

TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG

NGÔ VĂN TUÂN – 1613861

ĐỒ ÁN MÔN HỌC
ỨNG DỤNG NEURAL NETWORK
TRONG NHẬN DẠNG BỆNH CỦA CÂY

GIẢNG VIÊN HƯỚNG DẪN
PGS.TS. HÀ HOÀNG KHA

TP. HỒ CHÍ MINH, 2019

Mục lục

Chương 1: Giới thiệu	1
1.1. Giới thiệu	1
1.2. Mục tiêu đề tài.....	2
Chương 2: Mạng thần kinh nhân tạo	3
2.1 Giới thiệu về cấu trúc cơ bản của một mạng thần kinh nhân tạo	3
2.2 Một số network topology phổ biến.....	4
2.3 Sự khác biệt giữa artificial neural network và biological neural network	5
2.4. Giới thiệu về cấu trúc của một neuron	5
2.5. Fully connected layer.....	7
2.6. Loss function	8
2.7. Back propagation.....	9
2.8. Convolutional neural layer	12
2.9. Pooling layer.....	15
Chương 3: Xây dựng cấu trúc mạng neuron	16
3.1. Tích chập với kernel one-by-one [1x1]	16
3.1.1. Biến đổi features.....	16
3.1.2. Giúp mạng thần kinh sâu hơn.....	16
3.1.3 Sử dụng 1x1 kernel trong Inception module	16
3.2. Inception topology	17
3.2.1. Factorizing convolutions	18
3.2.2. Auxiliary classifier	20
3.2.3. Efficient grid size reduction	21
3.2.4. Cấu trúc mạng Inception-V3	22

3.2.5. Label smoothing as regularization.....	22
Chương 4: Xây dựng mô hình và kết quả thực nghiệm.....	23
4.1. Chuẩn bị dataset.....	23
4.2. Xây dựng mô hình mạng dựa trên TensorFlow framework	24
4.3. Đồ thị hàm loss trong quá trình train.....	31
4.4. Đánh giá trên tập validation	31
4.5. Xây dựng giao diện phần mềm.	32
Chương 5: Đánh giá, kết luận và hướng phát triển	35
6.1. Đánh giá	35
6.2. Kết luận	35
6.3. Hướng phát triển	35
Tài liệu tham khảo	36

Danh sách hình ảnh

Hình 1. Neural network	3
Hình 2. Network topology	4
Hình 3. Real neuron.....	5
Hình 4. Virtual neuron.....	5
Hình 5. Activation function	6
Hình 6. Feed forward.....	7
Hình 7. Fully connected neural network	7
Hình 8. Back propagation.....	11
Hình 9. Convolutional neural network	12
Hình 10. Input image	13
Hình 11. Thuật toán pooling.....	15
Hình 12. Cấu trúc Inception module	17
Hình 13. Ý tưởng của inception	17
Hình 14. Thay thế filter 5x5 bởi 2 filter 3x3	18
Hình 15. Inception module A	18
Hình 16. Thay thế 3x3 filter bằng 3x1 và 1x3 filters	19
Hình 17. Inception module B	19
Hình 18. Inception module C	20
Hình 19. Auxiliary classifier	20
Hình 20. Efficient grid size reduction	21
Hình 21. Cấu trúc mạng Inception-V3	22
Hình 22. Cấu trúc của mạng	24
Hình 23. Các lớp tích chập đầu tiên	25
Hình 24. Inception module A	26
Hình 25. Grid size reduction	27
Hình 26. Inception module B	28
Hình 27. Grid size reduction	29
Hình 28. Inception module C	30
Hình 29. Fully connect	30
Hình 30. Batch accuracy.....	31
Hình 31. Loss function	31
Hình 32. Graphic user interface.....	32
Hình 33. Dự đoán đúng bệnh Apple__Back_rot.....	32
Hình 34. Dự đoán đúng bệnh Blueberry__healthy.....	33
Hình 35. Dự đoán đúng bệnh Tomato__late_blight.....	33
Hình 36. Dự đoán sai bệnh Grape__healthy.	34

Chương 1: Giới thiệu

1.1. Giới thiệu

Trí tuệ nhân tạo là một lĩnh vực đã và đang được nghiên cứu, phát triển mạnh mẽ trên toàn thế giới. Máy học (machine learning) nổi lên như một trong những phương pháp tổng quát để giải các bài toán trí tuệ nhân tạo.

Dựa vào phương thức học, machine learning có thể được chia làm 3 loại:

- Học có giám sát (Supervised Learning): Trong mô hình này, cả đầu vào và đầu ra của thuật toán đều được cung cấp sẵn. Nhiệm vụ của thuật toán là tìm ra mối liên hệ giữa đầu vào và đầu ra để áp dụng mối liên hệ đó cho những dữ liệu chưa thấy bao giờ.
- Học không giám sát (Unsupervised Learning): Trong mô hình này, người phát triển không biết hoặc không thể tìm được đầu ra. Do đó, nhiệm vụ của thuật toán là tìm mối liên hệ giữa các đầu vào.
- Học củng cố (Reinforcement Learning): Trong mô hình này, người phát triển cần được cung cấp một mô hình để đánh giá kết quả của thuật toán. Nhiệm vụ của thuật toán là làm sao cho ra kết quả được đánh giá cao nhất có thể.

Ngoài ra, còn có rất nhiều các thuật toán machine learning, có thể kể đến như: Linear Regression, Logistic Regression, Decision Tree, SVM, Naive Bayes, kNN, K-Means, Random Forest, ...

Cùng với sự phát triển không ngừng của các thiết bị phần cứng với tốc độ tính toán ngày càng cao và khả năng tính toán song song. Các thuật toán machine learning có tính tổng quát cao ngày càng được ưu chuộng. Trong đó, neural network với khả năng xấp xỉ tốt các hệ phi tuyến được sử dụng phổ biến nhất.

Tuy nhiên, việc ứng dụng neural network trong computer vision thật sự chỉ đạt được bước tiến lớn khi người ta bắt đầu ứng dụng convolution neural network (CNN).

Các lớp Convolution neural có tác dụng phân tích ra các features từ dữ liệu đầu vào. Các feature được tách ra sẽ được đưa qua các fully connected layer để thực hiện mục tiêu thuật toán. Việc phân tích ra các features quan trọng trước khi đi tìm quy luật rõ ràng đem lại sự thuận lợi hơn rất nhiều cho việc tính toán bởi vì điều này làm giảm không gian tìm kiếm đi rất nhiều.

1.2. Mục tiêu đề tài

Mục tiêu đề tài: Sử dụng mạng Convolutional Neural Network (CNN) để nhận dạng bệnh của cây từ ảnh chụp của lá cây.

Công việc cần làm để thực hiện mục tiêu:

- Tìm hiểu về artificial neural network.
- Tìm hiểu về TensorFlow Machine Learning Platform.
- Thu thập dữ liệu (dataset).
- Phân tích, xây dựng cấu hình mạng neuron InceptionV3.
- Implement mô hình mạng sử dụng TensorFlow framework.
- Huấn luyện mạng và kiểm thử kết quả.
- Xây dựng giao diện người dùng.

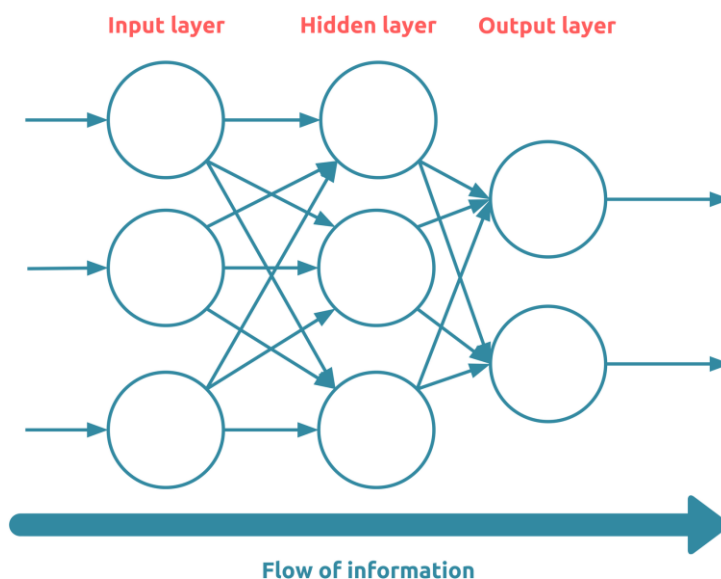
Chương 2: Mạng thần kinh nhân tạo

2.1 Giới thiệu về cấu trúc cơ bản của một mạng thần kinh nhân tạo

Định nghĩa: Mạng thần kinh nhân tạo (artificial neural network) là một hệ thống tính toán dựa trên mô phỏng quá trình hoạt động của bộ não động vật. Hệ thống này sẽ học cách thực hiện nhiệm vụ dựa vào tập mẫu, không dựa theo một quy tắc được lập trình trước.

Mạng thần kinh nhân tạo là một tập hợp các neurons được tổ chức theo lớp:

- Input layer: mạng thông tin, dữ liệu khởi tạo vào hệ thống
- Hidden layer: lớp nằm ở giữa input layer và output layer, nơi mà các neurons nhân tạo đưa vào các weights và đưa ra các output thông qua hàm kích hoạt (activation function).
- Output layer: Lớp cuối cùng của mạng thần kinh, đưa ra output cho chương trình.



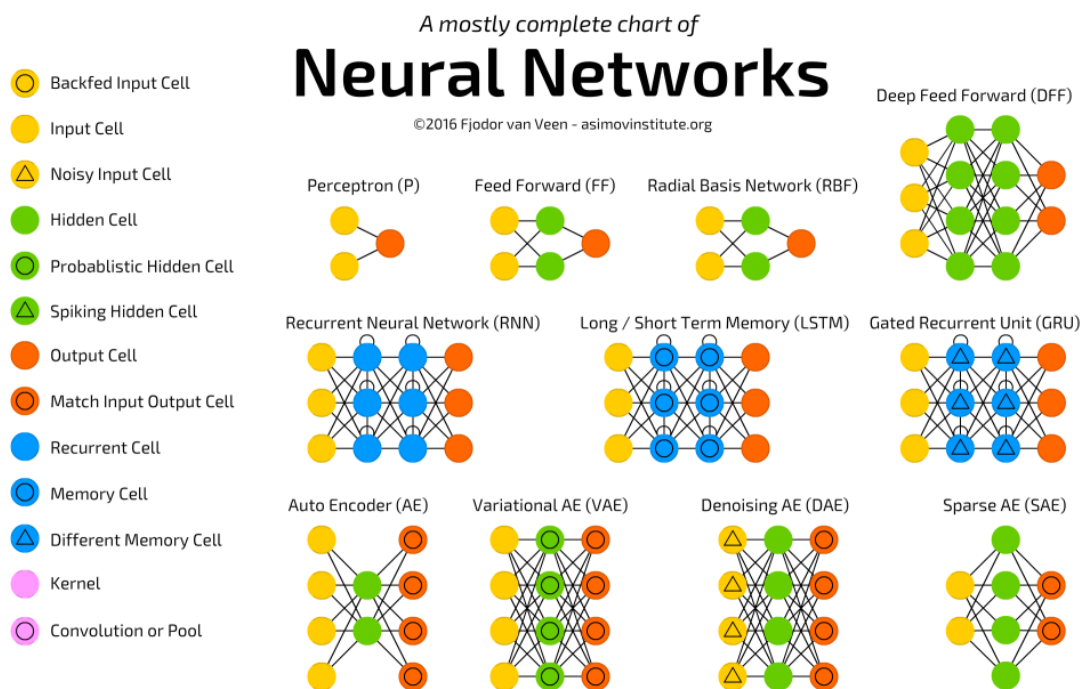
Hình 1. Neural network

Mạng thần kinh nhân tạo với nhiều hơn 2 lớp có thể được gọi là mạng thần kinh sâu (deep artificial neural network). Lợi ích của việc sử dụng mạng học sâu là một mẫu phức tạp hơn có thể được nhận dạng.

2.2 Một số network topology phổ biến

Một số network topology của mạng thần kinh nhân tạo bao gồm:

- Perceptron
- Feed forward
- Radial basis network
- Deep feed forward
- Recurrent neural network
- Long/Short term memory network
- Gated recurrent unit
- Auto encoder
- Variational Auto Encoder
- Denoising Auto encoder
- Sparse auto encoder



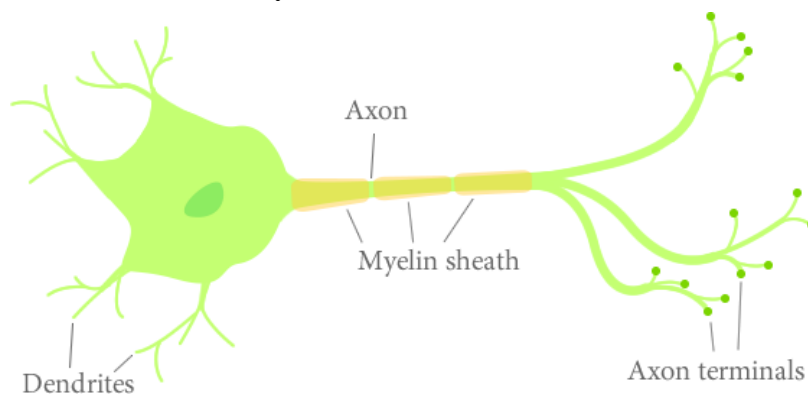
Hình 2. Network topology

2.3 Sự khác biệt giữa artificial neural network và biological neural network

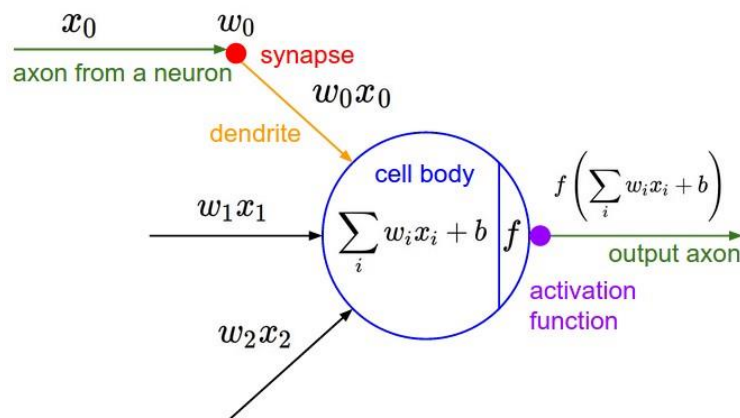
Một số sự khác biệt giữa bộ não và mạng thần kinh nhân tạo:

- Neuron nhân tạo kích hoạt hoàn toàn khác với neuron trong bộ não.
- Một người bình thường có 100 tỉ neurons và 100 nghìn tỷ kết nối, hoạt động với công suất tiêu thụ 20W (đủ để thắp sáng một bóng đèn). Trong khi đó, mạng thần kinh nhân tạo lớn nhất có 10 triệu neurons và 1 tỉ kết nối, chạy với 16000 CPU tương đương công suất tiêu thụ là 3 triệu watt.
- Một bộ não giới hạn bởi 5 loại dữ liệu đầu vào của 5 giác quan.
- Một đứa trẻ không học nhận dạng một vật bằng cách xem 100 000 bức ảnh của sự vật đó được dán nhãn phải – không phải, nhưng đó lại là cách máy tính hoạt động.
- Chúng ta không học cách tính tích phân từng phần của mỗi neural.

2.4. Giới thiệu về cấu trúc của một neuron



Hình 3. Real neuron



Hình 4. Virtual neuron

Một tế bào neuron được đặc trưng bởi các thông số bao gồm:

- Vector hệ số weights $\bar{w} = \{w_0, w_1, w_2, \dots\}$.
- Bias b .
- Hàm kích hoạt (Activation function). Hàm kích hoạt là một hàm phi tuyến với mục tạo ra sự phi tuyến cho mạng, giúp mạng có thể xấp xỉ các hàm số phi tuyến.

Đầu vào của một neuron là một vector $\bar{x} = \{x_0, x_1, x_2, \dots\}$

Đầu ra của một neuron là một số (scalar) thường được ký hiệu là y .

Một neuron sẽ thực hiện việc nhân vô hướng giữa w và x , thêm vào bias, kết quả là một số và số này được đưa qua hàm kích hoạt.

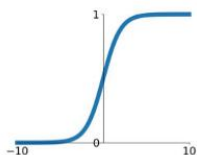
$$y = f(\bar{w} \cdot \bar{x} + b) = f\left(\sum_i w_i x_i + b\right)$$

Hàm kích hoạt:

Sau khi thực hiện phép nhân vô hướng, kết quả được đưa qua một hàm phi tuyến, hàm này được gọi là hàm kích hoạt (Activation function). Trong quá khứ, các hàm kích hoạt phổ biến là sigmoid và tanh. Gần đây, người ta nhận thấy rằng ReLU cho kết quả tốt hơn trong những mạng học sâu, giải thích bởi lý thuyết gọi là vanishing gradient.

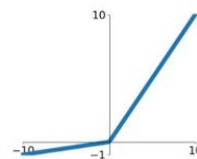
Sigmoid

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



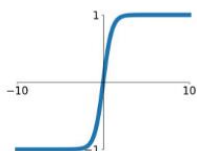
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

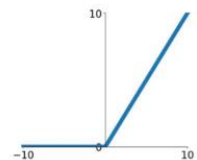


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

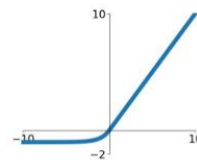
ReLU

$$\max(0, x)$$



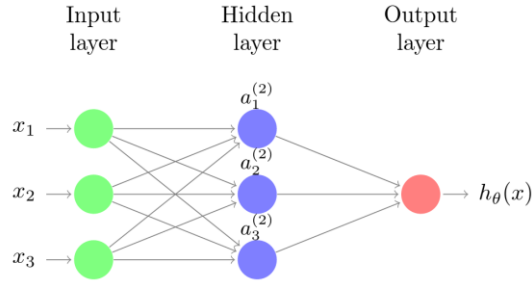
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Hình 5. Activation function

Ví dụ: Cho mạng neural network có đầu vào và các hệ số như hình bên, activation function là sigmoid, tính đầu ra của hệ thống.



Hình 6. Feed forward

$$a_1^{(2)} = g \left(m_{11}^{(1)} x_1 + m_{12}^{(1)} x_2 + m_{13}^{(1)} x_3 + b_1^{(1)} \right)$$

$$a_2^{(2)} = g \left(m_{21}^{(1)} x_1 + m_{22}^{(1)} x_2 + m_{23}^{(1)} x_3 + b_2^{(1)} \right)$$

$$a_3^{(2)} = g \left(m_{31}^{(1)} x_1 + m_{32}^{(1)} x_2 + m_{33}^{(1)} x_3 + b_3^{(1)} \right)$$

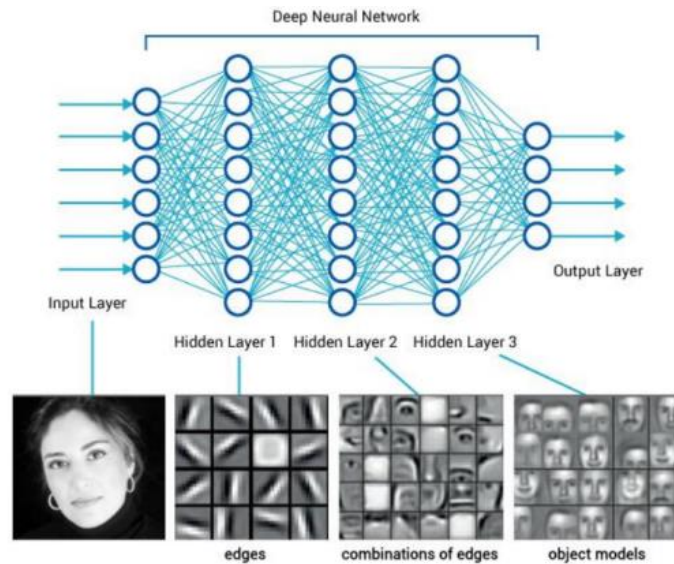
$$h = g \left(m_{11}^{(2)} x_1 + m_{12}^{(2)} x_2 + m_{13}^{(2)} x_3 + b_1^{(2)} \right)$$

Trong đó, g là hàm kích hoạt.

2.5. Fully connected layer

Một lớp mạng được gọi là fully connected khi mỗi neuron ở của nó liên kết với tất cả các neuron ở lớp trước.

Hình dưới mô tả kết nối của trong một deep artificial neural network. Trong đó, có 4 lớp fully connected là Hidden Layer 1, Hidden Layer 2, Hidden Layer 3 và Output Layer.



Hình 7. Fully connected neural network

2.6. Loss function

Loss function (kí hiệu là L) là hàm đánh giá độ chính xác của model, thể hiện khoảng cách giữa đầu ra của model và đầu ra mong muốn. Do đó, loss function có giá trị càng nhỏ càng tốt.

Loss function thành phần cốt lõi của evaluation function và objective function. Cụ thể, trong công thức thường gặp:

$$L_D(f_w) = \frac{1}{|D|} \sum_{(x,y) \in D} L(f_w(x), y)$$

Loss function trả về một số thực không âm thể hiện sự chênh lệch giữa hai đại lượng: \hat{y} (label được dự đoán) và y label đúng. Loss function giống như một hình thức để bắt model đóng phạt mỗi lần nó dự đoán sai, và số mức phạt tỉ lệ thuận với độ trầm trọng của sai sót. Trong mọi bài toán supervised learning, mục tiêu của ta luôn bao gồm giảm thiểu tổng mức phạt phải đóng. Trong trường hợp lý tưởng, loss function sẽ trả về giá trị cực tiểu bằng 0.

Cách xây dựng loss function:

Vì loss function đo đặc khoảng cách giữa y và \hat{y} nên không lạ gì nếu ta nghĩ ngay đến việc lấy trị tuyệt đối hiệu giữa chúng:

$$L(\hat{y}, y) = |\hat{y} - y|$$

Tuy nhiên loss function này không thuận tiện trong việc cực tiểu hóa, bởi vì đạo hàm của nó không liên tục (nhớ là đạo hàm của $f(x) = |x|$ không liên tục tại $x = 0$). Các phương pháp cực tiểu hóa hàm số thông dụng thường đòi hỏi phải tính được đạo hàm. Do đó, ta lấy hiệu hai bình phương:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Khi tính đạo hàm theo \hat{y} , ta được $\nabla L = \frac{1}{2} \times 2 \times (\hat{y} - y) = \hat{y} - y$. Có thể thấy rằng hằng số $\frac{1}{2}$ được thêm vào chỉ để cho công thức đạo hàm được đẹp hơn, không có hằng số phụ. Loss function này được gọi là **square loss**. Square loss có thể được sử dụng cho cả regression và classification, nhưng thực tế thì nó thường được dùng cho regression hơn.

Đối với binary classification, ta có một cách tiếp cận khác để xây dựng loss function. Nhắc lại là đối với dạng bài này, thì nếu model trả về $\hat{y} < 0$ tức là thích đáp án -1 hơn, trả về $\hat{y} > 0$ tức là thích đáp án +1 hơn.

Một cách rất tự nhiên, ta thấy rằng loss function của binary classification cần phải đạt được một số tiêu chí sau:

- Ta cần phải phạt model nhiều hơn khi dự đoán sai hơn là khi dự đoán đúng. Vì thế, tiêu chí đầu tiên của ta là khi model dự đoán sai (y khác dấu với \hat{y}), loss function phải trả về giá trị lớn hơn so với khi model dự đoán đúng (y cùng dấu với \hat{y}).
- Nếu có hai đáp án \hat{y}_1 và \hat{y}_2 đều cùng dấu (hoặc khác dấu) với y thì ta nên phạt đáp án nào nhiều hơn? Như đã nói, giá trị tuyệt đối $|\hat{y}|$ thể hiện "độ thích" của model đối với một phương án. Giá trị này càng lớn thì model càng "thích" một phương án. Trong trường hợp \hat{y} cùng dấu với y , phương án được thích là phương án đúng, do đó, model càng thích thì ta phải càng khuyến khích và phạt ít đi. Cũng với lập luận như vậy, nếu \hat{y} khác dấu với y , vì phương án được thích là phương án sai nên model càng thích thì ta phải càng phạt nặng để model không tái phạm nữa.

Một cách tổng quát, đối với binary classification thì các loss function thường có dạng như sau:

$$L(\hat{y}, y) = f(y \cdot \hat{y})$$

trong đó f là một hàm không âm và không tăng.

Logistic loss (hay log loss)

$$L_{\log}(\hat{y}, y) = \log_2(1 + \exp(-y \cdot \hat{y}))$$

Trong công thức trên, hàm $\exp(\)$ là hàm lũy thừa theo cơ số tự nhiên. Khi nhìn vào đồ thị của hàm số này, ta thấy nó thỏa tất cả mọi tính chất của loss function mà ta đã nói ở phần trước. Đây là một hàm liên tục, không âm và không tăng. Không những không tăng, log loss còn luôn giảm, có nghĩa là nó luôn phân biệt giữa các dự đoán có độ thích khác nhau bất kể đúng hay sai.

2.7. Back propagation

Phương pháp phổ biến nhất để train một artificial neural network là Gradient Descent (GD). Để áp dụng GD, chúng ta cần tính được gradient của hàm loss theo từng ma trận trọng số $W^{(l)}$ và vector bias $b^{(l)}$.

$$\begin{aligned}z^{(l)} &= W^{(l)T} a^{(l-1)} + b^{(l)} \\ \hat{y}^{(l)} &= f(z^{(l)}) \\ a^{(l)} &= \hat{y}^{(l)}\end{aligned}$$

Bước này được gọi là feed forward vì cách tính toán được thực hiện từ đầu đến cuối của network.

Giả sử $J(W, b, X, Y)$ là hàm loss của bài toán, trong đó W, b là tập hợp các ma trận trọng số giữa các layer và bias của mỗi layer, X, Y là cặp dữ liệu huấn luyện với mỗi cột tương ứng với một điểm dữ liệu. Để áp dụng gradient descent, chúng ta cần tính được:

$$\frac{\partial J}{\partial W^{(l)}}; \frac{\partial J}{\partial b^{(l)}}, l = 1, 2, \dots, L$$

Một ví dụ của hàm mất mát là hàm Mean Square Error (MSE) tức trung bình của bình phương lỗi.

$$J(W, b, X, Y) = \frac{1}{N} \sum_{n=1}^N \|y_n - \hat{y}_n\|_2^2 = \frac{1}{N} \sum_{n=1}^N \|y_n - a_n^{(L)}\|_2^2$$

Với N là số cặp dữ liệu (x, y) trong tập training.

Theo những công thức ở trên, việc tính toán trực tiếp giá trị này là cực kỳ phức tạp vì hàm mất mát không phụ thuộc trực tiếp vào các hệ số. Phương pháp phổ biến nhất được dùng có tên là Backpropagation giúp tính gradient ngược từ layer cuối cùng đến layer đầu tiên. Layer cuối cùng được tính toán trước vì nó gần gũi hơn với predicted outputs và hàm mất mát. Việc tính toán gradient của các layer trước được thực hiện dựa trên một quy tắc quen thuộc có tên là chain rule, tức đạo hàm của hàm hợp.

Stochastic Gradient Descent có thể được sử dụng để tính gradient cho các ma trận trọng số và biases dựa trên một cặp điểm training (x, y) . Để đơn giản, ta coi J là hàm mất mát nếu chỉ xét cặp điểm này, ở đây J là hàm mất mát bất kỳ, không chỉ hàm MSE như ở trên.

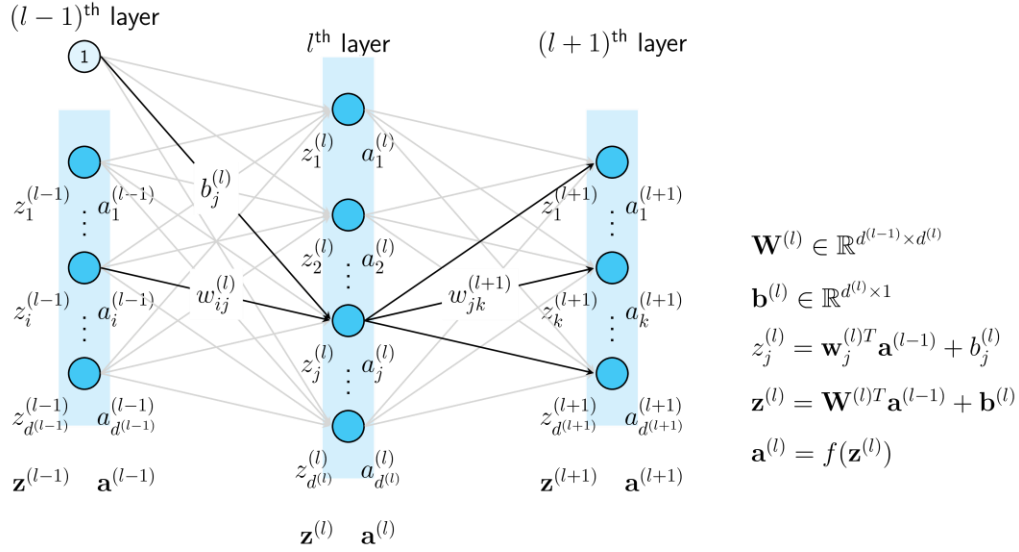
$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} a_i^{(L-1)}$$

Trong đó, $e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$ thường là một đại lượng dễ tính toán và $\frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = a_i^{(L-1)}$ vì

$$z_j^{(L)} = w_j^{(L)T} a_i^{(L-1)} + b_j^{(L)}.$$

Tương tự như thế, đạo hàm của hàm mất mát theo bias của layer cuối cùng là:

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$



Hình 8. Back propagation

Dựa vào hình trên, ta có thể tính được:

$$\frac{\partial J}{\partial \mathbf{w}_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial \mathbf{w}_{ij}^{(l)}} = e_j^{(l)} a_i^{(l-1)}$$

Với:

$$\begin{aligned}
 e_j^{(l)} &= \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\
 &= \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) f'(z_j^{(l)}) \\
 &= \left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} \mathbf{w}_{ik}^{(l+1)} \right) f'(z_j^{(l)}) \\
 &= \left(\mathbf{w}_{j:}^{(l+1)} e^{(l+1)} \right) f'(z_j^{(l)})
 \end{aligned}$$

Trong đó $e^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, \dots, e_{d^{(l+1)}}^{(l+1)}] \in \mathbb{R}^{d^{(l+1)} \times 1}$ và $\mathbf{w}_{j:}^{(l+1)}$ được hiểu là hàng thứ j của ma trận $\mathbf{W}^{(l+1)}$. (Chú ý dấu hai chấm, khi không có dấu này, mặc định ký hiệu nó cho vector cột).

Dấu sigma tính tổng ở hàng thứ hai trong phép tính trên xuất hiện vì $a_j^{(l)}$ đóng góp vào tất cả các $z_k^{(l+1)}, k = 1, 2, \dots, d^{(l+1)}$. Biểu thức đạo hàm ngoài dấu ngoặc là vì $a_j^{(l)} = f(z_j^{(l)})$. Do đó, ta có thể thấy rằng, việc activation function có đạo hàm đơn giản sẽ có ích nhiều trong việc tính toán.

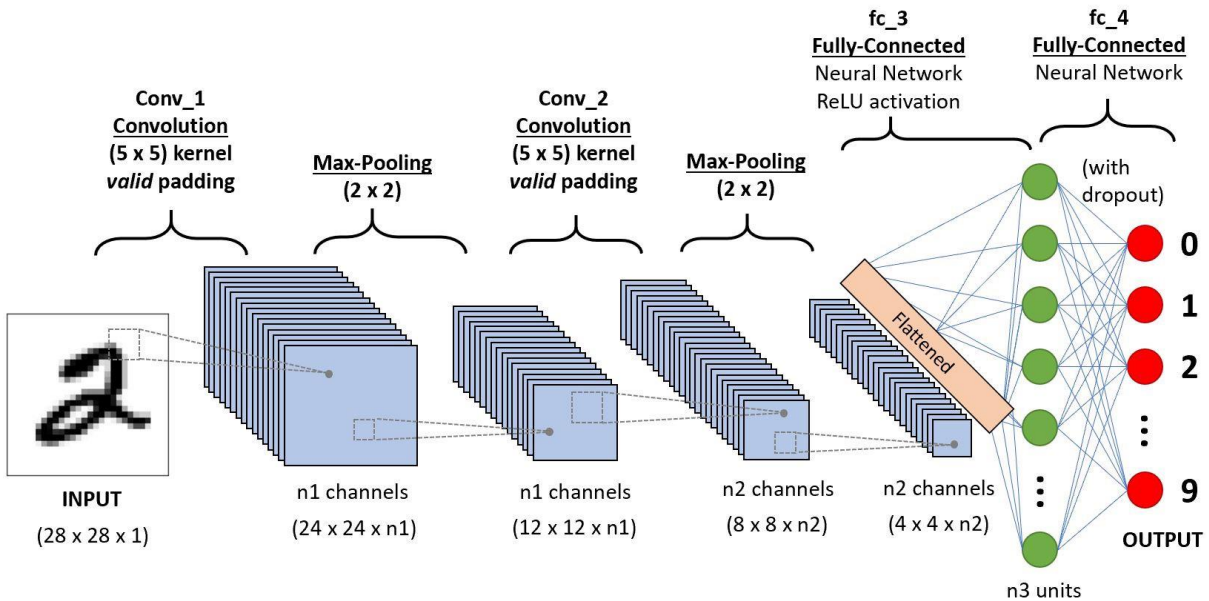
Với cách làm tương tự, ta có thể suy ra:

$$\frac{\partial J}{\partial \mathbf{b}_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$

Nhận thấy rằng trong các công thức trên đây, việc tính các $e_j^{(l)}$ đóng một vai trò quan trọng. Hơn nữa, để tính các giá trị này, ta cần tính được $e_j^{(l+1)}$. Nói cách khác, ta cần tính ngược từ cuối. Cái tên backpropagation cũng xuất phát từ việc này.

2.8. Convolutional neural layer

Convolution Neural Network là thuật toán học sâu với đầu vào là ảnh. Trước đây, ta sử dụng các bộ tách features thuộc loại hard-coded như HOG. Tuy nhiên, với CNN, các feature được tách ra bằng các lớp convolution. Sau đó, các features được đưa qua fully connected layer để thực hiện mục tiêu tính toán.



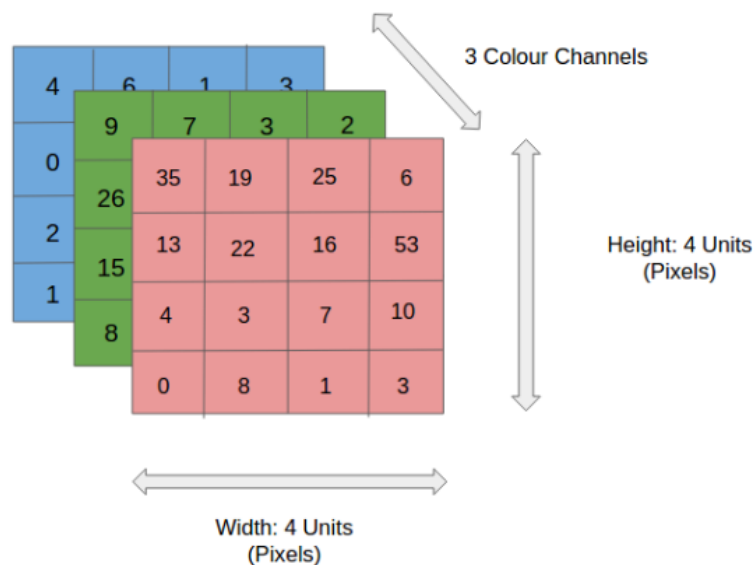
Hình 9. Convolutional neural network

Một bức ảnh là một ma trận với các phần tử là các giá trị của các điểm ảnh. Do đó, tại sao chúng ta không trải phẳng ma trận hình ảnh (ảnh 3x3 thành vector 9x1) và cho vào một fully connected layer cho việc classification.

Trong trường hợp một bức ảnh binary đơn giản, phương pháp này có thể cho ra một kết quả tương đối chính xác (ta có thể sử dụng phương pháp này để nhận dạng chữ số đối với MNIST dataset). Tuy nhiên, với những bức ảnh phức tạp, phương pháp này không đem lại kết quả.

Một mạng CNN có khả năng phân tích các features từ ảnh input. Cấu trúc này thích hợp hơn cho tập dữ liệu hình ảnh vì nó giúp giảm số lượng parameter và số lượng parameter được sử dụng lại. Cấu trúc CNN có thể được train để hiểu được các bức ảnh phức tạp một cách tốt hơn.

Ảnh đầu vào:



Hình 10. Input image

Trong ảnh trên, ta có một bức ảnh RGB được tách ra thành 3 mặt phẳng màu (Đỏ, Xanh lá và xanh dương).

Chúng ta có thể thấy rằng khối lượng tính toán cho một bức ảnh 8K (7680x4320) là rất lớn. Nhiệm vụ của CNN là giảm kích thước hình ảnh xuống tới kích thước dễ xử lý hơn mà không mất đi các features quan trọng cần thiết cho một kết quả dự đoán chính xác.

Convolution Layer — The Kernel:

Ví dụ ta có ma trận input là ma trận 5x5, một channel và ma trận kernel, như sau:

$$I = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}, K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Kết quả của Convolute ma trận I (5x5x1) với kernel matrix (3x3x1) là một ma trận 3x3x1.

Để thực hiện phép convolution, ma trận kernel K phải được xoay 180 độ, tuy nhiên, vì ma trận K đối xứng nên kết quả vẫn được ma trận K.

Sau đó, ta đưa kernel K quét trong ma trận I, ở mỗi vị trí, ta thực hiện phép nhân từng phần tử và cộng các kết quả lại. Quá trình được mô tả bởi hình sau:

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

1	1 _{x1}	1 _{x0}	0 _{x1}	0
0	1 _{x0}	1 _{x1}	1 _{x0}	0
0	0 _{x1}	1 _{x0}	1 _{x1}	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved
Feature

1	1	1 _{x1}	0 _{x0}	0 _{x1}
0	1	1 _{x0}	1 _{x1}	0 _{x0}
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Convolved
Feature

1	1	1	0	0
0 _{x1}	1 _{x0}	1 _{x1}	1	0
0 _{x0}	0 _{x1}	1 _{x0}	1	1
0 _{x1}	0 _{x0}	1 _{x1}	1	0
0	1	1	0	0

Image

4	3	4
2		

Convolved
Feature

1	1	1	0	0
0	1 _{x1}	1 _{x0}	1 _{x1}	0
0	0 _{x0}	1 _{x1}	1 _{x0}	1
0	0 _{x1}	1 _{x0}	1 _{x1}	0
0	1	1	0	0

Image

4	3	4
2	4	

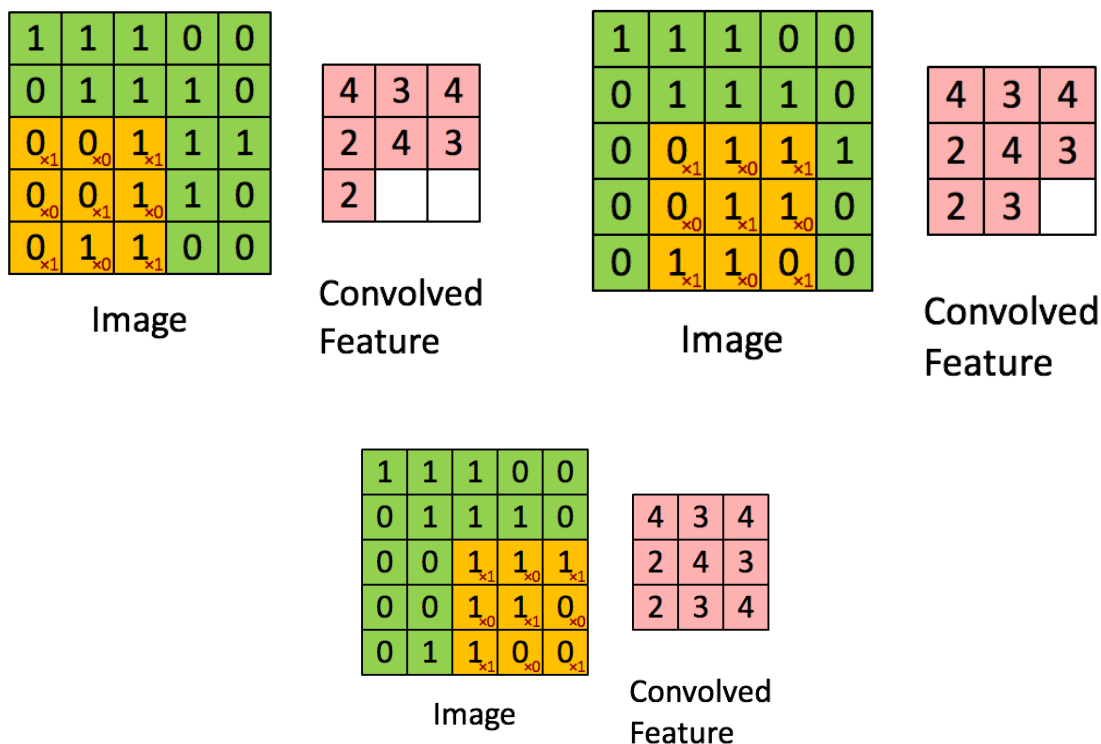
Convolved
Feature

1	1	1	0	0
0	1	1 _{x1}	1 _{x0}	0 _{x1}
0	0	1 _{x0}	1 _{x1}	1 _{x0}
0	0	1 _{x1}	1 _{x0}	0 _{x1}
0	1	1	0	0

Image

4	3	4
2	4	3

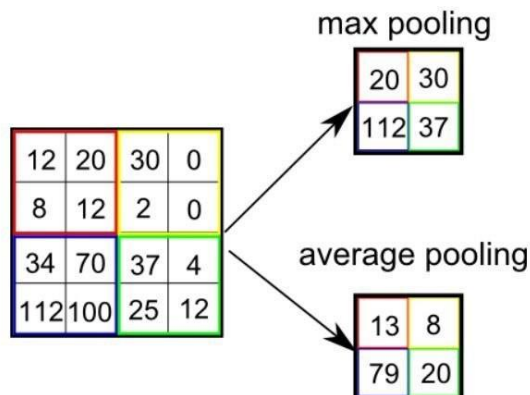
Convolved
Feature



2.9. Pooling layer

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

Gọi pooling size kích thước $K \times K$. Input của pooling layer có kích thước $H \times W \times D$, ta tách ra làm D ma trận kích thước $H \times W$. Với mỗi ma trận, trên vùng kích thước $K \times K$ trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả.



Hình 11. Thuật toán pooling

Chương 3: Xây dựng cấu trúc mạng neuron

3.1. Tích chập với kernel one-by-one [1x1]

Một cách đơn giản, 1x1 convolution giúp giảm số chiều dữ liệu. Ví dụ, sử dụng 20 filters 1x1 cho một ảnh 200x200 với 50 features, ta được một ma trận có kích thước mới là 200x200x20. 1x1 convolution là cách tốt nhất để giảm số chiều dữ liệu trong mạng CNN.

3.1.1. Biến đổi features

Mặc dù 1x1 convolution là một kỹ thuật “feature pooling”, nó có nhiều ý nghĩa hơn là chỉ tính tổng features của các channel trong một lớp. 1x1 convolution có thể được hiểu như một phép biến đổi phụ thuộc vào hệ trục tọa độ trong không gian lọc (filter space). Một điều quan trọng là phép biến đổi này tuyến tính tuyệt đối, nhưng trong đa số trường hợp ứng dụng, sau 1x1 convolution, ta lại sử dụng một hàm kích hoạt phi tuyến như ReLU. Phép biến đổi này được huấn luyện bằng (stochastic) gradient descent. Nhưng một khác biệt quan trọng là nó giúp giảm khả năng overfitting do kernel size nhỏ (1x1).

3.1.2. Giúp mạng thần kinh sâu hơn

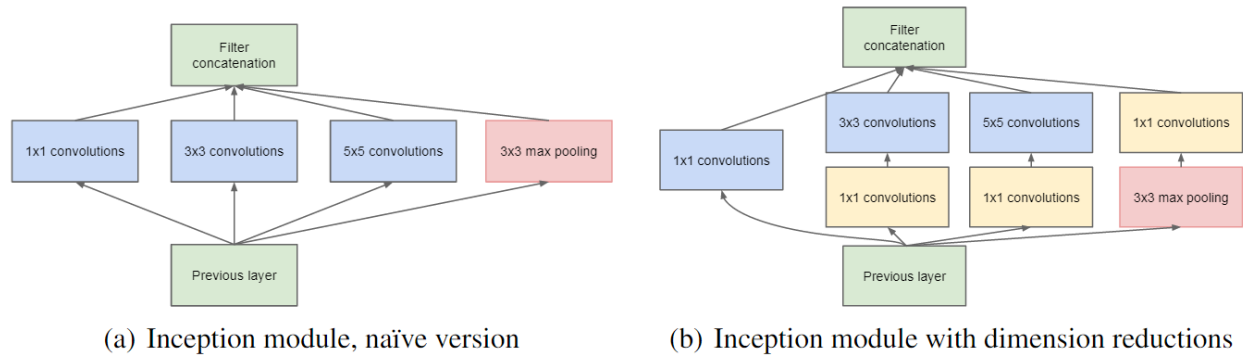
1x1 convolution được giới thiệu lần đầu tiên trong paper có tên là “Network in network”. Trong paper này, mục tiêu của tác giả là tạo ra một mạng sâu hơn mà không đơn giản là bổ sung thêm nhiều layer hơn. Tác giả đã thay thế một vài filter với một lớp perceptron nhỏ hơn và kết hợp với một số 1x1 and 3x3 convolutions. Theo cách này, network có thể xem như là “rộng hơn” thay vì là “sâu hơn”. Một điều cần chú ý là trong machine learning, rộng hơn thường đồng nghĩa với việc cần nhiều dữ liệu để train hơn. Tập các 1x1(xF) convolutions, một cách toán học, tương đương với multi-layer perceptron.

3.1.3 Sử dụng 1x1 kernel trong Inception module

Trong cấu trúc GooLeNet, 1x1 convolution được sử dụng với mục đích:

- Giúp mạng sâu hơn bằng cách thêm các inception module giống như Network in Network paper được mô tả ở trên.
- Giảm số chiều bên trong các inception module

Để tăng tính phi tuyến, RELU được sử dụng sau mỗi 1x1 convolution.



Hình 12. Cấu trúc Inception module

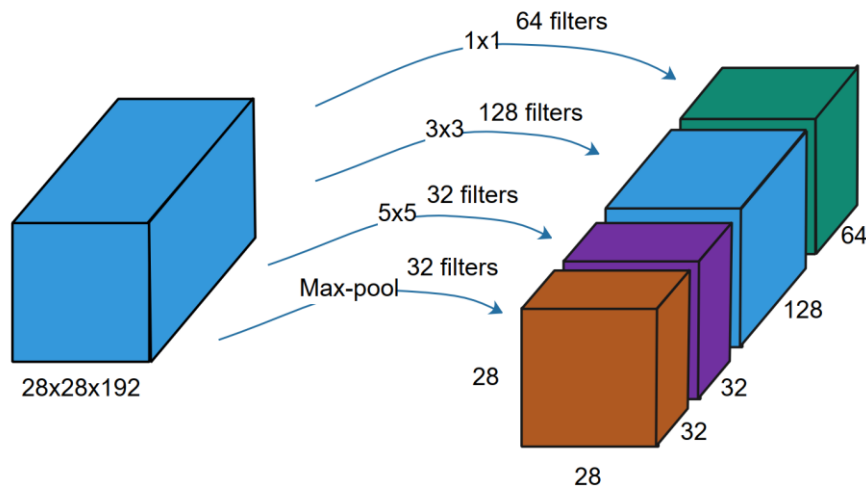
Từ hình b, có thể thấy rằng, 1x1 convolution (màu vàng) được sử dụng trước 3x3 và 5x5 convolution để giảm số chiều. Chú ý rằng, hai bước convolution liên tiếp có thể được gộp lại thành một bước convolution tương đương. Nhưng trong trường hợp này và trong đa số trường hợp trong mạng deep neural network, convolution được theo sau bởi các hàm kích hoạt phi tuyến, do đó, mất đi tính tuyến tính và không thể kết hợp thành một phép toán tương đương được nữa.

Trong thiết kế network kiểu này, cần chú ý rằng convolution kernel khởi tạo phải có kích thước lớn hơn 1x1 để có khả năng học được thông tin về không gian.

3.2. Inception topology

Khi thiết kế một lớp mạng convolution trong cấu trúc của mạng thần kinh nhân tạo, ta thường phải lựa chọn nên sử dụng mạng 1x1, 3x3, 5x5 hay max-pooling.

Một ý tưởng nảy ra là tại sao ta không sử dụng đồng thời các loại lớp này và nối các kết quả lại với nhau. Đây chính là ý tưởng sơ khai của inception module.



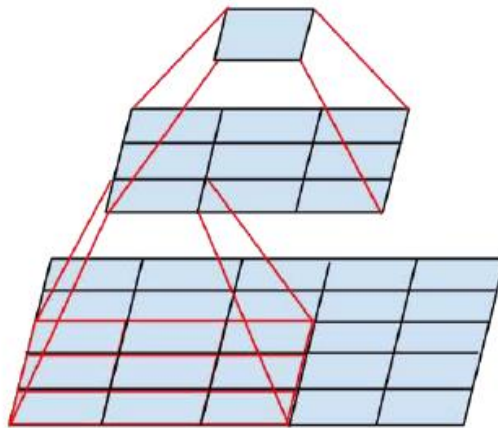
Hình 13. Ý tưởng của inception

3.2.1. Factorizing convolutions

Mục đích của factorizing convolutions là làm giảm số kết nối/tham số mà không làm giảm tính hiệu quả của mạng.

Factorization thành các convolution nhỏ hơn

Hai 3x3 convolution có thể thay thế cho 5x5 convolution.



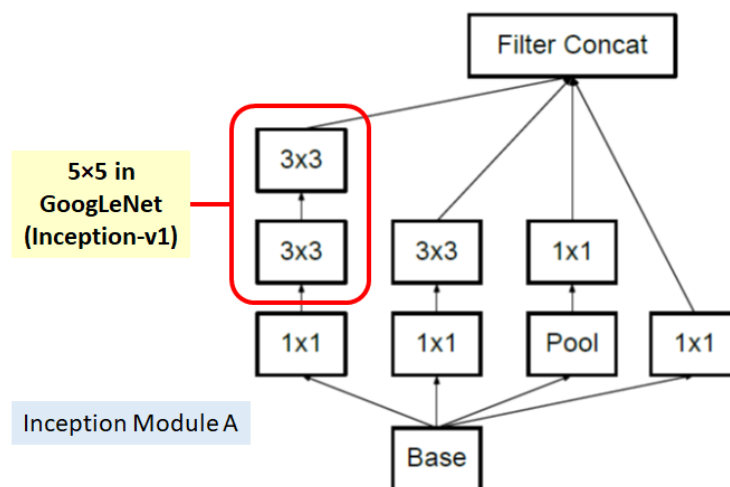
Hình 14. Thay thế filter 5x5 bởi 2 filter 3x3

Sử dụng 1 filter 5x5, ta có số tham số là $5 \times 5 = 25$.

Sử dụng 2 filter 3x3, ta có số tham số là $3 \times 3 + 3 \times 3 = 18$.

Việc sử dụng 2 filter 3x3 thay cho 1 filter 5x5 giúp số lượng tham số giảm 28%.

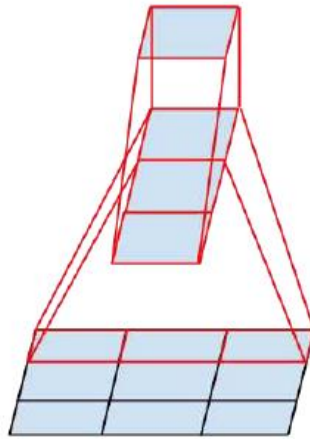
Với kỹ thuật này, ta có một inception module mới (inception module A):



Hình 15. Inception module A

Factorization thành Convolutions bất đối xứng

Một 3×1 convolution theo sau bởi 1×3 convolution có thể thay thế một 3×3 convolution:



Hình 16. Thay thế 3×3 filter bằng 3×1 và 1×3 filters

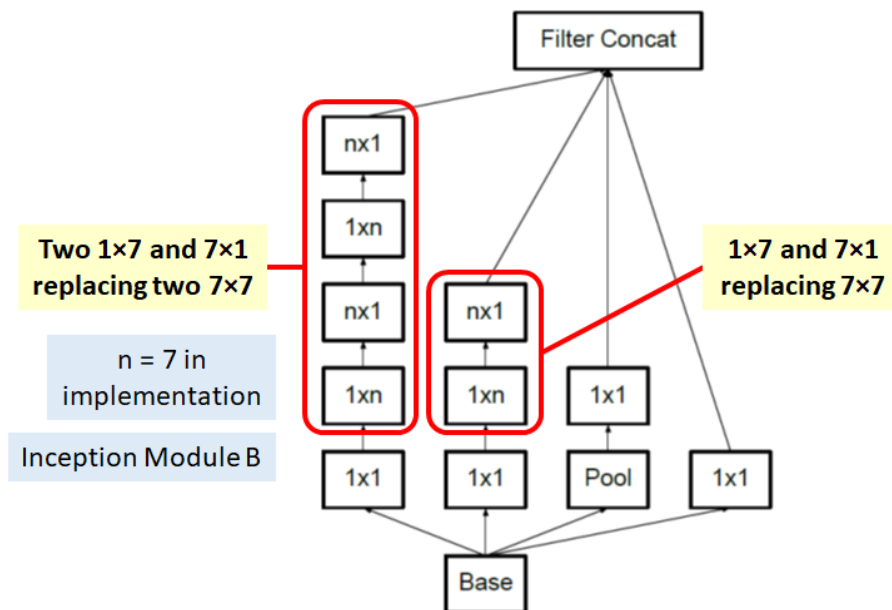
Sử dụng 3×3 filter, ta có số tham số là $3 \times 3 = 9$.

Sử dụng 3×1 và 1×3 filters, ta có số tham số là $3 \times 1 + 1 \times 3 = 6$.

Việc sử dụng 3×1 và 1×3 filters thay cho 1 filter 3×3 giúp số lượng tham số giảm 33%.

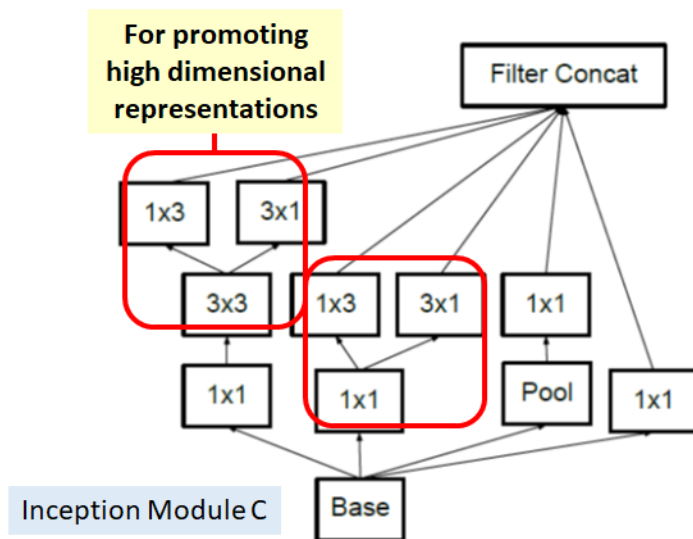
Có thể thấy rằng nếu thay thế 3×3 filter bởi hai 2×2 filters, số lượng tham số là $2 \times 2 \times 2 = 8$. Ta thấy số lượng tham số chỉ giảm 11%. Do đó ta không dùng cách này.

Với kỹ thuật này, ta có một inception modules mới (inception module B):



Hình 17. Inception module B

Inception module C với nhiệm vụ promote high dimensional representations được mô tả như sau:



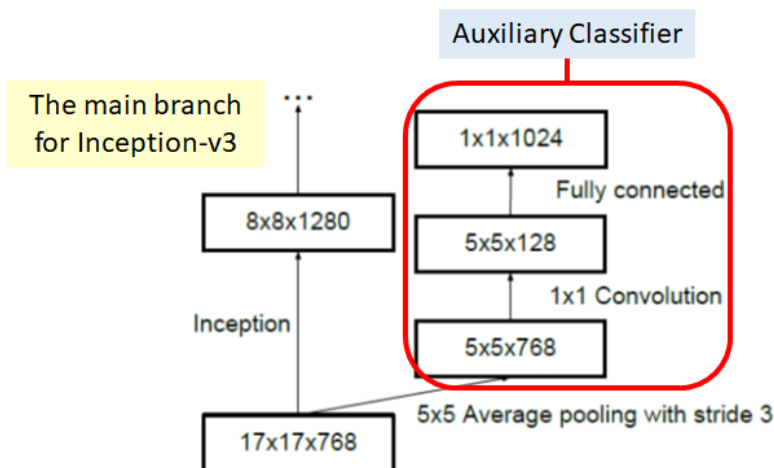
Hình 18. Inception module C

Tác giả đề nghị 3 loại inception module. Với factorization, số lượng tham số của cả network giảm đi khá nhiều, giảm khả năng bị overfitting và giúp network có thể được thiết kế sâu hơn.

3.2.2. Auxiliary classifier

Auxiliary Classifiers đã được giới thiệu trong GoogLeNet / Inception-v1. Đến Inception-v3, đã có một số cải tiến nhất định.

Chỉ một Auxiliary Classifiers được sử dụng sau lớp 17x17 cuối cùng, thay vì 2 như trong các phiên bản trước.



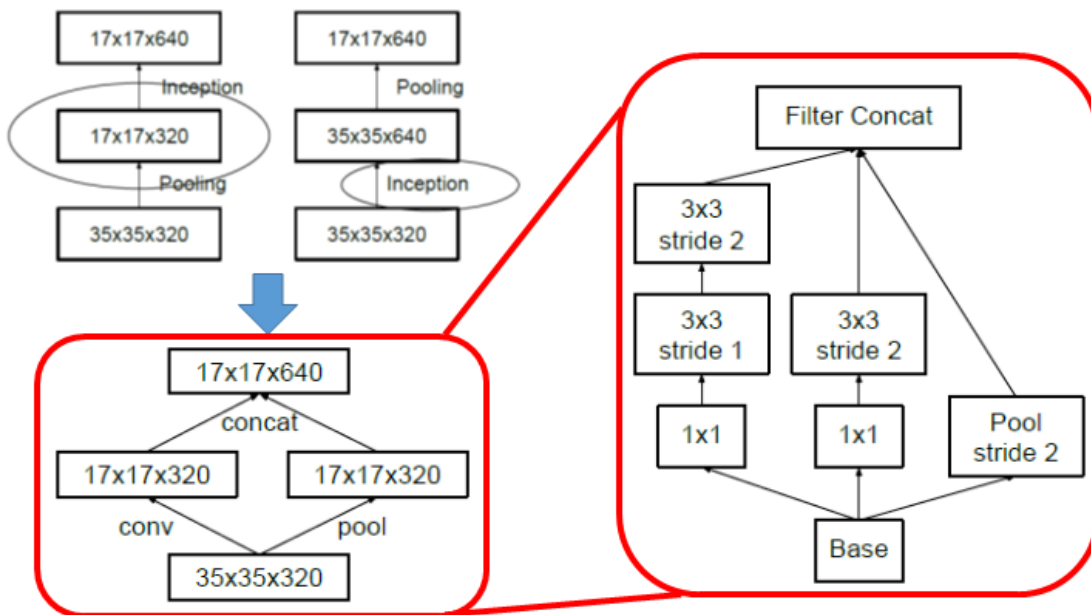
Hình 19. Auxiliary classifier

Chức năng của auxiliary classifiers cũng đã thay đổi. Trong GoogLeNet / Inception-v1, auxiliary classifiers được sử dụng để có mạng sâu hơn. Trong Inception-V3, auxiliary classifiers được sử dụng như một thông số điều chỉnh overfitting (regularizer).

Batch normalization cũng được sử dụng trong auxiliary classifier.

3.2.3. Efficient grid size reduction

Thông thường, feature map được downsizing bằng max pooling như trong AlexNet và VGGNet. Tuy nhiên, điều này gây ra hai vấn đề, một là giảm độ chính xác nếu cấu hình max pooling theo sau bởi convolution, hai là tăng khối lượng tính toán nếu cấu hình convolution theo sau bởi max pooling. Ở đây, chúng ta có một cách hiệu quả hơn để giảm kích thước như sau:



Hình 20. Efficient grid size reduction

Ở phía ở góc trên trái, ta có các thông thường để downsizing.

Efficient grid size reduction được thể hiện ở góc dưới trái.

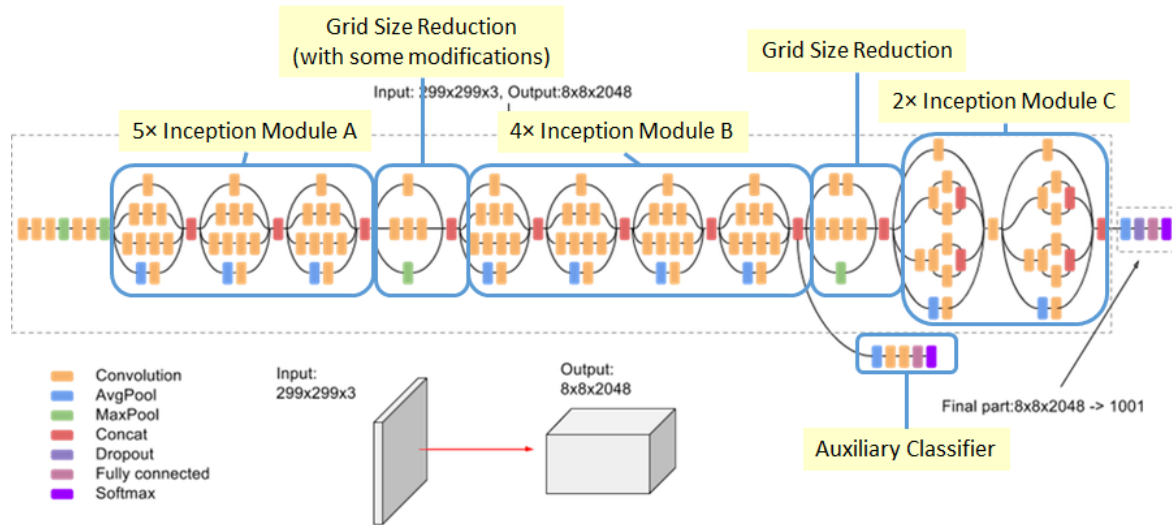
Ở bên phải, ta có cấu trúc chi tiết của efficient grid size reduction.

Với efficient grid size reduction, 320 feature maps là kết quả của convolution và 320 features maps là kết quả của max pooling được nối lại và tạo thành 640 feature maps.

Efficient grid size reduction giúp giảm khối lượng tính toán trong khi vẫn giữ được hiệu quả của mạng.

3.2.4. Cấu trúc mạng Inception-V3

Có một số cấu hình được giới thiệu trong paper. Cấu hình sau được implement trong code.



Hình 21. Cấu trúc mạng Inception-V3

Với độ cấu hình như trên, Inception-V3 cần khối lượng tính toán cao gấp 2.5 lần GoogLeNet nhưng đem lại kết quả tốt hơn nhiều so với VGGNet.

3.2.5. Label smoothing as regularization

Mục tiêu của label smoothing là để tránh giá trị label lớn nhất lớn hơn rất nhiều so với các giá trị label còn lại.

$$\text{new_labels} = (1 - \epsilon) \times \text{one_hot_labels} + \epsilon / K$$

Trong đó:

new_labels là giá trị label mới.

ϵ là một hyperparameter và thường được chọn là 0.1.

one_hot_labels là label cũ.

K là số lượng class cần phân biệt.

Chương 4: Xây dựng mô hình và kết quả thực nghiệm

4.1. Chuẩn bị dataset

Dataset được sử dụng là **Plant Diseases Dataset**.

Dataset gồm 38 cặp lá cây, bệnh lá cây được label theo cú pháp:

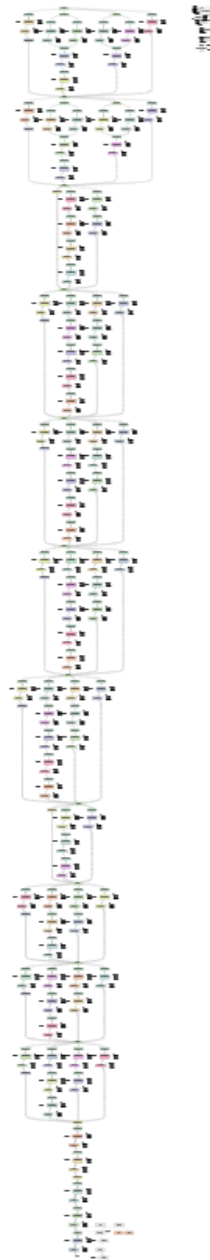
<tên cây>__<bệnh>

38 cặp lá cây, bệnh đó là:

STT	Tên tiếng Anh
0	Apple__Apple_scab
1	Apple__Black_rot
2	Apple__Cedar_apple_rust
3	Apple__healthy
4	Blueberry__healthy
5	Cherry_(including_sour)__Powdery_mildew
6	Cherry_(including_sour)__healthy
7	Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot
8	Corn_(maize)__Common_rust
9	Corn_(maize)__Northern_Leaf_Blight
10	Corn_(maize)__healthy
11	Grape__Black_rot
12	Grape__Esca_(Black_Measles)
13	Grape__Leaf_blight_(Isariopsis_Leaf_Spot)
14	Grape__healthy
15	Orange__Haunglongbing_(Citrus_greening)
16	Peach__Bacterial_spot
17	Peach__healthy
18	Pepper,_bell__Bacterial_spot
19	Pepper,_bell__healthy
20	Potato__Early_blight
21	Potato__Late_blight
22	Potato__healthy
23	Raspberry__healthy
24	Soybean__healthy
25	Squash__Powdery_mildew
26	Strawberry__Leaf_scorch
27	Strawberry__healthy
28	Tomato__Bacterial_spot
29	Tomato__Early_blight
30	Tomato__Late_blight

31	Tomato__Leaf_Mold
32	Tomato__Septoria_leaf_spot
33	Tomato__Spider_mites Two-spotted_spider_mite
34	Tomato__Target_Spot
35	Tomato__Tomato_Yellow_Leaf_Curl_Virus
36	Tomato__Tomato_mosaic_virus
37	Tomato__healthy

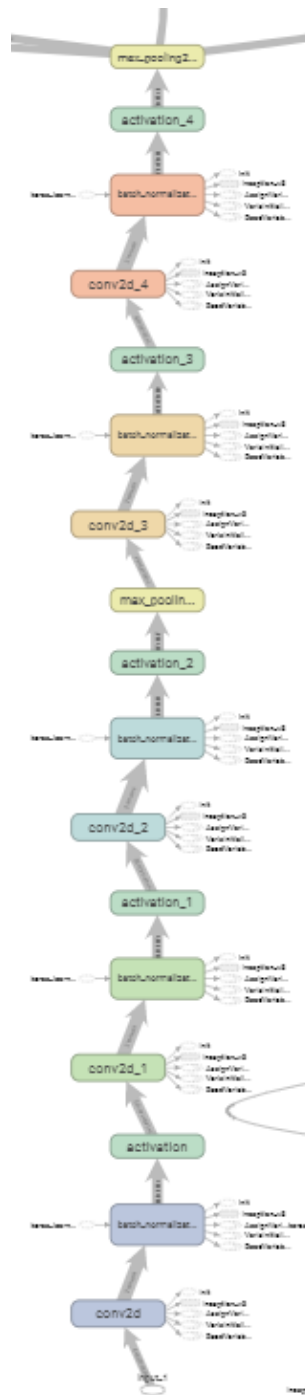
4.2. Xây dựng mô hình mạng dựa trên TensorFlow framework



Hình 22. Cấu trúc của mạng

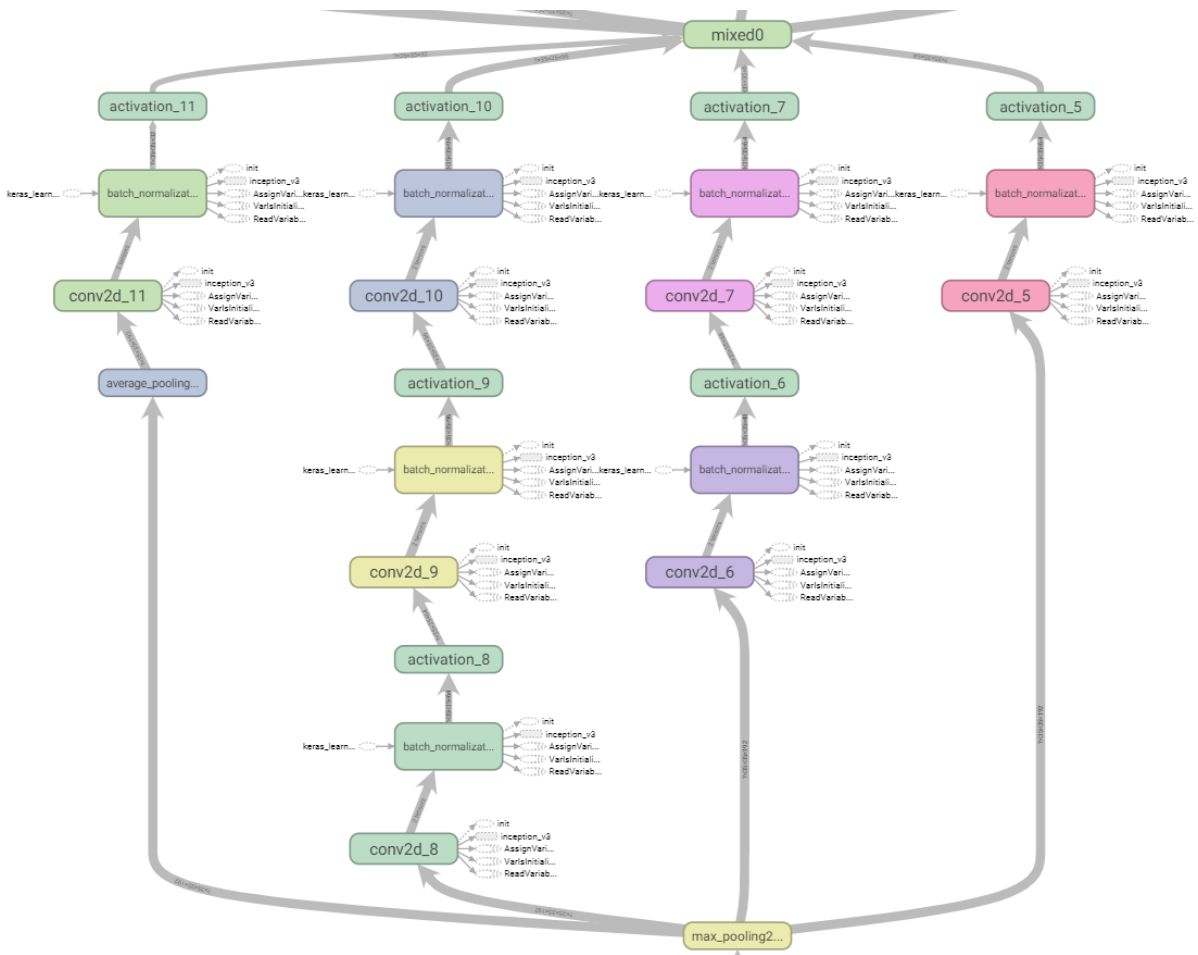
Hình trên mô tả cấu trúc của mạng neuron được implement bằng TensorFlow, qua đó, giúp ta hình dung được kích thước của mạng neuron dùng cho bài toán. Các thành phần cụ thể sẽ được phân tích sau.

Đầu tiên ta có 5 lớp tích chập:



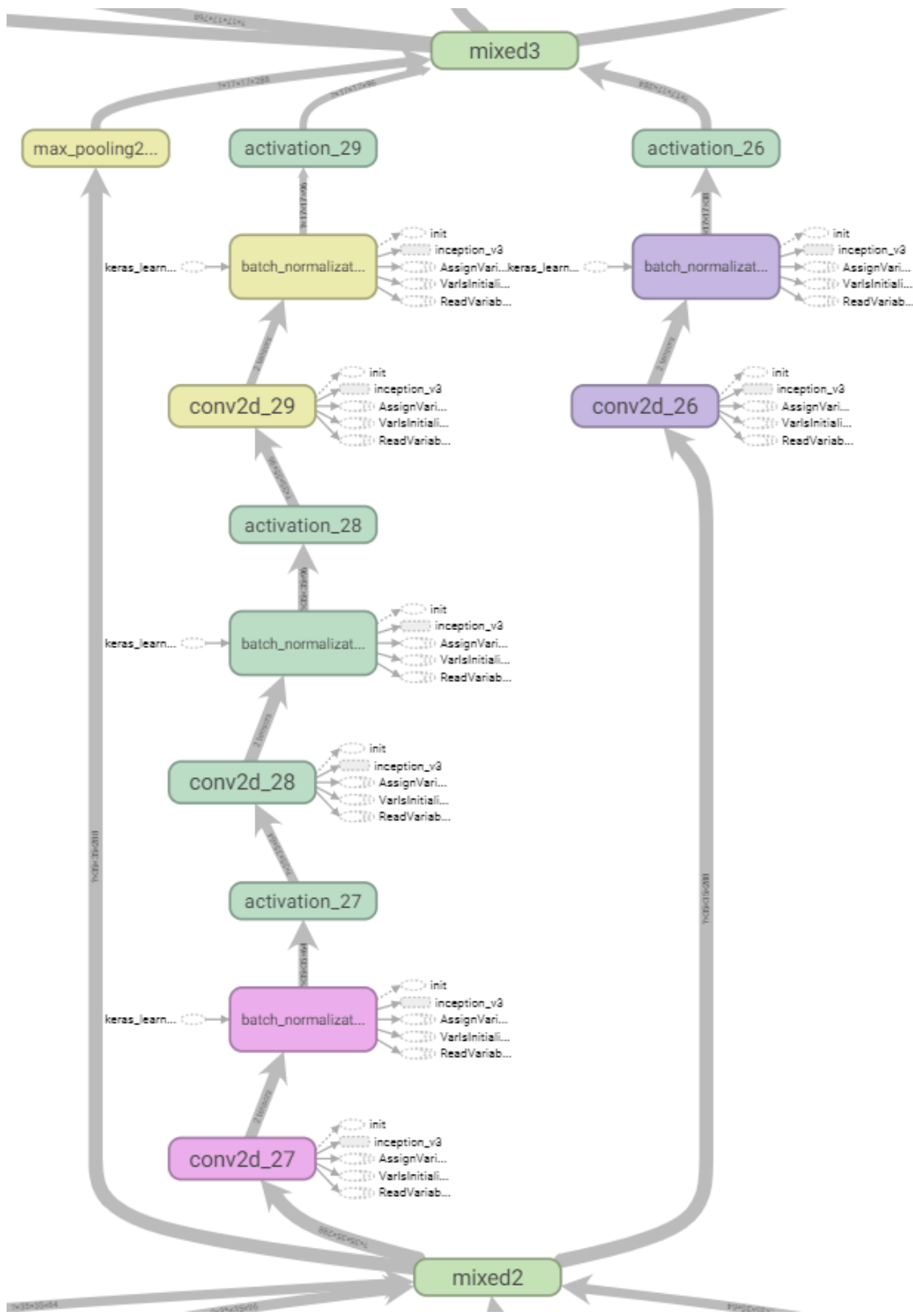
Hình 23. Các lớp tích chập đầu tiên

Sau đó, ta có 3 Inception module A:



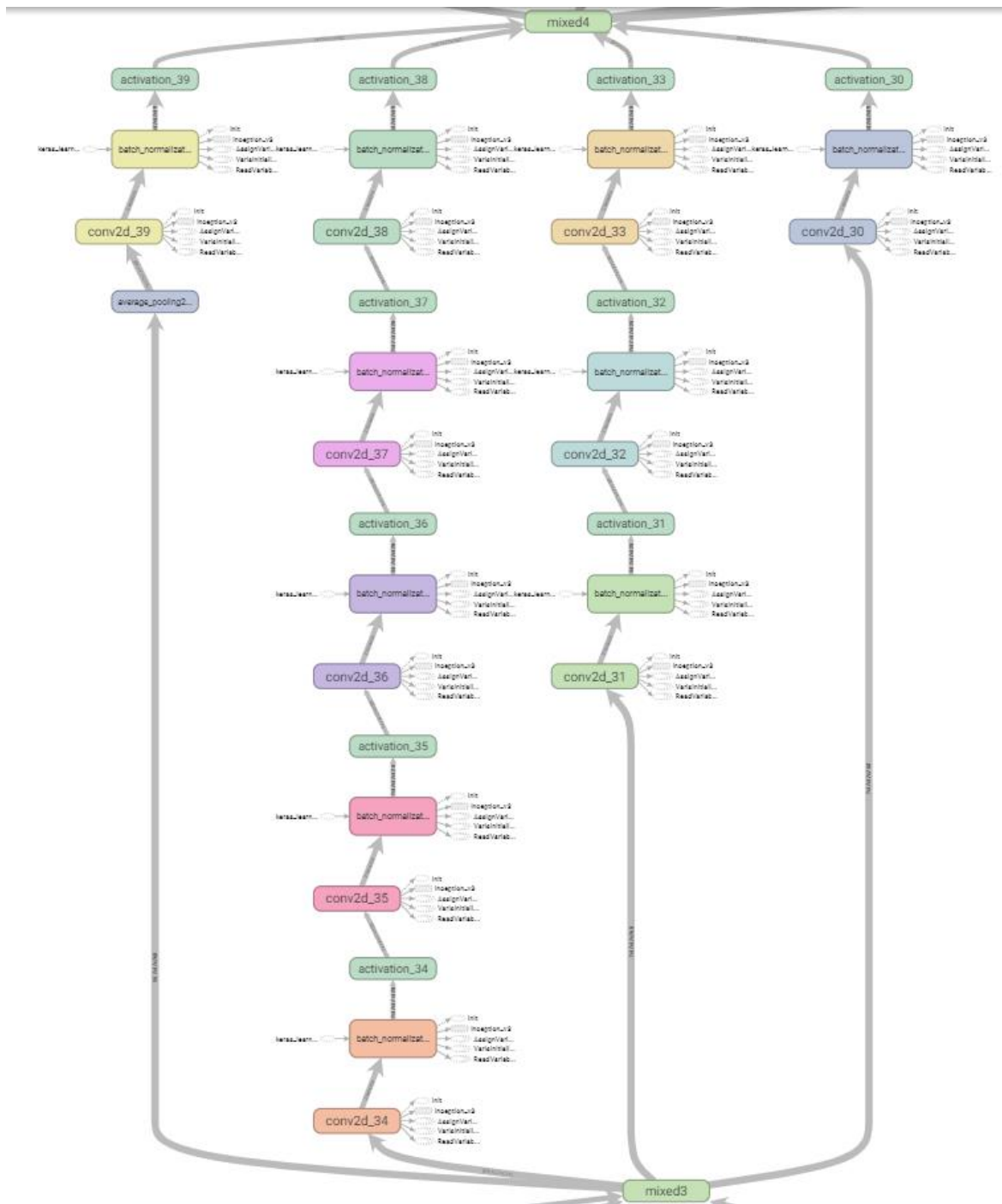
Hình 24. Inception module A

Sau đó là một lớp grid size reduction:



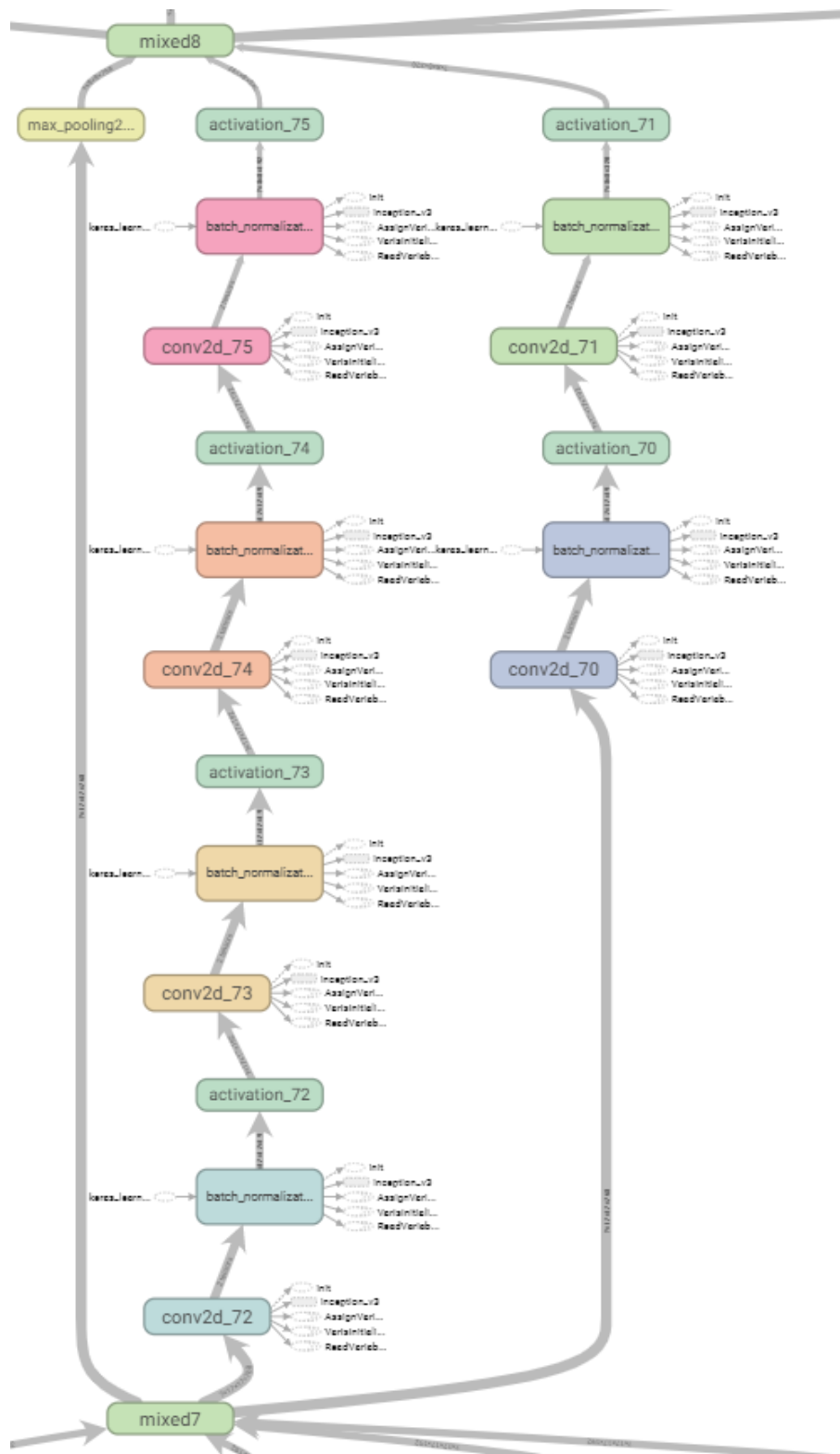
Hình 25. Grid size reduction

Tiếp theo là 4 lớp Inception module B:



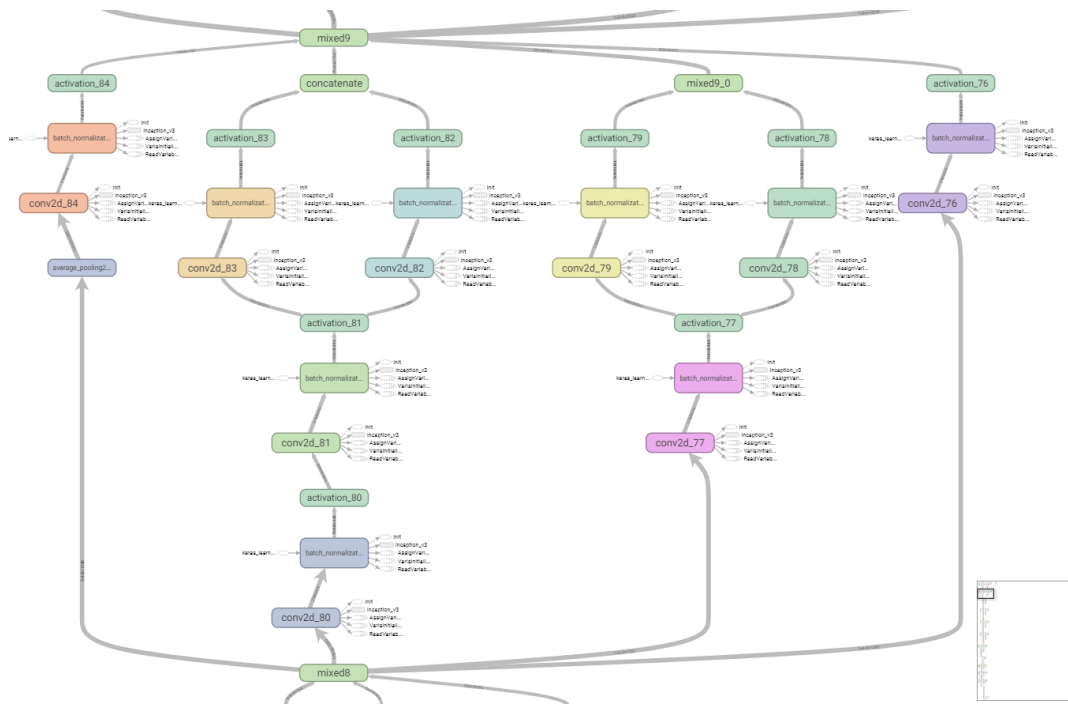
Hình 26. Inception module B

Sau đó ta lại có thêm một lớp grid size reduction nữa:



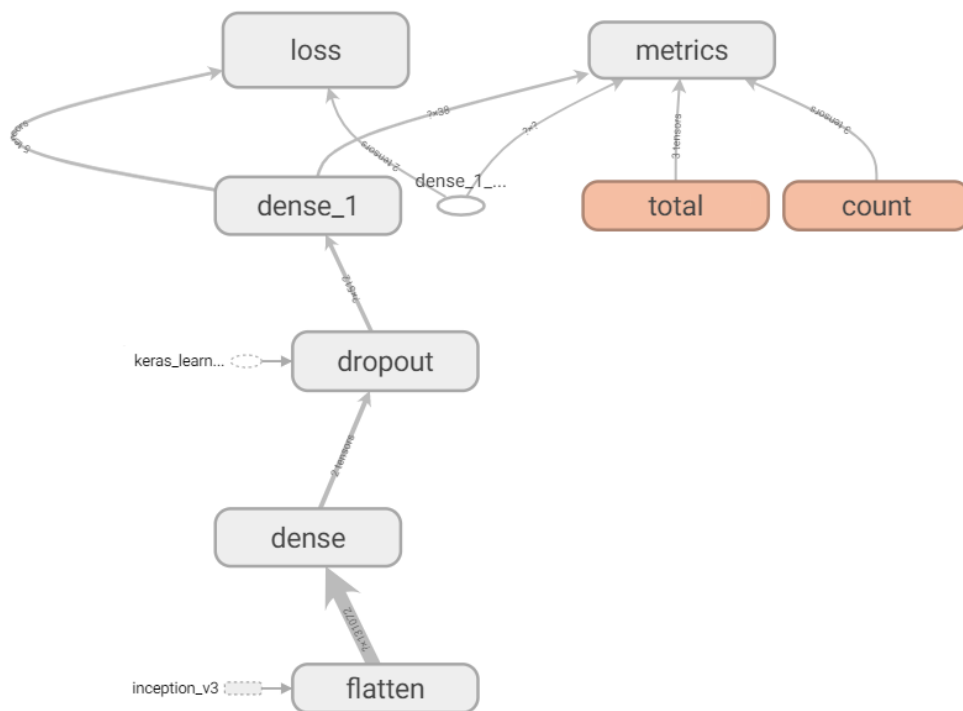
Hình 27. Grid size reduction

Cuối cùng của Inception V3 là 2 lớp Inception module C:



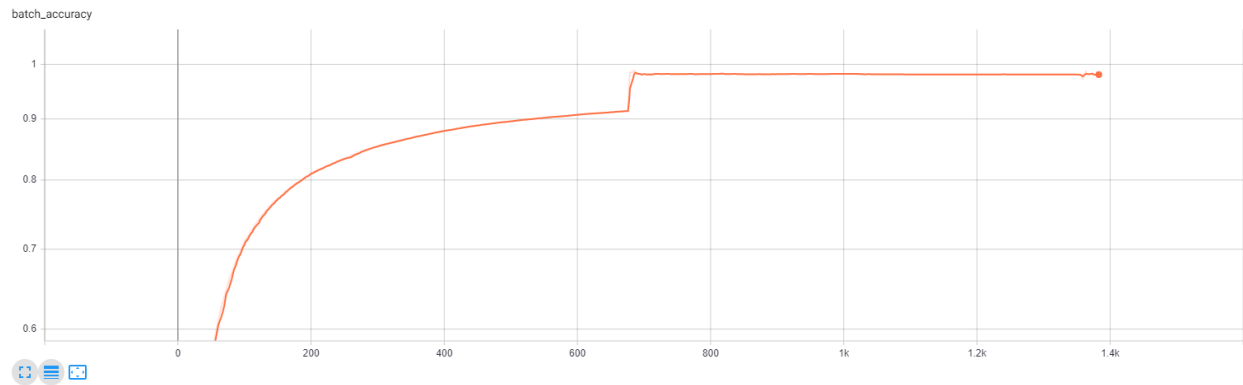
Hình 28. Inception module C

Để đưa ra kết quả phân loại, ta dùng 2 lớp fully connected layer:

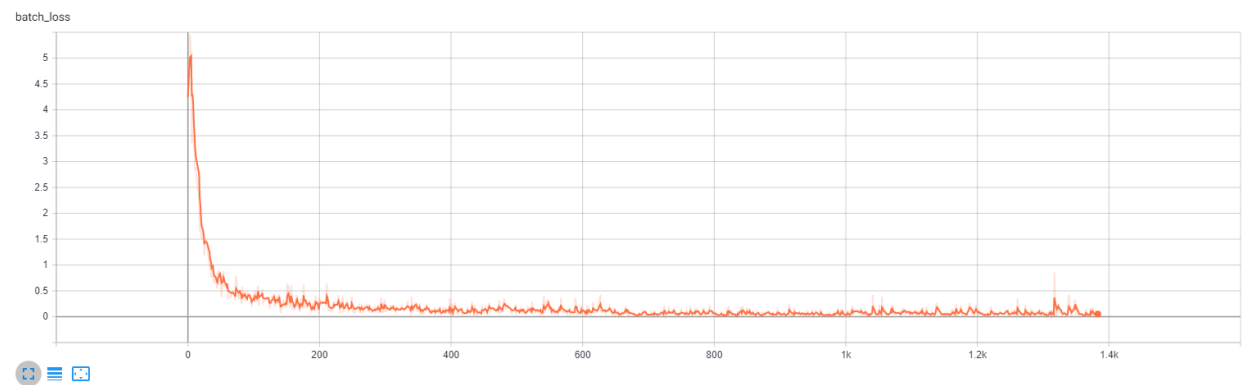


Hình 29. Fully connect

4.3. Đồ thị hàm loss trong quá trình train



Hình 30. Batch accuracy



Hình 31. Loss function

Sau 3 epochs (1854 batches) ta thấy batch accuracy tiến gần đến 1 và batch loss tiến gần về 0.

Giá trị loss cuối cùng trên tập train là 0.0724.

Giá trị batch accuracy cuối cùng trên tập train là 0.9804.

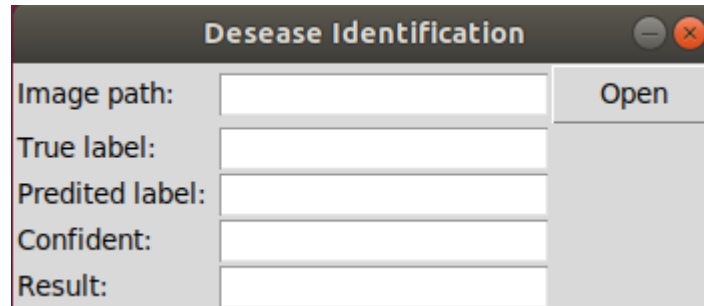
4.4. Đánh giá trên tập validation

Test loss: 0.38438466123360043

Test acc: 0.90295553

4.5. Xây dựng giao diện phần mềm.

Giao diện phần mềm **Disease Identification**:



The screenshot shows a window titled "Disease Identification" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are five input fields stacked vertically, each with a label to its left: "Image path:", "True label:", "Predited label:", "Confident:", and "Result:". To the right of the "Image path:" field is a button labeled "Open". The fields are currently empty.

Hình 32. Graphic user interface

Giao diện gồm có:

Button “Open”: nút mở dialog để chọn đường dẫn tới file ảnh của lá cây.

Image path: đường dẫn đến ảnh của lá cây.

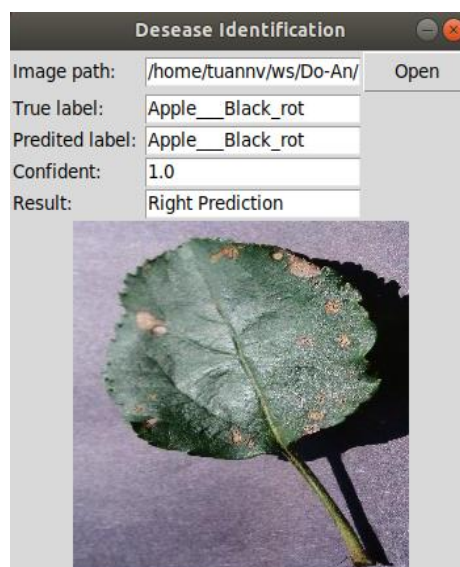
True label: nhãn đúng cho lá cây.

Predicted label: nhãn do machine learning model dự đoán.

Confident: Độ tự tin của model với dự đoán của nó, có giới hạn giá trị trong khoảng từ 0 đến 1, thường có giá trị gần 1.

Result: Phần mềm tự so sánh “True label” và “Predicted label” để đưa ra kết luận model chạy đúng hay sai. Nếu đúng, kết quả sẽ là “Right Prediction”, nếu sai, kết quả là “Wrong prediction”.

Ví dụ 1. Dự đoán đúng bệnh Apple__Black_rot.



The screenshot shows the same "Disease Identification" window, but now it contains data. The "Image path:" field shows the path "/home/tuannv/ws/Do-An/". The "Open" button is still present. The "True label:" field contains "Apple__Black_rot". The "Predited label:" field also contains "Apple__Black_rot". The "Confident:" field shows "1.0". The "Result:" field shows "Right Prediction". Below the text fields is a photograph of a green apple leaf with several brown, necrotic spots, characteristic of Black Rot disease.

Hình 33. Dự đoán đúng bệnh Apple__Back_rot

Ta có:

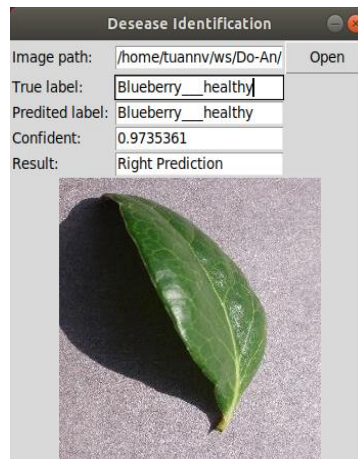
True label: Apple___Black_rot.

Predicted label: Apple___Black_rot.

Confident của model là 1.0 . Tức là model tin tưởng 100% vào dự đoán của nó.

So sánh “True label” và “Predicted Lable”, phần mềm thấy hai kết quả giống nhau, nên kết luận model đã chạy đúng. Do đó Result là “Right Prediction”.

Ví dụ 2. Dự đoán đúng bệnh Blueberry__healthy.



Hình 34. Dự đoán đúng bệnh Blueberry__healthy

Ta có:

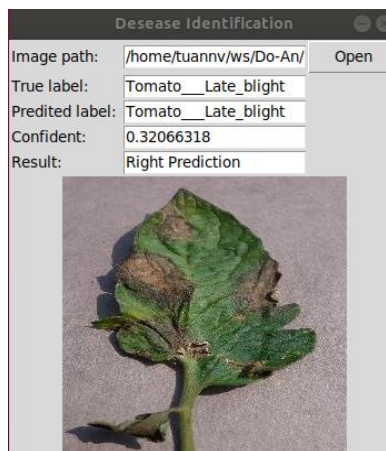
True label: Blueberry___healthy.

Predicted label: Blueberry___healthy.

Confident của model là 0.9735361. Tức là model tin tưởng 97.35% vào dự đoán của nó.

So sánh “True label” và “Predicted Lable”, phần mềm thấy hai kết quả giống nhau, nên kết luận model đã chạy đúng. Do đó Result là “Right Prediction”.

Ví dụ 3. Dự đoán đúng bệnh Tomato__late_blight



Hình 35. Dự đoán đúng bệnh Tomato__late_blight

Ta có:

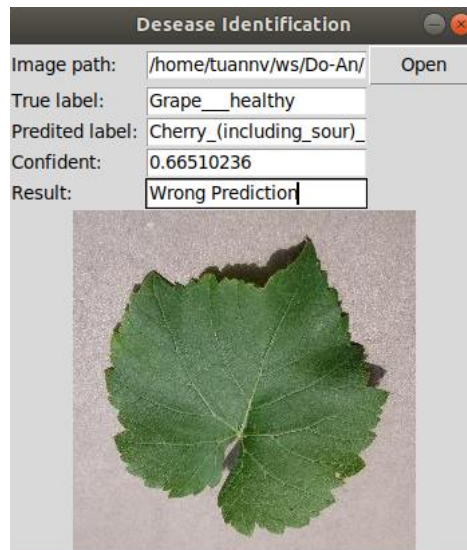
True label: Tomato___Late_blight.

Predicted label: Tomato___Late_blight.

Confident của model là 0.32066318. Tức là model tin tưởng 32.07% vào dự đoán của nó.

So sánh “True label” và “Predicted Lable”, phần mềm thấy hai kết quả giống nhau, nên kết luận model đã chạy đúng. Do đó Result là “Right Prediction”.

Ví dụ 4. Dự đoán sai bệnh Grape___healthy.



Hình 36. Dự đoán sai bệnh Grape___healthy.

Ta có:

True label: Grape___healthy.

Predicted label: Cherry_(including_sour)___healthy.

Confident của model là 0.66510236. Tức là model tin tưởng 66.51% vào dự đoán của nó.

So sánh “True label” và “Predicted Lable”, phần mềm thấy hai kết quả khác nhau, nên kết luận model đã chạy sai. Do đó Result là “Wrong Prediction”.

Chương 5: Đánh giá, kết luận và hướng phát triển

6.1. Đánh giá

Trong mô hình được xây dựng trước, xe đã có khả năng bám theo làn đường khá tốt, có khả năng xử lý đoạn đường thẳng và khúc cua với tốc độ tối đa là 0.2m/s (duty cycles động cơ là 20%) và sai số trong khoảng 10cm.

Sử dụng mô hình mạng InceptionV3, model có khả năng nhận dạng loại lá cây và bệnh của lá cây với tỉ lệ chính xác 90%.

Chương trình nhận dạng chạy trên máy tính Core i7-7900K và GPU GTX1060 cho tốc độ xử lý cao.

6.2. Kết luận

Một số kết quả mà đề tài đạt được:

- Tìm hiểu về một số cấu trúc của một mạng neuron.
- Tìm hiểu về cấu trúc của CNN.
- Tìm hiểu về cấu hình mạng InceptionV3.
- Sử dụng TensorFlow Framework để implement mạng thần kinh.
- Sử dụng thư viện Tkinter của Python để lập trình giao diện.

Đồng thời, đề tài còn một số công việc chưa thể hoàn thành tốt:

- Độ chính xác của model trên tập validation chưa cao 90%.
- Trong một số trường hợp, một dự đoán là sai cũng có thể có một confident cao. Điều này ảnh hưởng khá nhiều đến khả năng ứng dụng vào thực tế.

6.3. Hướng phát triển

Nghiên cứu hiện tại là nền tảng để có thể thực hiện những nghiên cứu sâu hơn, xa hơn và hoàn thiện hơn. Những nghiên cứu này có thể trở thành tiền đề để có thể ứng dụng lên trong nông nghiệp.

Hiện tại, để có được kết quả dự đoán, ảnh đầu vào phải là bức ảnh của một lá cây duy nhất, và ở khoảng cách đủ gần. Điều này gây bất tiện trong quá trình sử dụng phần mềm. Do đó, đề tài định hướng phát triển sao cho có thể nhận dạng được bệnh cây từ những bức ảnh gồm nhiều lá cây của một cây.

Tài liệu tham khảo

- [1] <https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>
- [2] <https://khanh-personal.gitbook.io/ml-book-vn/chapter1/ham-mat-mat>
- [3] <https://machinelearningcoban.com/2017/02/24/mlp/>
- [3] <https://iamaaditya.github.io/2016/03/one-by-one-convolution/>
- [4] <https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c>
- [4] <https://www.tensorflow.org/learn>