (InceptionNet)	14
3.5.	EfficientNet.....	17
4.	Báo cáo kết quả và nhận xét.....	19
4.1	Báo cáo kết quả	19
4.2	Nhận xét.....	24
III-	TÀI LIỆU THAM KHẢO	25

I- THÔNG TIN NHÓM VÀ ĐÁNH GIÁ ĐỒ ÁN

1. Thông tin thành viên, phân công công việc và đóng góp

Số thứ tự nhóm: **10**

STT	MSSV	Họ và tên	Phân công công việc	Đóng góp vào công việc
1	18120066	Bùi Đoàn Hữu Nhân	Phân tích và cài đặt mạng nơ-ron MLP. Tổng hợp và trình bày báo cáo.	100%
2	18120085	Nguyễn Tấn Thìn	Phân tích và cài đặt mạng nơ-ron EfficientNet. Phân tích tập dữ liệu.	100%
3	18120090	Phạm Nguyên Minh Thy	Phân tích và cài đặt mạng nơ-ron ResNet. Báo cáo kết quả và nhận xét.	100%
4	18120097	Đinh Hữu Phúc Trung	Phân tích và cài đặt mạng nơ-ron InceptionNet. Tiền xử lý tập dữ liệu.	100%
5	18120649	Nguyễn Phạm Phúc Việt	Phân tích và cài đặt mạng nơ-ron VGG. Phân tích bài toán.	100%

2. Đánh giá mức độ hoàn thành đồ án

STT	Yêu cầu	Mức độ hoàn thành
1	Phân tích kỹ bài toán và tập dữ liệu hình ảnh được cung cấp. Chọn lựa và trình bày kiểu mạng nơron để giải quyết bài toán.	100%
2	Cài đặt mạng Nơron.	100%
3	Báo cáo kết quả đạt được sau quá trình phân tích và cài đặt.	100%
Tổng		100%

II- NỘI DUNG

1. Phân tích bài toán

Phát biểu bài toán

Mục tiêu của bài toán là xây dựng một mô hình (model) có thể phân loại chính xác hình ảnh lá cây vào nhóm khỏe mạnh hoặc vào các nhóm bệnh tương ứng.

Một số yếu tố ảnh hưởng đến độ chính xác của mô hình như:

- Điều kiện môi trường biến thiên về ánh sáng, góc nhìn, lá cây với nhiều hình dáng, độ tuổi khác nhau.
- Sự tương đồng giữa các triệu chứng bệnh. Ngoài ra, một lá có thể xuất hiện kết hợp nhiều loại bệnh thay vì chỉ có loại bệnh với một triệu chứng rõ ràng.

Ý nghĩa thực tiễn của bài toán

Việc chẩn đoán bệnh trên cây trong có ý nghĩa lớn trong nông nghiệp, giúp người trồng phát hiện và có những biện pháp phù hợp để phòng tránh thiệt hại cho những mầm bệnh này gây ra. Những phương pháp chẩn đoán bệnh hiện tại dựa trên con người tốn nhiều thời gian và chi phí để thực hiện. Do đó những mô hình dự đoán bệnh dựa trên hình ảnh hứa hẹn sẽ cải thiện hiệu suất công việc, tiết kiệm thời gian và công sức con người mà vẫn mang lại kết quả tốt.

Cách tiếp cận bài toán

Bài toán liên quan đến việc phân loại dữ liệu dữ liệu phi cấu trúc (unstructured data) dạng hình ảnh, vì vậy ta có thể nghĩ tới việc sử dụng các mạng nơ-ron tích chập (Convolutional Neural Network) cho việc giải quyết bài toán này. Đồng thời ta cũng sử dụng một mạng perceptron đa lớp (Multi-layer Perceptron) để có cái nhìn rõ hơn về hiệu năng của các mạng này trong việc giải quyết bài toán phân loại ảnh.

2. Phân tích và tiền xử lý tập dữ liệu

2.1. Phân tích tập dữ liệu

Thống kê tỉ lệ các lớp trong tập train:

Loại lá	Healthy	Multiple_Diseases	Rust	Scab
Số lượng	516	91	622	592
Tổng	1821			

Nhận xét về hình dáng, đặc trưng lá các ảnh thuộc các trường hợp:

- Healthy: đây là các lá bình thường, có màu xanh đều, khá trơn, không có dấu hiệu lạ



- Multiple Diseases: đây là các lá bị một hoặc nhiều bệnh như rust, scab, có thể xuất hiện những đốm rỉ sét hoặc những mảng, chấm nâu, đen



- Rust: đây là các lá có xuất hiện các đốm rỉ sét



- Scab: đây là các lá xuất hiện các mảng màu xám đen



2.2. Tiền xử lý tập dữ liệu

Chia tập train và validation theo tỉ lệ 8:2. Chuẩn hóa dữ liệu train và validation về dạng chuẩn trong khoảng $[0,1]$ bằng cách chia cho 255.

Số lượng ảnh sử dụng cho training quá ít dễ dẫn đến hiện tượng overfit trên tập huấn luyện. Do đó cần phải sinh thêm data để model có thể học được tốt hơn. Ở đây ta sử dụng class ImageDataGenerator để sinh thêm data với các phép augmentation như là xoay, lật, trượt và phóng to, thu nhỏ ảnh.

Dùng 150 epochs để train model CNN đồng nghĩa với việc sử dụng $1821 \times 0.8 \times 150 = 218520$ dữ liệu train với ImageDataGenerator sinh data realtime trong khi train mỗi epoch.

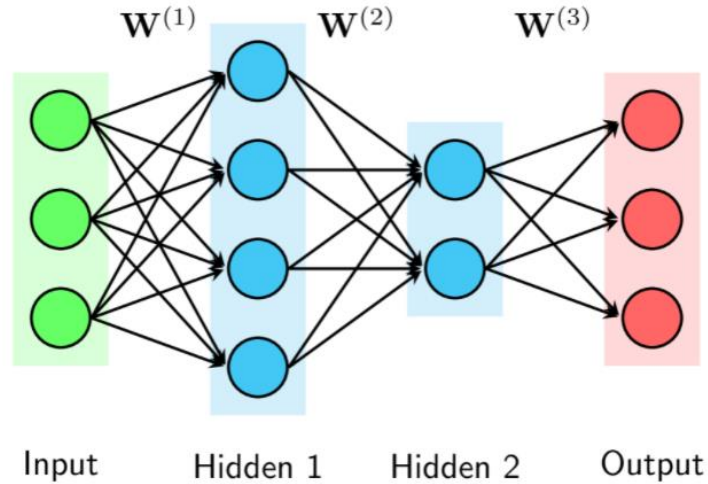
3. Phân tích các kiến trúc neural network dùng để giải quyết bài toán

Phần này trình bày kiến trúc cơ bản nhất của những neural network mà nhóm đã sử dụng như: MLP, VGG, ResNet, InceptionNet, EfficientNet.

3.1. Multi-layer Perceptron

Multilayer-Perceptron (MLP) là mạng nơ-ron nhân tạo đơn giản, xuất hiện trước các kiến trúc mạng nơ-ron phức tạp phục vụ cho những mục đích đặc thù về xử lý hình ảnh hay ngôn ngữ tự nhiên như CNN, LSTM, ...

MLP còn được gọi là multi-layer neural network (mạng nơ-ron đa tầng). MLP thường bao gồm tầng input, tầng output và những tầng ẩn ở giữa. Các tầng được liên kết đầy đủ với nhau. Hình bên dưới mô tả một MLP với 2 tầng ẩn.

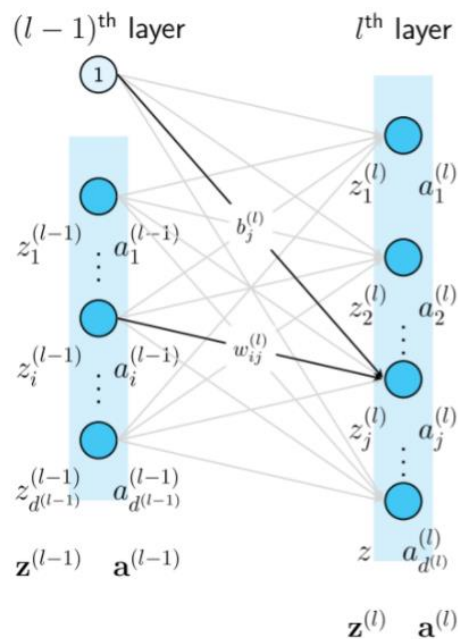


Tầng (Layer)

Như đã nói ở trên, MLP gồm 1 tầng input, 1 tầng output và những tầng ẩn ở giữa. Số tầng của mạng MLP ký hiệu là L . Khi đếm số tầng của mạng, ta không tính tầng input. Ví dụ mạng MLP ở hình trên có số tầng $L = 3$

Nút (Unit)

Mỗi hình tròn trong hình biểu diễn cho một nút.



Tại tầng thứ l gồm có nhiều nút. Ở nút thứ j , đầu vào cho nút này là $z_j^{(l)}$, đầu ra là $a_j^{(l)}$ sau khi đi qua hàm kích hoạt. Vector đầu vào của tầng thứ l là $\mathbf{z}^{(l)}$ và vector đầu ra là $\mathbf{a}^{(l)}$

Trọng số (Weight) và hệ số điều chỉnh (Bias)

Mỗi tầng đều có ma trận trọng số riêng. Chẳng hạn ma trận $\mathbf{W}^{(l)}$ thể hiện kết nối từ tầng $(l-1)$ đến tầng l . Hệ số điều chỉnh của tầng l là $\mathbf{b}^{(l)}$.

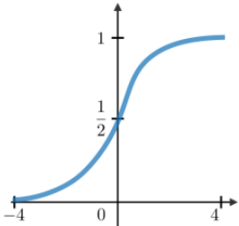
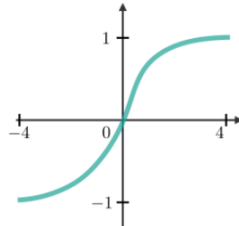
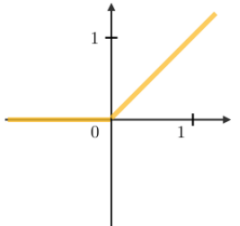
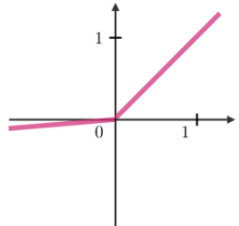
Hàm kích hoạt (Activation function)

Đầu ra tại một tầng được tính theo công thức

$$a^{(l)} = f^{(l)}(z^{(l)}) = f^{(l)}(W^{(l)T} a^{(l-1)} + b^{(l)})$$

với $f^{(l)}(\cdot)$ là một hàm kích hoạt phi tuyến.

Các hàm kích hoạt thường dùng là:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

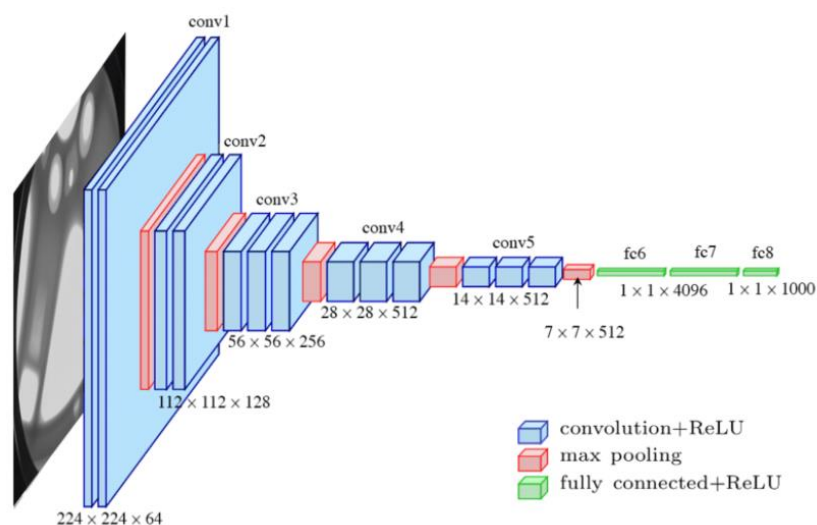
Trong đó hàm ReLU hiện nay được sử dụng rộng rãi vì đơn giản trong tính toán và giúp cho việc huấn luyện nhanh hơn rất nhiều.

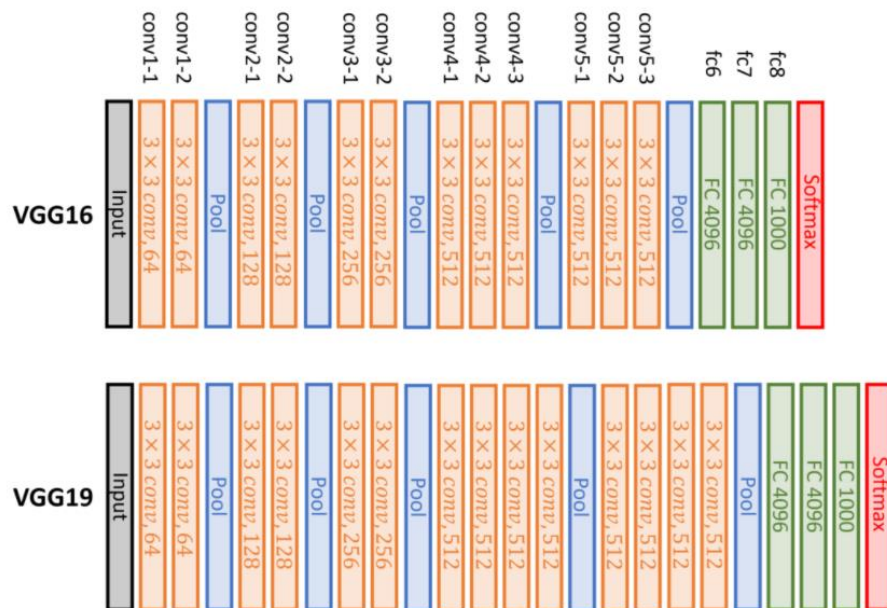
Lan truyền ngược (Backpropagation)

Phương pháp phổ biến dùng để tối ưu mạng MLP là Gradient descend.

3.2. VGG-16 và VGG-19

VGG là kiến trúc mạng CNN cổ điển được đề xuất bởi K. Simonyan và A. Zisserman trong paper “Very Deep Convolutional Network for Large-Scale Image Recognition”. VGG hướng tới một mạng nơ-ron đơn giản nhưng tăng về chiều sâu. VGG chủ yếu sử dụng filters có kích thước nhỏ 3x3, cùng với pooling layers và fully connected layers.



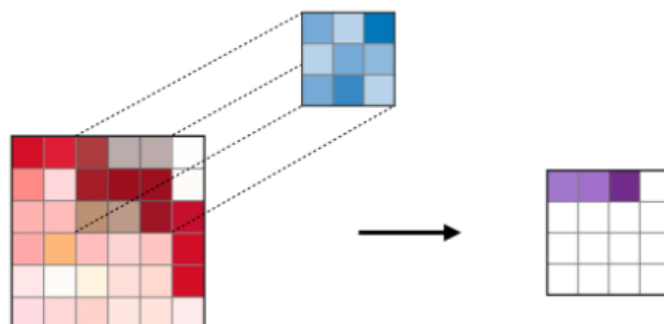


Đầu vào của VGG là ảnh RGB có kích thước 224x224. Ảnh này được đưa qua một tập các convolution layers sử dụng filter kích thước 3x3, bước trượt (stride) là 1 và có padding để đảm bảo kích thước ảnh được bảo toàn. Sau convolution layers là các max-pooling layers với bước trượt là 2 để giảm kích thước của feature map. Ở cuối mạng VGG là 3 lớp fully connected, sau cùng là một lớp softmax cho nhiệm vụ phân loại hình ảnh.

VGG có kiến trúc đơn giản, dễ hiểu, được sử dụng trong nhiều bài toán phân loại hình ảnh và đạt được độ chính xác cao. Tuy nhiên VGG có nhược điểm là kích thước lớn, nhiều tham số và tốn nhiều thời gian để huấn luyện.

Convolution layer

Phép convolution (tích chập) sử dụng một filter kích thước $k \times k$ (k thường là số lẻ như 3, 5, 7) trượt qua ảnh theo hướng từ trái sang phải, từ trên xuống dưới và thực hiện phép nhân tương ứng từng phần tử.



Đối với ảnh màu có 3 kênh màu thì filter sử dụng sẽ có kích thước $k \times k \times D$ với $D = 3$ là số kênh màu của ảnh.

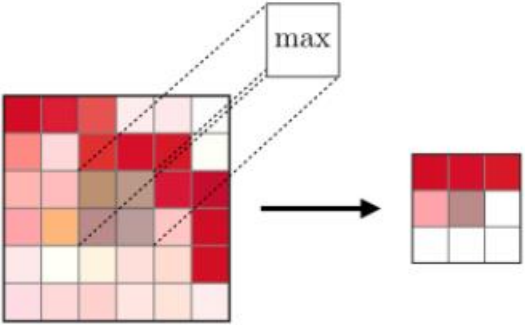
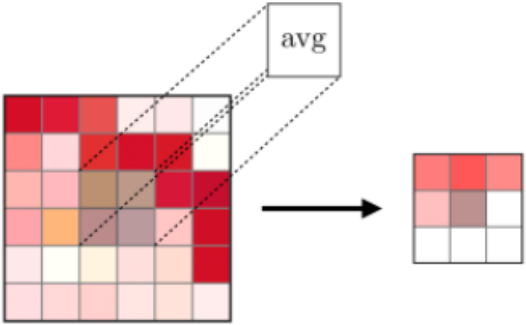
Padding

Do đặc điểm của phép convolution, nên ảnh sau khi thực hiện phép tính này sẽ bị giảm kích thước, các thông tin ở cạnh sẽ bị mất dần. Để khắc phục điều này, người ta thêm các pixel giá trị 0 vào cạnh của ảnh gọi là padding.

Ảnh 3x3 được thêm pad	Filter 3x3	=	Feature map 3x3																																											
<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>4</td><td>2</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>4</td><td>5</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0	0	0	4	2	0	0	1	1	0	0	0	3	4	5	0	0	0	0	0	0	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> </table>	0	1	0	0	1	0	0	1	0		<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>1</td><td>5</td><td>2</td></tr> <tr><td>4</td><td>9</td><td>7</td></tr> <tr><td>4</td><td>5</td><td>6</td></tr> </table>	1	5	2	4	9	7	4	5	6
0	0	0	0	0																																										
0	0	4	2	0																																										
0	1	1	0	0																																										
0	3	4	5	0																																										
0	0	0	0	0																																										
0	1	0																																												
0	1	0																																												
0	1	0																																												
1	5	2																																												
4	9	7																																												
4	5	6																																												

Pooling layer

Pooling là phép toán downsampling, thường theo sau lớp convolution có tác dụng giảm kích thước feature map. Có 2 loại pooling thường dùng là max-pooling và average-pooling.

Max pooling	Average pooling
	
Chọn ra giá trị lớn nhất trong vùng đang xét	Tính giá trị trung bình cho vùng đang xét

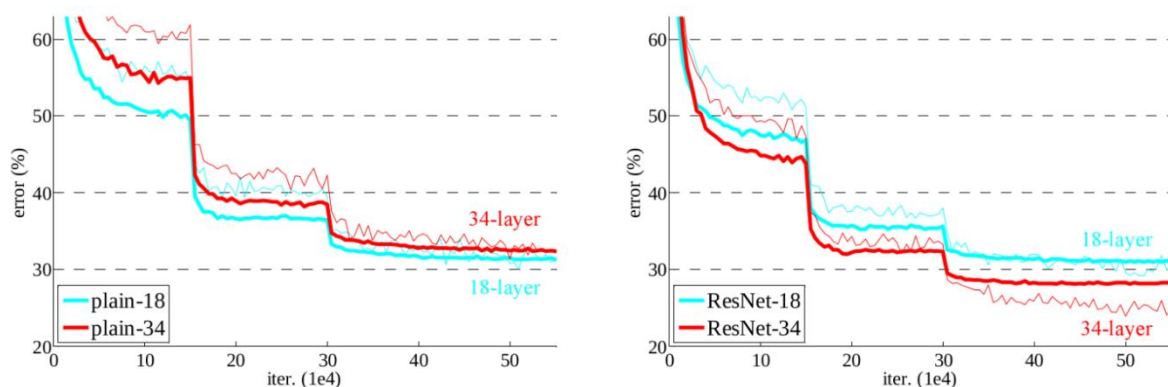
3.3. Residual Network (ResNet)

Resnet là một mạng có khả năng huấn luyện rất sâu với mạng chứa hàng trăm layers nhưng vẫn đạt được kết quả tốt.

Không phải cứ thêm nhiều layers vào mạng thì sẽ cho ra được kết quả tốt. Vấn đề đạo hàm bị triệt tiêu – vanishing gradients xảy ra khi cố gắng thêm nhiều layers vào mạng. Khi mạng học sâu bắt đầu hội tụ thì xảy ra vấn đề độ chính xác trở nên bị bão hòa, sau đó giảm rất nhanh và làm tăng độ lỗi huấn luyện.

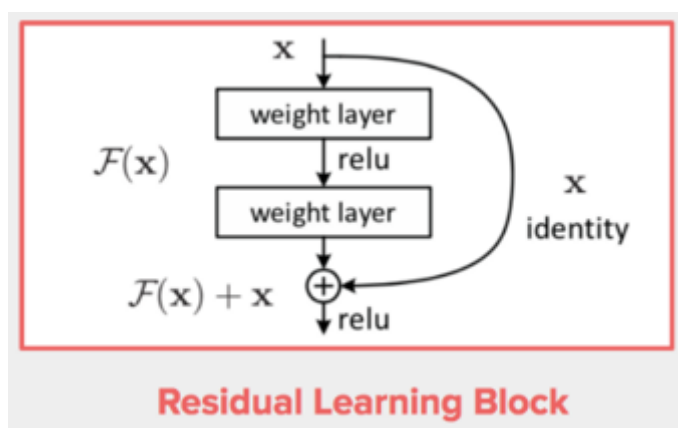
Khi so sánh với cấu trúc mạng VGG, để huấn luyện hiệu quả VGG chỉ có thể có nhiều nhất là 19 layers đối với VGG-19 hay 16 layers với VGG-16. Còn đối với Resnet, model có thể học được rất sâu với hàng trăm layers khi có cấu trúc đặc biệt là Residual Blocks.

Comparison



Biểu đồ bên trái cho thấy mạng thông thường thì 18 layers cho kết quả tốt hơn 34 layer với độ lỗi thấp hơn. Biểu đồ bên phải cho thấy ResNet-34 với 34 layers cho kết quả tốt hơn ResNet-18 với 18 layers.

Residual Learning Block



Block có thể dùng một identity mapping cho x đối với $\mathcal{F}(x)$ cho bước shortcut connection trước khi qua ReLU.

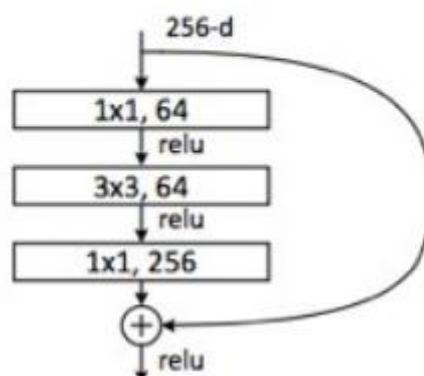
Một lợi ích cho bước shortcut identity mapping là không sử dụng thêm tham số.

Các mạng ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

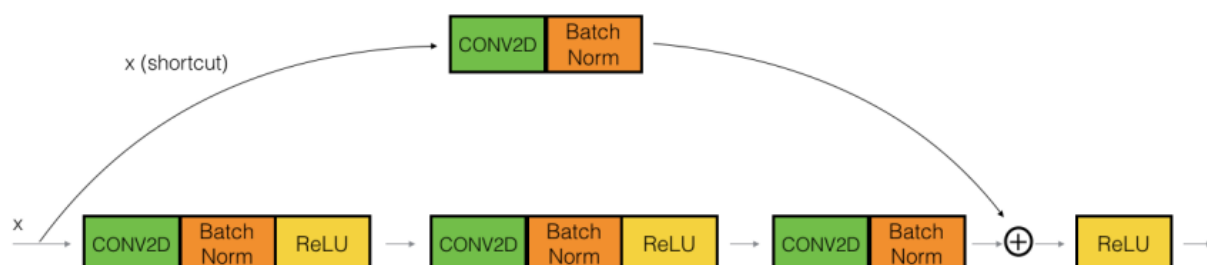
ResNet50

ResNet 50 residual block



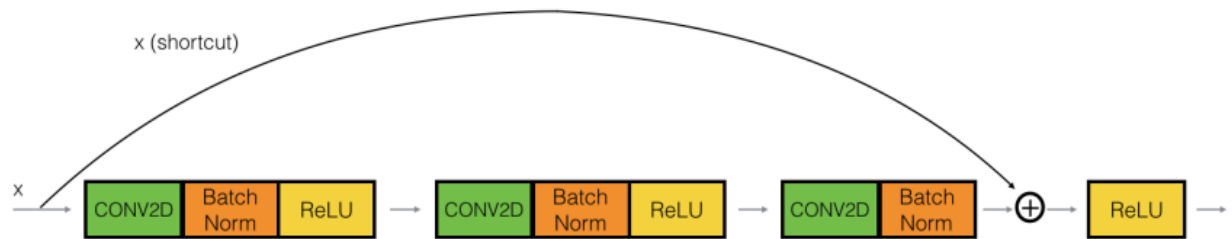
Cấu trúc mỗi Residual Block có 2 loại:

1. Block nét đứt: X được biến đổi đi qua Conv2D + BN



Trước khi $x + F(x)$ thì x đi qua Conv2D + BN để resize lại x bằng kích thước của $F(x)$ để thực hiện phép cộng rồi đi qua ReLU

2. Block nét liền: X giữ Nguyên



Không cần resize lại x vì đi qua các bước ở trên, $f(x)$ vẫn giữ nguyên kích thước bằng x .

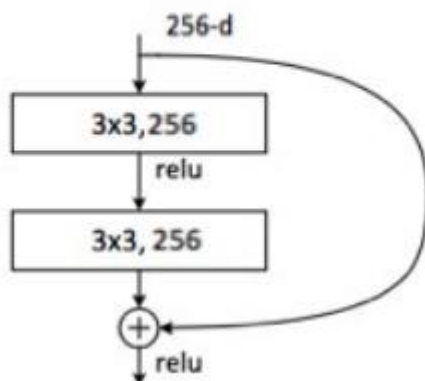
- Khi x đi qua bước identity block, Conv2D dùng kernel = (1, 1) để thay đổi kích thước của depth của x cho giống với phần depth $f(x)$ khi đi qua các bước biến đổi non-linear $\rightarrow x$ thay đổi ít so với input image.
- Số lượng tham số sẽ không nhiều hơn so với input image vì trong residual block các bước thực hiện không yêu cầu thêm tham số.
- Ý tưởng dùng kernel = [1, 1] cho phép giảm số lượng filter (64) trước khi tăng lên (256) ở bước CONV layer cuối cùng.

BatchNormalization

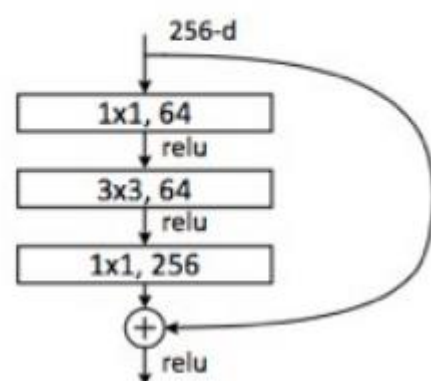
- Giúp chuẩn hóa đầu ra về trạng thái zero-mean với độ lệch chuẩn 1. Tránh hiện tượng Non-zero mean, là hiện tượng dữ liệu phân bố không đều có thể quá lớn hoặc quá bé. Từ đây dẫn đến đạo hàm có thể quá bé hoặc quá lớn làm cho model không ổn định.
- Khi dùng BatchNormalization có thể dùng learning rate cao để train giúp model chạy nhanh hơn và ổn định hơn.
- BatchNormalization tránh hiện tượng x rơi vào các khoảng bão hòa do đó giảm sự phụ thuộc vào giá trị khởi tạo.

Cải tiến từ ResNet34 sang Resnet50

ResNet 34
residual block



ResNet 50
residual block



Tốc độ training nhanh hơn ở Resnet50.

Giả sử có bức hình $100 \times 100 \times 1$. Dùng phép tính Convolution kernel=(3, 3), stride=1, padding=1. Tạo ra output kích thước 100×100 .

Mỗi unit của output thực hiện 9 phép nhân và 8 phép cộng \rightarrow 17 phép tính.

Vậy phép convolution cho toàn bức ảnh cần $100 \times 100 \times 17 = 170,000$ phép tính.

Resnet-34: $170,000 \times 256 \times 2 = 87,040,000$ phép tính.

Nếu dùng kernel 1×1 thì số phép tính sẽ giảm đi hẳn khi chỉ cần thực hiện $100 \times 100 = 10,000$.

Resnet-50: $10,000 \times 256 + 170,000 \times 64 + 10,000 \times 64 = 14,080,000$.

Số lượng phép tính giảm được cỡ 6 lần. Giúp cho việc tính toán nhanh hơn nhưng vẫn có thể học sâu vào bức ảnh với số lượng layers lớn.

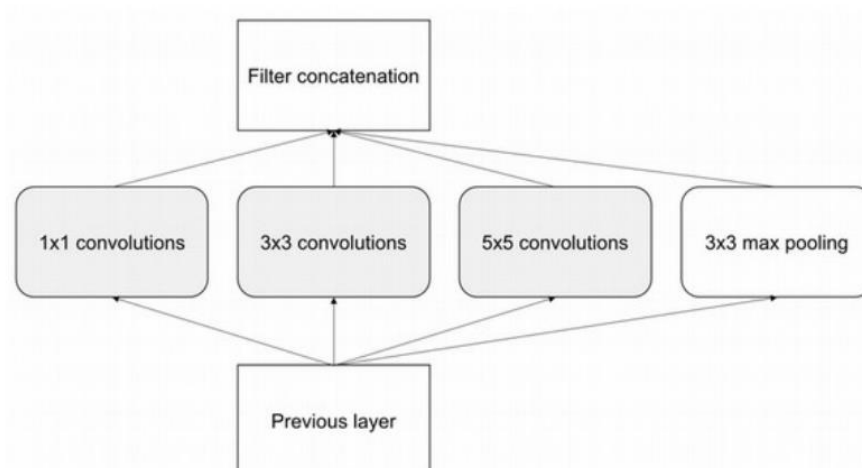
3.4. Inception Network (InceptionNet)

Khi so sánh với VGGNet, Inception Network chứng tỏ được sự hiệu quả trong việc tính toán cả về số lượng tham số trong mạng lẫn chi phí tiết kiệm về bộ nhớ.

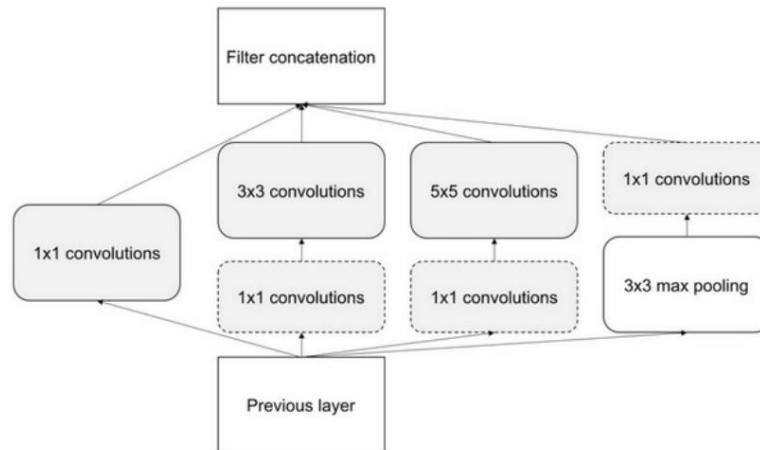
Cải thiện tốc độ training khi sử dụng nhiều convolutions 1×1 để thay thế cho 3×3 và 5×5

Mặc dù Inception cải thiện được số lượng tham số để tăng số lượng layers khi train để model học sâu hơn, nhưng Inception vẫn chưa hoàn toàn giải quyết được vấn đề: khi training càng nhiều layers thì độ khó khi training sẽ tăng lên khiến cho dễ gặp tình trạng đạo hàm bị triệt tiêu và dẫn đến hiện tượng model bị bão hòa.

Cấu trúc 1 Inception Block đơn giản



Cấu trúc cải tiến



Khi dùng thêm convolution 1x1 vào trước 3x3 và 5x5 convolution sẽ làm giảm 1 lượng lớn tham số cần dùng.

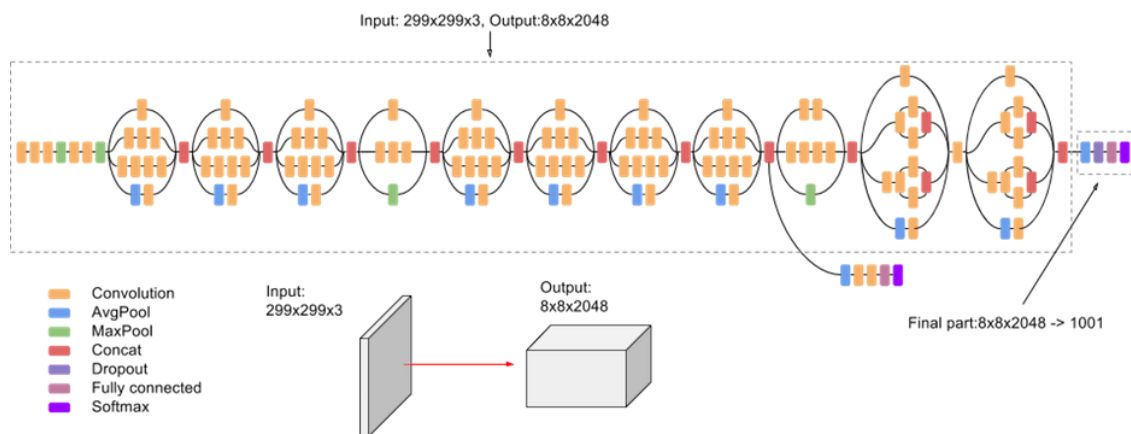
Giả sử có input $28 \times 28 \times 192$:

- Đi qua convolution 5×5 với 32 filter thì số tham số cần dùng là $5 \times 5 \times 192 \times 32 = 153600$ tham số.

- Nếu dùng convolution 1x1 với 16 filter trước 5x5 convolution với 32 filter thì số tham số là $1 \times 1 \times 192 \times 16 + 5 \times 5 \times 16 \times 32 = 15872$.

→ Giảm được 10 lần số lượng tham số.

Inception V3



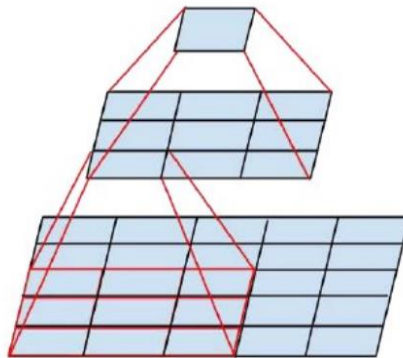
Average Pooling

Dùng Average Pooling thay cho FC layer sau convolutional layer cuối cùng cũng làm giảm số lượng tham số. Giả sử với input $7 \times 7 \times 1024$ thì có thể chuyển đổi thành $1 \times 1 \times 1024$ mà không cần dùng tham số nào. Nếu dùng FC layer thì cần dùng $(7 \times 7 \times 1024) \times 1024 = 51380224$ tham số.

Factorizing Convolutions

- Smaller Convolutions

Thay thế 1 kernel 5x5 bằng 2 kernel 3x3.

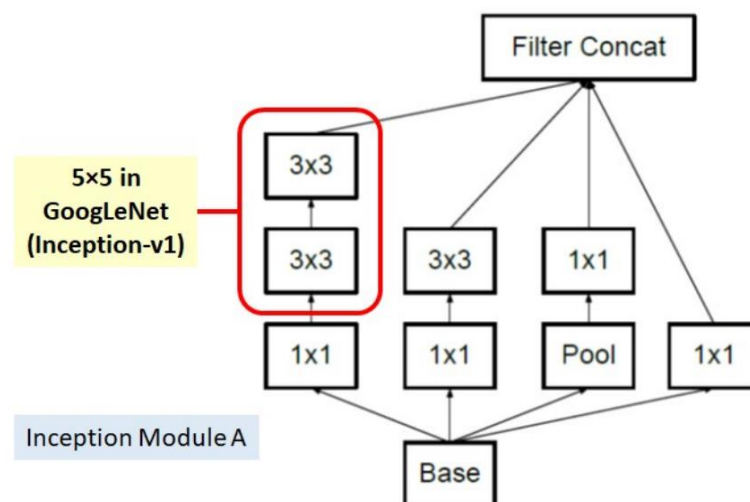


Kernel 5x5: Số lượng tham số: $5 \times 5 = 25$.

2 kernel 3x3: Số lượng tham số: $2 \times 3 \times 3 = 18$.

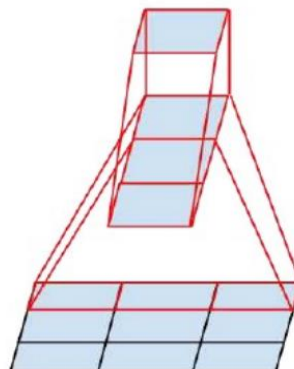
→ Giảm 28% lượng tham số.

Thu được **Inception Module A**.



- **Asymmetric Convolutions**

Thay thế kernel 3x3 bằng kernel 3x1 và 1x3.

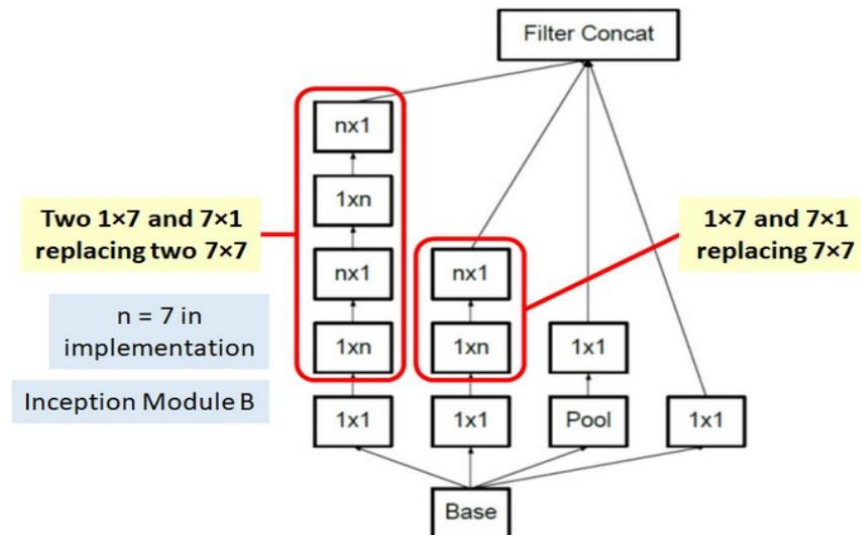


Kernel 3x3: Số lượng tham số $3 \times 3 = 9$.

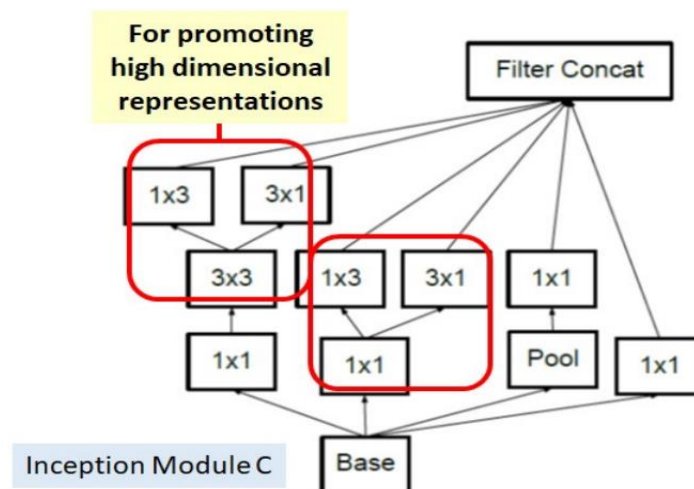
Kernel 3x1 và 1x3: Số lượng tham số $3*1 + 1*3 = 6$.

→ Giảm 33% số lượng tham số

Thu được **Inception Module B**



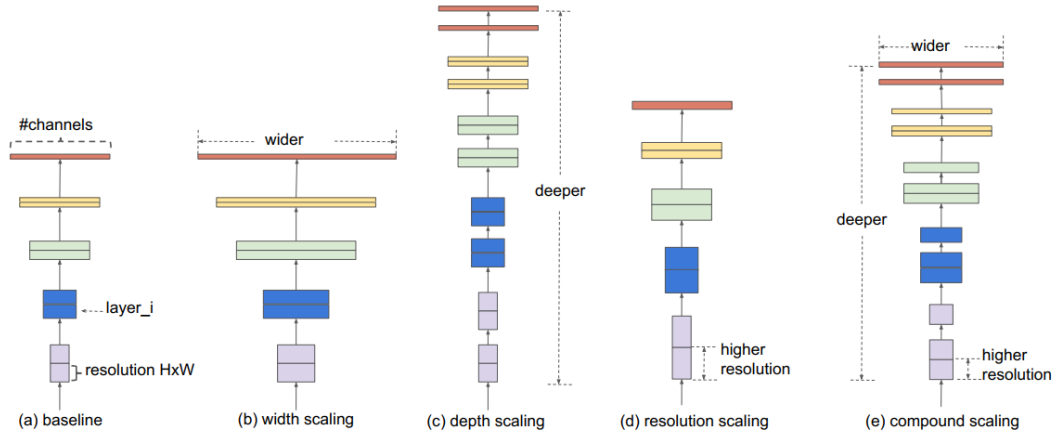
Inception Module C



3.5. EfficientNet

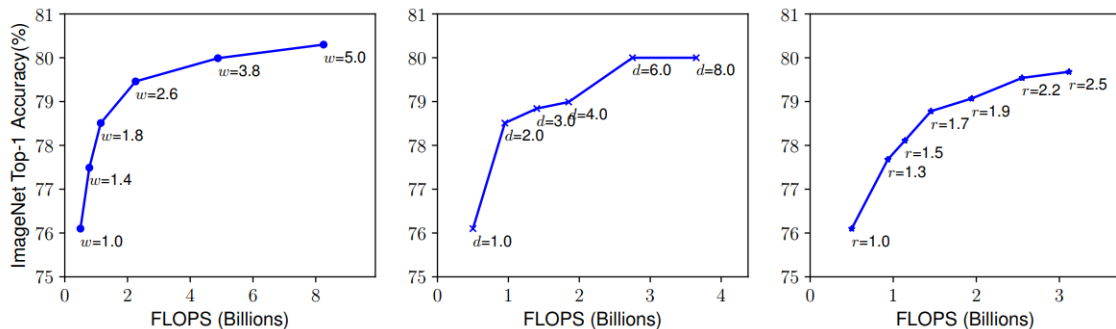
Mạng Convolutional Neural Network (ConvNet) thường được xây dựng trên một kích thước, chi phí cố định, sau đó được mở rộng (scale up) ra tùy theo tài nguyên sẵn có để cải thiện độ chính xác của mô hình. Tuy nhiên việc mở rộng mạng ConvNet chưa được nghiên cứu kỹ và chưa có nguyên lý rõ ràng. Cách thông thường là tăng một cách ngẫu nhiên độ sâu (depth), rộng (width) của mạng ConvNet hoặc sử dụng ảnh có độ phân giải cao (high-res image). Quá trình này đòi hỏi việc tinh chỉnh khá thủ công và không hiệu quả.

EfficientNet đưa ra phương pháp mở rộng mạng ConvNet một cách hệ thống. Việc mở rộng có nguyên lý đảm bảo cân bằng giữa độ sâu (width), rộng (width) và độ phân giải (resolution) để cải thiện hiệu suất của mô hình baseline.



Ba yếu tố cần xem xét quá trình mở rộng mạng ConvNet gồm:

- **Depth (d):** tăng/giảm chiều sâu bằng cách thêm/bớt layers là cách làm phổ biến nhất ở nhiều kiến trúc mạng. Một cách trực giác, mạng ConvNet càng sâu thì càng có khả năng học được các đặc trưng phức tạp hơn, giàu ngữ nghĩa hơn (more complex and richer features) và tổng quát được cho nhiều tác vụ khác. Tuy nhiên mạng ConvNet sâu sẽ khó huấn luyện hơn do vấn đề gradient vanishing.
- **Width (w):** thay đổi độ rộng là cách thường dùng cho các mô hình có kích thước nhỏ. Mô hình rộng có xu hướng học được nhiều đặc trưng chi tiết (fine-grained features) và dễ huấn luyện hơn. Tuy nhiên mạng càng rộng thì càng khó học các đặc trưng trừu tượng (higher level features)
- **Resolution (r):** với ảnh có độ phân giải cao, thì mạng ConvNet có tiềm năng học được nhiều chi tiết hơn (fine-grained patterns). Độ phân giải tăng thì độ chính xác (accuracy) tăng, nhưng sẽ giảm đối với ảnh độ phân giải quá cao



Biểu đồ trên mang ý nghĩa: việc mở rộng bất kỳ kích thước width, depth, resolution của mạng đều làm tăng độ chính xác, tuy nhiên độ chính xác bắt đầu giảm khi model quá lớn.

Tác giả của EfficientNet chỉ ra rằng các kích thước (dimensions) không độc lập mà có quan hệ với nhau. Do đó, để đạt được độ chính xác cao hơn, điểm mấu chốt là cân bằng việc điều chỉnh giữa các kích thước depth, width và resolution thay vì chỉ điều chỉnh một kích thước như cách thông thường (conventional single-dimension scaling).

EfficientNet đề xuất phương pháp gọi là **compounding scaling method**, sử dụng hệ số kết hợp (compound coefficient) ϕ để tinh chỉnh các kích thước depth, width, resolution của mạng một cách đồng nhất và có hệ thống:

$$\begin{aligned}
&\text{depth: } d = \alpha^\phi \\
&\text{width: } w = \beta^\phi \\
&\text{resolution: } r = \gamma^\phi \\
&\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
&\alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}$$

trong đó ϕ là hệ số do người cài đặt xác định dựa trên kích thước nguồn tài nguyên sẵn có, α, β, γ xác định cách phân bổ tài nguyên vào các kích thước width, depth và resolution

Kiến trúc mạng EfficientNet-B0

Stage i	Operator \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

Bảng trên mô tả kiến trúc của mô hình EfficientNet-B0 baseline. Kiến trúc baseline này sử dụng Mobile inverted bottleneck convolution (MBConv) tương tự như MobileNetV2 và MnasNet nhưng lớn hơn một chút. Sau đó, kiến trúc baseline này được mở rộng (scale up) thành một họ các mô hình gọi là EfficientNets. Áp dụng phương pháp compound scaling method để mở rộng mô hình baseline qua 2 bước:

- Bước 1: Gán $\phi = 1$, tiến hành small grid search để tìm α, β, γ . Đối với EfficientNet-B0, giá trị tốt nhất là $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, thỏa mãn ràng buộc $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- Bước 2: Giữ giá trị α, β, γ tìm được ở bước trên, mở rộng mô hình baseline với nhiều giá trị ϕ khác nhau, kết quả thu được nhiều mô hình EfficientNet-B1 đến B7.

Tóm lại, EfficientNet với compound scaling method giúp dễ dàng mở rộng mạng ConvNet baseline lên bất kỳ kích cỡ nào mong muốn nhưng vẫn giữ được tính hiệu quả của mô hình

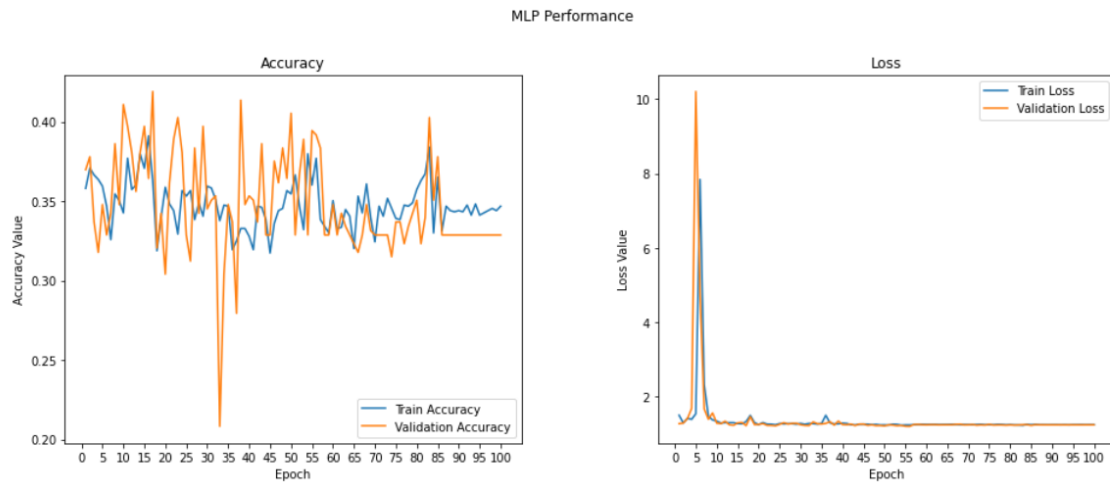
4. Báo cáo kết quả và nhận xét

4.1 Báo cáo kết quả

Khi training model, nhóm tiến hành lưu lại file weight của lần train có val_loss thấp nhất. Với mỗi mô hình, nhóm khảo sát biểu đồ accuracy, loss trên tập train và validation. Ngoài ra nhóm cũng tính confusion matrix của mô hình trên tập validation.

- MLP

Độ chính xác trên tập train	Độ chính xác trên tập validation	Thời gian training
38.39%	39.18%	53 phút



Đồ thị loss và accuracy của MLP

Confusion Matrix

```
[[ 43  0  57  0]
 [  0  0  18  0]
 [ 20  0 100  0]
 [ 19  0 108  0]]
```

Classification Report

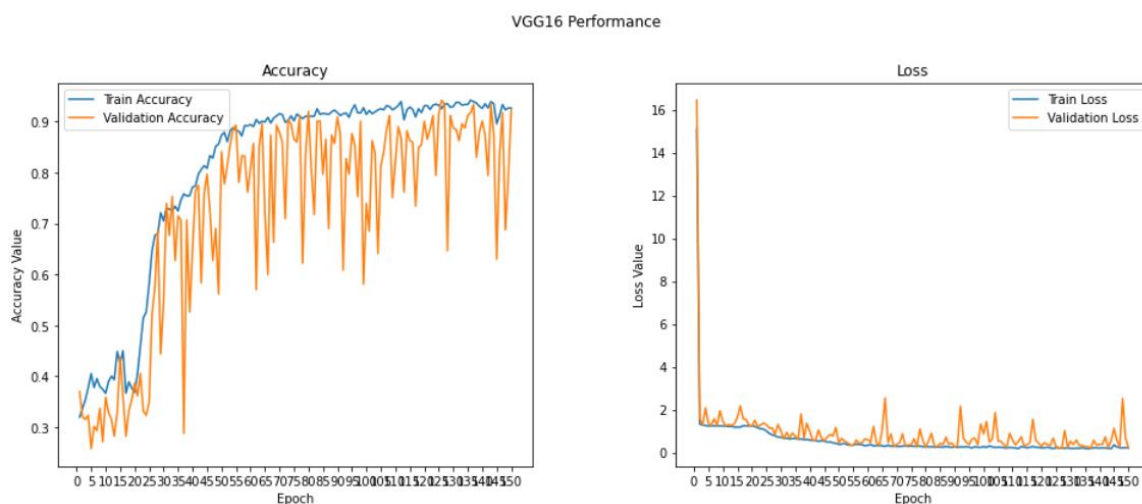
	precision	recall	f1-score	support
healthy	0.52	0.43	0.47	100
multiple_diseases	0.00	0.00	0.00	18
rust	0.35	0.83	0.50	120
scab	0.00	0.00	0.00	127
accuracy			0.39	365
macro avg	0.22	0.32	0.24	365
weighted avg	0.26	0.39	0.29	365

Confusion matrix, Precision, Recall, F1-score của MLP

Nhận xét: MLP dự đoán sai khá nhiều, vì mô hình không đủ mạnh để học được những đặc trưng phức tạp đủ để phân loại các lá. MLP không đưa ra dự đoán nào thuộc lớp multiple_diseases với scab. Hơn nữa các dự đoán thuộc 2 lớp healthy với rust của MLP cũng sai khá nhiều.

• VGG16

Độ chính xác trên tập train	Độ chính xác trên tập validation	Thời gian training
94.44%	93.42%	3 tiếng 10 phút



Đồ thị loss và accuracy của VGG-16

Confusion Matrix

```
[[ 99   0   1   0]
 [  1   6   9   2]
 [  1   0 119   0]
 [  8   1   1 117]]
```

Classification Report

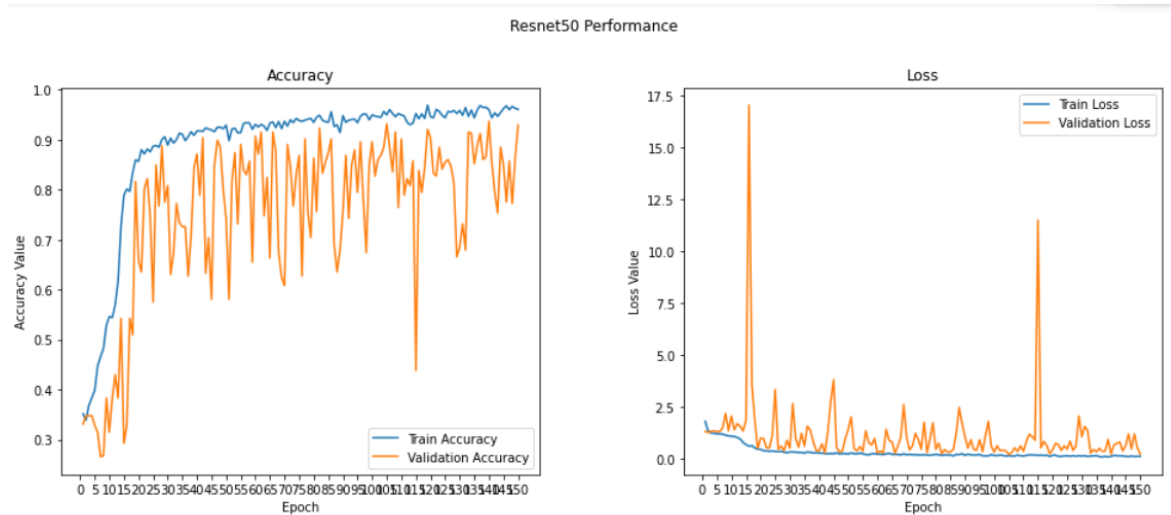
	precision	recall	f1-score	support
healthy	0.91	0.99	0.95	100
multiple_diseases	0.86	0.33	0.48	18
rust	0.92	0.99	0.95	120
scab	0.98	0.92	0.95	127
accuracy			0.93	365
macro avg	0.92	0.81	0.83	365
weighted avg	0.93	0.93	0.93	365

Confusion matrix, Precision, Recall, F1-score của VGG-16

Nhận xét: VGG-16 cho kết quả phân loại khá tốt. Mô hình hay sai ở những dự đoán healthy và rust nguyên nhân có lẽ là mô hình chưa đủ mạnh để nhận ra trường hợp khó của scab. Một số lá bị mảng xám (scab) nhưng bị phân loại nhầm thành healthy có lẽ vì mảng xám khá mờ. Một số lá bị multiple_diseases nhưng mô hình chỉ nhận ra triệu chứng đốm đỏ nên phân loại nhầm thành rust.

- **ResNet50**

Độ chính xác trên tập train	Độ chính xác trên tập validation	Thời gian training
93.41%	93.15%	1 tiếng 30 phút



Đồ thị loss và accuracy của ResNet50

Confusion Matrix

```
[[ 95   3   0   2]
 [  1  16   0   1]
 [  1  11 108   0]
 [  5   1   0 121]]
```

Classification Report

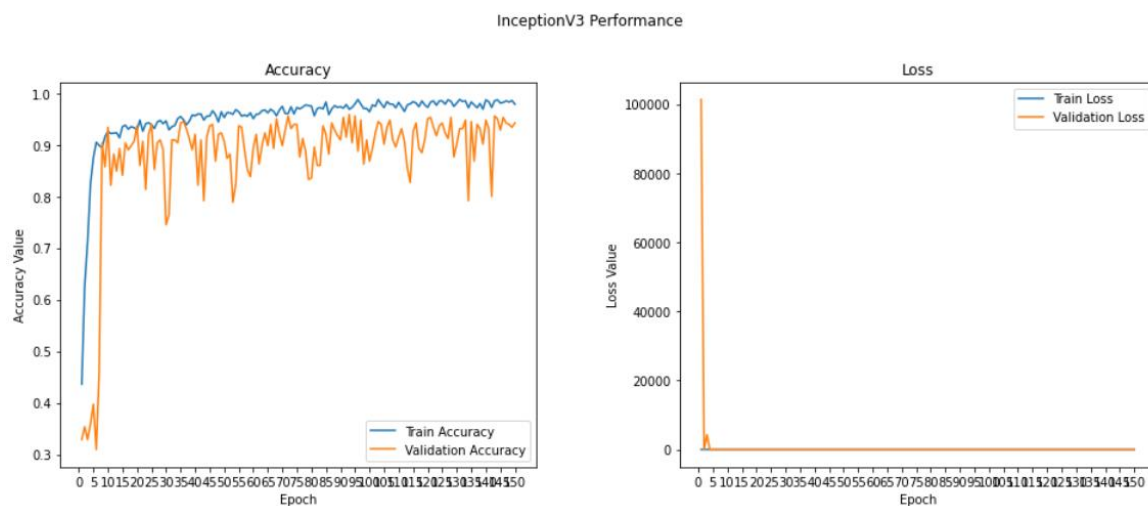
	precision	recall	f1-score	support
healthy	0.93	0.95	0.94	100
multiple_diseases	0.52	0.89	0.65	18
rust	1.00	0.90	0.95	120
scab	0.98	0.95	0.96	127
accuracy			0.93	365
macro avg	0.86	0.92	0.88	365
weighted avg	0.95	0.93	0.94	365

Confusion matrix, Precision, Recall, F1-score của ResNet50

Nhận xét: ResNet50 cho kết quả dự đoán khá tốt, lỗi sai tập trung chủ yếu ở dự đoán healthy và multiple_diseases. Có những lá chỉ bị rust nhưng mô hình lại nhận nhầm thêm triệu chứng scab nên đã phân loại nhầm vào multiple_diseases. Có những lá bị scab nhưng mảng xám khá mờ nên bị phân loại nhầm thành healthy.

- InceptionV3**

Độ chính xác trên tập train	Độ chính xác trên tập validation	Thời gian training
97.46%	94.52%	1 tiếng 30 phút



Đồ thị loss và accuracy của InceptionV3

Confusion Matrix

```
[[ 98   0   1   1]
 [  1  11   5   1]
 [  0   1 119   0]
 [  5   5   0 117]]
```

Classification Report

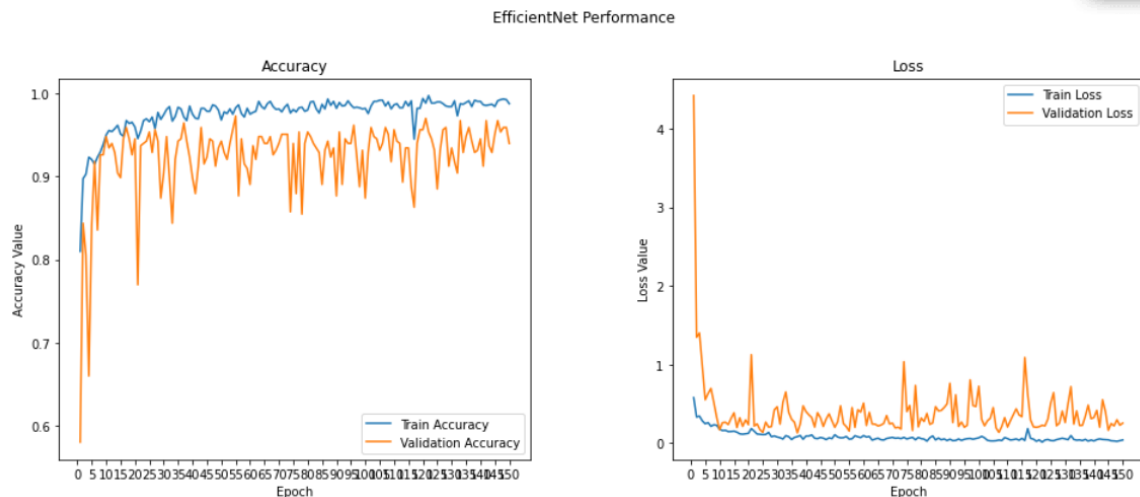
	precision	recall	f1-score	support
healthy	0.94	0.98	0.96	100
multiple_diseases	0.65	0.61	0.63	18
rust	0.95	0.99	0.97	120
scab	0.98	0.92	0.95	127
accuracy			0.95	365
macro avg	0.88	0.88	0.88	365
weighted avg	0.95	0.95	0.94	365

Confusion matrix, Precision, Recall, F1-score của InceptionV3

Nhận xét: InceptionV3 cho kết quả phân loại khá tốt. Một số lá bị multiple_diseases nhưng mô hình chỉ nhận ra được triệu chứng rust và phân loại nhầm vào rust. Một số lá bị scab nhưng mảng xám bị mờ nên mô hình phân loại nhầm thành healthy.

- **EfficientNet**

Độ chính xác trên tập train	Độ chính xác trên tập validation	Thời gian training
98.08%	96.44%	2 tiếng 40 phút



Đồ thị loss và accuracy của EfficientNet

Confusion Matrix

```
[[100   0   0   0]
 [  1  14   2   1]
 [  0   1 119   0]
 [  8   0   0 119]]
```

Classification Report

	precision	recall	f1-score	support
healthy	0.92	1.00	0.96	100
multiple_diseases	0.93	0.78	0.85	18
rust	0.98	0.99	0.99	120
scab	0.99	0.94	0.96	127
accuracy			0.96	365
macro avg	0.96	0.93	0.94	365
weighted avg	0.97	0.96	0.96	365

Confusion matrix, Precision, Recall, F1-score của EfficientNet

Nhận xét: EfficientNet phân loại các lá rất tốt cả với trường hợp lá bị nhiều bệnh (multiple_diseases) , chỉ có 1 dự đoán multiple_diseases bị sai. Ở cả 4 trường hợp thì số lá bị dự đoán sai đều rất ít.

4.2 Nhận xét

Bảng tổng kết kết quả các mô hình

Mô hình	Độ chính xác trên tập training	Độ chính xác trên tập validation	Thời gian training (150 epoch)
MLP	38.39%	39.18%	53 phút (100 epoch)
VGG-16	94.44%	93.42%	3 tiếng 10 phút
ResNet50	93.41%	93.15%	1 tiếng 30 phút
InceptionV3	97.46%	94.52%	1 tiếng 30 phút
EfficientNet	98.08%	96.44%	2 tiếng 40 phút

Đối với mạng MLP thông thường với 3 lớp ẩn, model cũng không thể học hết được các đặc trưng của ảnh để cho được kết quả tốt. Do đó, kết quả của mô hình MLP khá thấp. Nhóm đã thử tăng số lượng lớp ẩn lên và với nhiều node hơn nhưng kết quả cũng không khả quan và còn thấp hơn kết quả hiện tại.

Thời gian training của VGG16 và EfficientNet khá lâu so với Resnet50 và InceptionV3. Điều đó cũng đã được giải thích, cấu trúc của ResNet50 và InceptionV3 có nhiều Convolutional Layer với kernel 1x1 làm giảm được số lượng tính toán cũng như giảm lượng tham số.

Nhìn chung các mô hình dễ bị sai trong việc phân biệt trường hợp lá healthy với lá bị scab nhưng triệu chứng mảng xám đen khá mờ, khó nhận ra. Một trường hợp nữa là lá bị 2 bệnh (multiple_diseases) nhưng mô hình thường chỉ nhận ra triệu chứng đốm đỏ của rust, mà không nhận ra triệu chứng mảng xám của scab dẫn đến phân loại sai.

Kết quả so sánh cho thấy mạng EfficientNet cho kết quả tốt nhất trong các model CNN.

III- TÀI LIỆU THAM KHẢO

- [1]. Vũ Hữu Tiệp, Machine Learning Cơ Bản
- [2]. [CS 229 – Machine Learning Cheetsheet – Stanford](#)
- [3]. [CS 230 – Deep Learning Cheetsheet – Stanford](#)
- [4]. [CNN VGG-16 and VGG-19 – DataHacker](#)
- [5]. Rethinking the Inception Architecture for Computer Vision
- [6]. HANDSON_COMPUTER_VISION_WITH_TENSORFLOW_2
- [7]. [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)
- [8]. [EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling](#)
- [9]. Mingxing Tan, Quoc V. Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2020
- [10]. [Confusion Matrix for Your Multi-Class Machine Learning Model](#)