Mục lục

## Danh mục từ viết tắt

## Danh mục các hình ảnh

# Danh mục các bảng

# Chương I: Giới thiệu về bài toán

## 1. Ngữ cảnh bài toán

Bài toán phát hiện vật thể có rất nhiều ứng dụng trong các hệ thống thông minh hiện nay và là lĩnh vực rất quan trọng trong phát triển hệ thống học sâu đặc biệt là thị giác máy tính (computer vision). Thị giác máy tính là một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh, phát hiện các đối tượng,... Thị giác máy tính được chia ra thành các nhiệm vụ khác nhau: Phân loại hình ảnh (Image Classification), Định vị vật thể (Object Localization), Phát hiện vật thể (Object Detection) và Phân vùng hình ảnh (Image Segmentation).

Thị giác máy tính đã được sử dụng rộng rãi để phát hiện khuôn mặt, phát hiện biển số xe, phát hiện chữ viết tay, phát hiện người đeo khẩu trang,... Hiện nay, có rất nhiều mô hình kiến trúc sinh ra (R-CNN, Faster R-CNN, Mask R-CNN, YOLO, SSD,...) được huấn luyện trên các tập dữ liệu lớn như ImageNet, MS Coco, CIFAR, LableMe,... mang lại độ chính xác cao cũng như tốc độ dự đoán nhanh cho các bài toán nhận dạng, phân vùng vật thể.

Bài toán muốn quét ảnh tài liệu từ máy ảnh để trích xuất trang tài liệu (hóa đơn, ảnh,...) xuất hiện trong ảnh.

Trong nội dung đồ án, em thực hiện nghiên cứu và huấn luyện mô hình học sâu để phát hiện và định vị tài liệu xuất hiện trong ảnh thu được từ thiết bị di động tại thời gian thực.

## 2. Tổng quan về bài toán phát hiện Tài liệu trong ảnh thực tế thu được từ máy ảnh

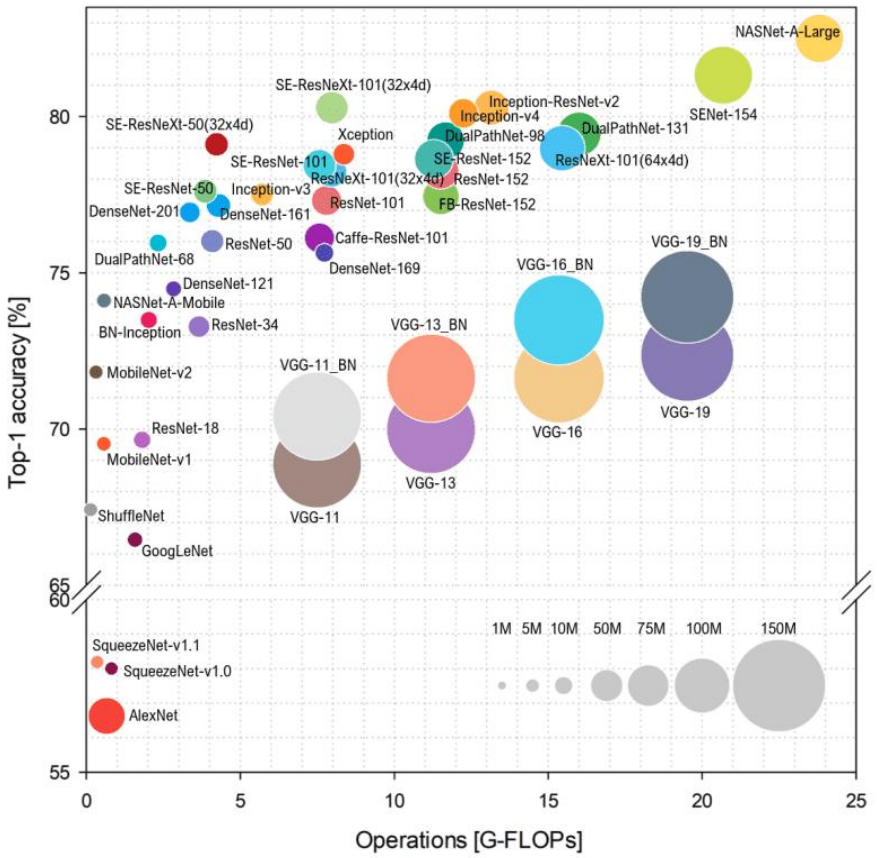
**Phát hiện và làm phẳng** ảnh là quá trình xử lý ảnh để phát hiện và làm phẳng vật thể xuất hiện trong ảnh. Trong đồ án, em thực hiện nghiên cứu quá trình trên với vật thể là tài liệu sử dụng phương pháp học sâu.

**Tài liệu** bao gồm sách, giấy tờ, hóa đơn giấy,...

**Làm phẳng** là quá trình chuyển đổi phối cảnh (góc nhìn) để có thể thu được thông tin chi tiết tốt hơn. Quá trình chuyển đổi phối cảnh (góc nhìn) không làm mất nội dung của ảnh.

Trong bài toán phát hiện tài liệu trong ảnh, ảnh có thể được phát hiện ở nhiều góc nhìn khác nhau nên quá trình làm phẳng ảnh sẽ đưa ảnh tài liệu thu được về góc nhìn trực diện để thông tin được hiển thị chi tiết hơn.

Trong nội dung đồ án, em sẽ nghiên cứu và phát triển mô-đun phát hiện và làm phẳng ảnh cho thiết bị di động. Thiết bị di động bị giới hạn về mặt phần cứng nên việc phát hiện và làm phẳng ảnh tại thời gian thực yêu cầu mô-đun có tốc độ xử lý nhanh và kích thước nhỏ. Năm 2017, Google cho ra mắt mạng tích chập MobileNet đáp ứng được nhu cầu về tốc độ cũng như kích thước nhẹ cho các mô hình học sâu dành cho thiết bị di động.




Năm 2018, Google tiếp tục ra mắt MobileNet v2 với những cải thiện về độ chính xác cũng như về kích thước và ra mắt MobileNet v3 ở 2019 với cải thiện cho các bài toán nhận dạng và phân vùng. Ngoài ra em sẽ huấn luyện mô hình với các mạng tích chập khác nhau để tìm được kết quả tốt nhất.

Trong bài toán này, thách thức lớn nhất chính là độ đa dạng của đầu vào. Ảnh đầu vào bao gồm sự đa dạng về góc nhìn, đa dạng về loại tài liệu đặc biệt là đa dạng của điều kiện môi trường làm ảnh hưởng đến độ chính xác của kết quả.

## 3. Hướng tiếp cận

### 3. 1. Hướng tiếp cận phát hiện Tài liệu trong ảnh thực tế thu được từ máy ảnh qua phương pháp xử lý ảnh

Phương pháp xử lý ảnh sử dụng các bộ lọc, kỹ thuật xử lý ảnh cơ bản để phát hiện ra các góc của tài liệu. Tư tưởng của phương pháp là phát hiện các cạnh của tài liệu, từ đó tìm giao điểm của các cạnh để dự đoán góc của tài liệu. Phương pháp bao gồm các bước:

1. Đưa ảnh về không gian màu phù hợp để tăng độ tương phản, làm nổi bật các cạnh của vật thể

2. Sử dụng các bộ lọc (Canny, Sobel,…) để phát hiện cạnh kết hợp với các bộ lọc nhiễu để tăng chất lượng của đầu ra
3. Sử dụng kỹ htuaajt HoughLine để tìm kiếm các điểm liên tiếp cùng nằm trên cùng một đường thẳng (phát hiện đường thẳng)
4. Tìm kiếm giao điểm của các đường thẳng được tìm kiếm ở bước trên (góc sẽ là giao điểm của 2 cạnh tài liệu phát hiện được)
5. Do các yếu tố gây nhiễu sẽ khiến quá trình tìm kiếm đường thẳng tìm được các đường thẳng không phải là cạnh của tài liệu nên phải thực hiện đánh giá 4 giao điểm trong tập các giao điểm tìm được có khả năng là tài liệu.

Phương pháp xử lý ảnh có ưu điểm tốc độ xử lý nhanh, kết quả thu được chính xác nhưng nhược điểm là khi ảnh thu được có yếu tố gây nhiễu khiến việc phát hiện và đánh giá xảy ra sai số thì kết quả thu được sẽ không chính xác. Kết quả của quá trình phụ thuộc hoàn toàn vào thuật toán đánh giá xem các điểm có là góc của tài liệu không.

Khi dự đoán tài liệu có độ tương phản cao so với nền, ít yếu tố gây nhiễu thì phương pháp đạt độ chính xác rất cao (sai số rất nhỏ). Ngược lại, khi tài liệu xuất hiện trong ảnh không đầy đủ, nhiều yếu tố gây nhiễu sẽ làm việc dự đoán bị sai.

### 3. 2. Hướng tiếp cận phát hiện Tài liệu trong ảnh thực tế thu được từ máy ảnh qua phương pháp học sâu

Để khắc phục nhược điểm của phương pháp xử lý ảnh nên trên thì em sử dụng học sâu để cải thiện việc dự đoán vị trí của tài liệu khi ảnh thu được có yếu tố gây nhiễu. Huấn luyện mô hình trên tập dữ liệu đa dạng về nội dung mang lại hiệu quả khi dự đoán với đầu vào là ảnh thực tế thu được từ máy ảnh.

Piệc dự đoán của mô hình học sâu mang lại hiệu quả ở đa dạng độ phức tạp của dữ liệu đầu vào, mô hình dự đoán được chính xác vị trí của tài liệu xuất hiện trong ảnh nhưng độ chính xác cụ thể góc tài liệu thì mô hình không có độ chính xác cao như phương pháp xử lý ảnh trên.

Phương pháp này vẫn không khắc phục được nhược điểm dự đoán không chính xác khi tài liệu xuất hiện trong ảnh không đầy đủ.

Để cải thiện độ chính xác của mô hình khi thực hiện dự đoán góc của tài liệu, trong nội dung đồ án em nghiên cứu và thực hiện quá trình 2 bước:

1. Sử dụng mô hình học sâu để định vị vị trí của tài liệu xuất hiện trong ảnh
2. Xác định chính xác vị trí góc tài liệu:
    a) Sử dụng mô hình học sâu thứ 2 để dự đoán
    b) Sử dụng thuật toán xử lý ảnh để dự đoán

## 4. Mục tiêu đồ án

## Chương II: Giới thiệu về mạng nơ-ron và mạng nơ-ron tích chập

### 1. Mạng nơ-ron:

#### 1. 1. Định nghĩa:

Neural Network đọc tiếng việt là Mạng nơ-ron nhân tạo, đây là một chuỗi những thuật toán được đưa ra để tìm kiếm các mối quan hệ cơ bản trong tập hợp các dữ liệu. Thông qua việc bắt bước cách thức hoạt động từ não bộ con người. Nói cách khác, mạng nơ ron nhân tạo được xem là hệ thống của các tế bào thần kinh nhân tạo. Đây thường có thể là hữu cơ hoặc nhân tạo về bản chất.

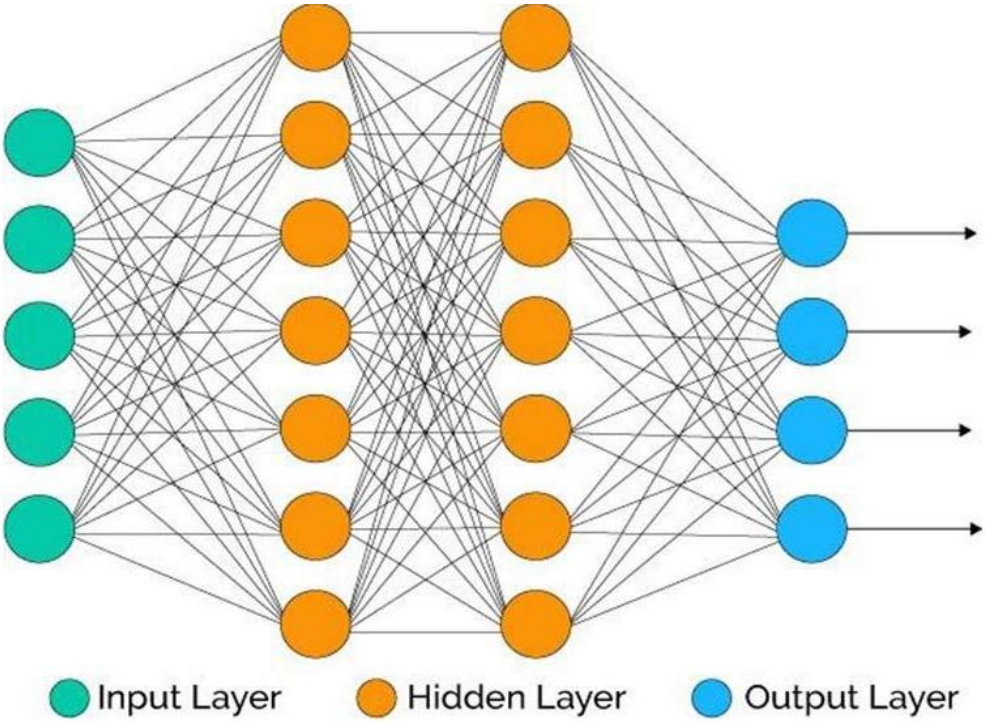
Neural Network có khả năng thích ứng được với mọi thay đổi từ đầu vào. Do vậy, nó có thể đưa ra được mọi kết quả một cách tốt nhất có thể mà bạn không cần phải thiết kế lại những tiêu chí đầu ra.
Khái niệm này có nguồn gốc từ trí tuệ nhân tạo, đang nhanh chóng trở nên phổ biến hơn trong sự phát triển của những hệ thống giao dịch điện tử.

#### 1. 2. Kiến trúc mạng nơ-ron:

Mạng Neural Network là sự kết hợp của những tầng perceptron hay còn gọi là perceptron đa tầng. Và mỗi một mạng Neural Network thường bao gồm 3 kiểu tầng là:

- Tầng input layer (tầng vào): Tầng này nằm bên trái cùng của mạng, thể hiện cho các đầu vào của mạng.
- Tầng output layer (tầng ra): Là tầng bên phải cùng và nó thể hiện cho những đầu ra của mạng.
- Tầng hidden layer (tầng ẩn): Tầng này nằm giữa tầng vào và tầng ra nó thể hiện cho quá trình suy luận logic của mạng.

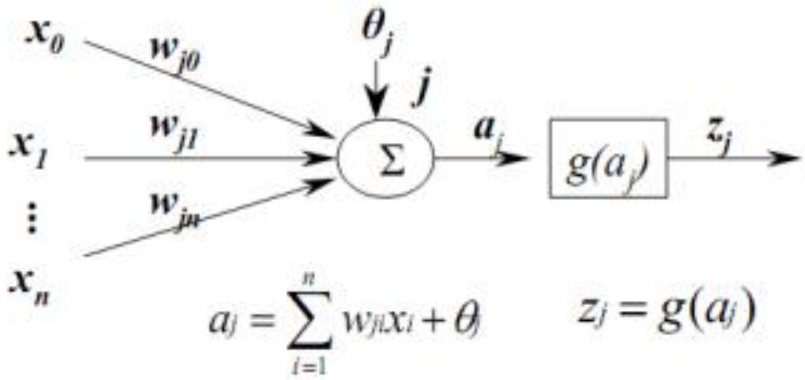
Lưu ý: Mỗi một Neural Network chỉ có duy nhất một tầng vào và 1 tầng ra nhưng lại có rất nhiều tầng ẩn.




## 1. 3. Các thành phần của mạng nơ-ron:

### 1. 3. 1. Đơn vị xử lý (nơ-ron):

Còn được gọi là một nơron hay một nút (node), thực hiện một công việc rất đơn giản: nó nhận tín hiệu vào từ các đơn vị phía trước hay một nguồn bên ngoài và sử dụng chúng để tính tín hiệu ra sẽ được lan truyền sang các đơn vị khác.




***Trong đó:***

***xi*** : các đầu vào

***wji*** : các trọng số tương ứng với các đầu vào

***θj*** : độ lệch (bias)

***aj*** : đầu vào mạng (net-input)

***zj*** : đầu ra của nơron

***g(x):*** hàm chuyển (hàm kích hoạt).

Trong một mạng nơron có ba kiểu đơn vị:

- Các đơn vị đầu vào (Input units), nhận tín hiệu từ bên ngoài.
- Các đơn vị đầu ra (Output units), gửi dữ liệu ra bên ngoài.
- Các đơn vị ẩn (Hidden units), tín hiệu vào (input) và ra (output) của nó nằm trong mạng.

Mỗi đơn vị j có thể có một hoặc nhiều đầu vào: x0, x1, x2, … xn, nhưng chỉ có một đầu ra zj. Một đầu vào tới một đơn vị có thể là dữ liệu từ bên ngoài mạng, hoặc đầu ra của một đơn vị khác, hoặc là đầu ra của chính nó.

### 1. 3. 2. Hàm kết hợp:

Mỗi một đơn vị trong một mạng kết hợp các giá trị đưa vào nó thông qua các liên kết với các đơn vị khác, sinh ra một giá trị gọi là net input. Hàm thực hiện nhiệm vụ này gọi là hàm kết hợp (combination function), được định nghĩa bởi một luật lan truyền cụ thể. Trong phần lớn các mạng nơron, chúng ta giả sử rằng mỗi một đơn vị cung cấp một bộ cộng như là đầu vào cho đơn vị mà nó có liên kết. Tổng đầu vào đơn vị j đơn giản chỉ là tổng trọng số của các đầu ra riêng lẻ từ các đơn vị kết nối cộng thêm ngưỡng hay độ lệch (bias) θj :

$$a_j = \sum_{i=1}^{n} w_{ji} x_i + \theta_j$$

Trường hợp wji > 0, nơron được coi là đang ở trong trạng thái kích thích. Tương tự, nếu như wji < 0, nơron ở trạng thái kiềm chế. Chúng ta gọi các đơn vị với luật lan truyền như trên là các sigma units.

Trong một vài trường hợp người ta cũng có thể sử dụng các luật lan truyền phức tạp hơn. Một trong số đó là luật sigma-pi, có dạng như sau:

$$a_j = \sum_{i=1}^{n} w_{ji} \prod_{k=1}^{m} x_{ik} + \theta_j$$

Rất nhiều hàm kết hợp sử dụng một "độ lệch" hay "ngưỡng" để tính net input tới đơn vị. Đối với một đơn vị đầu ra tuyến tính, thông thường, θj được chọn là hằng số và trong bài toán xấp xỉ đa thức θj = 1.

### 1. 3. 3. Hàm kích hoạt:

#### 1. 3. 3. 1. Định nghĩa:

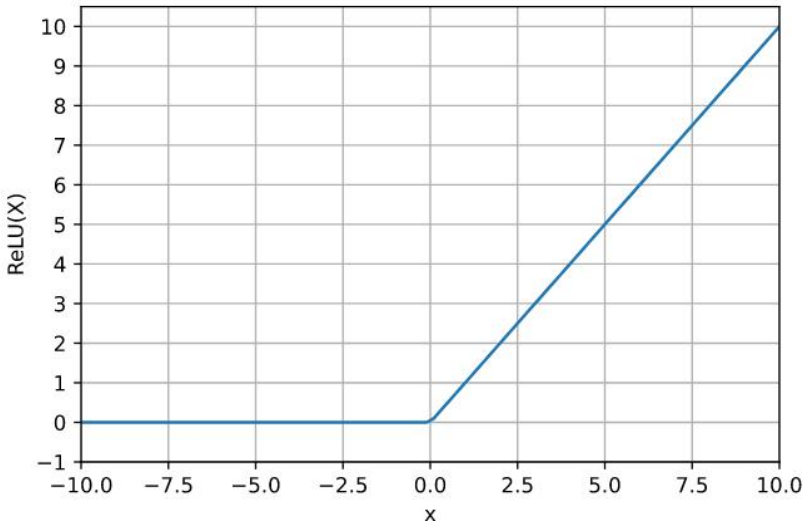
Phần lớn các đơn vị trong mạng nơron chuyển net input bằng cách sử dụng một hàm vô hướng (scalar-to-scalar function) gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị (unit's activation). Loại trừ khả năng đơn vị đó thuộc lớp ra, giá trị kích hoạt được đưa vào một hay nhiều đơn vị khác. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp (squashing)

Các hàm chuyển của các đơn vị ẩn (hidden units) là cần thiết để biểu diễn sự phi tuyến vào trong mạng. Lý do là hợp thành của các hàm đồng nhất là một hàm đồng nhất. Mặc dù vậy nhưng nó mang tính chất phi tuyến (nghĩa là, khả năng biểu diễn các hàm phi tuyến) làm cho các mạng nhiều tầng có khả năng rất tốt trong biểu diễn các ánh xạ phi tuyến. Tuy nhiên, đối với luật học lan truyền ngược, hàm phải khả vi (differentiable) và sẽ có ích nếu như hàm được gắn trong một khoảng nào đó. Do vậy, hàm sigmoid là lựa chọn thông dụng nhất.

#### 1. 3. 3. 2. Một số hàm kích hoạt:

- **ReLu:** f(x) = max(0, x)

Hàm ReLU đang được sử dụng khá nhiều trong những năm gần đây khi huấn luyện các mạng neuron. ReLU đơn giản lọc các giá trị < 0.

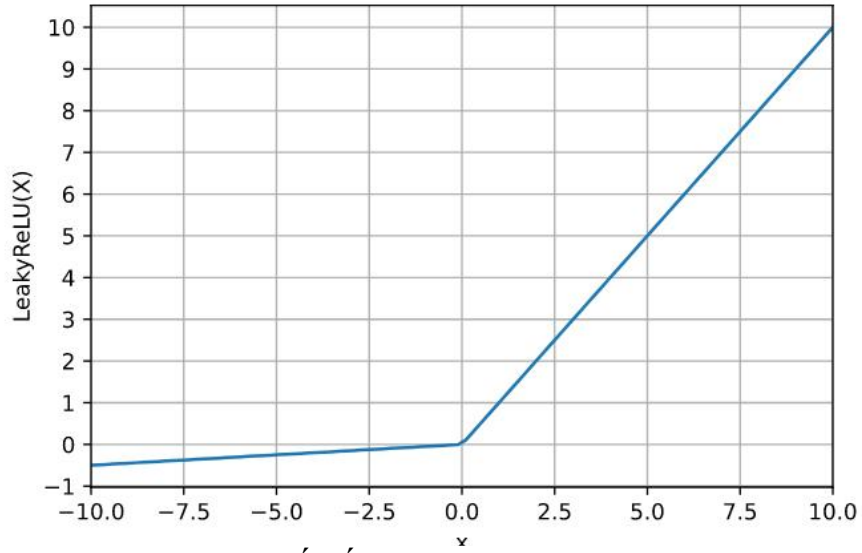


Tuy nhiên, các node có giá trị nhỏ hơn 0 khi qua ReLU sẽ thành giá trị 0 và không có ý nghĩa với các bước tiếp theo và hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent. (Dying ReLU)

- **Leaky ReLU:**

  f(x) = x khi x > 0

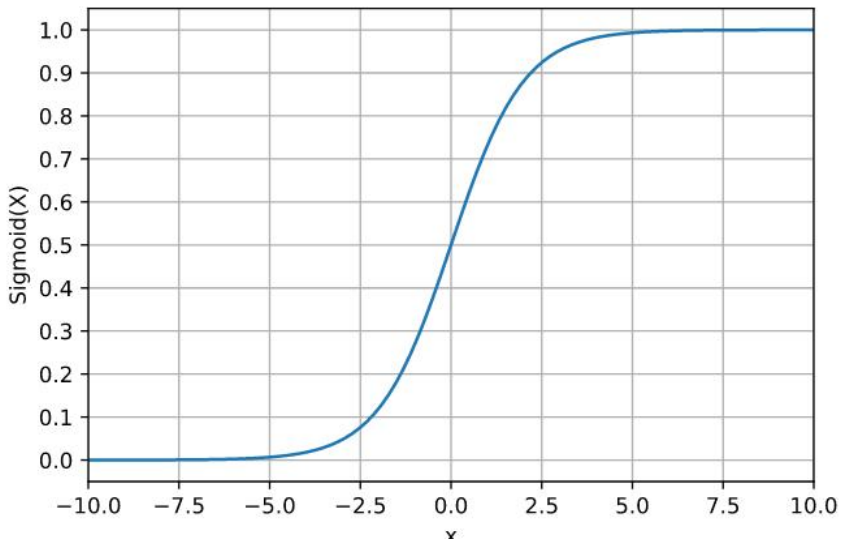
  f(x) = αx khi x < 0 và α là hằng số rất nhỏ 0.005




Leaky ReLU là một cố gắng trong việc loại bỏ "dying ReLU". Thay vì trả về giá trị 0 với các đầu vào <0 thì Leaky ReLU tạo ra một đường xiên có độ dốc nhỏ (xem đồ thị).

- **Sigmoid:**

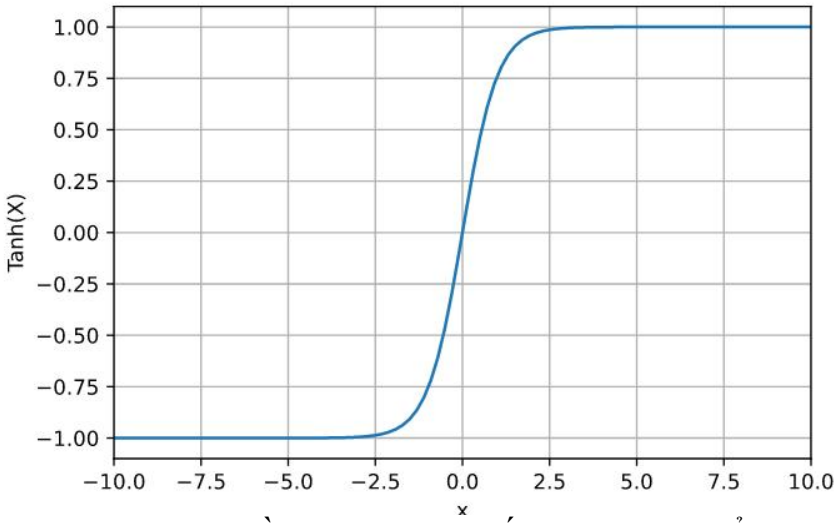
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Hàm Sigmoid nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0;1) (xem đồ thị phía trên). Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1.

- **Tanh:**

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

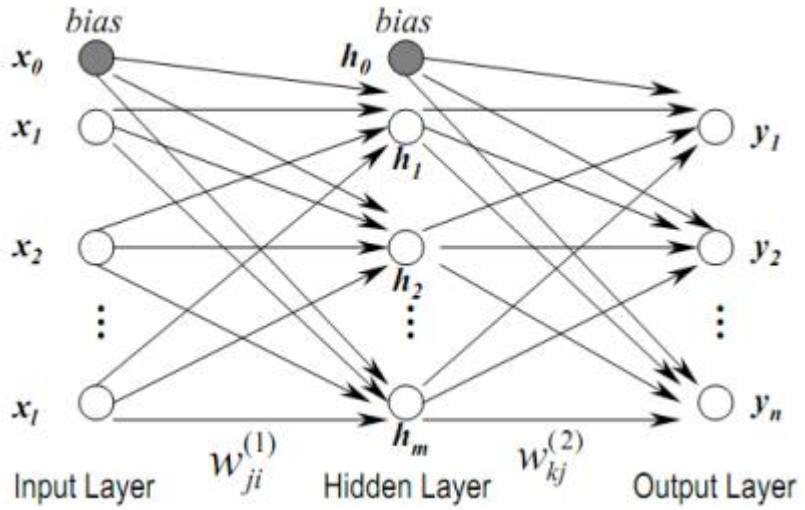



Hàm tanh nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (-1; 1). Cũng như Sigmoid, hàm Tanh bị bão hoà ở 2 đầu (gradient thay đổi rất ít ở 2 đầu)

## 1. 4. Các hình trạng của mạng:

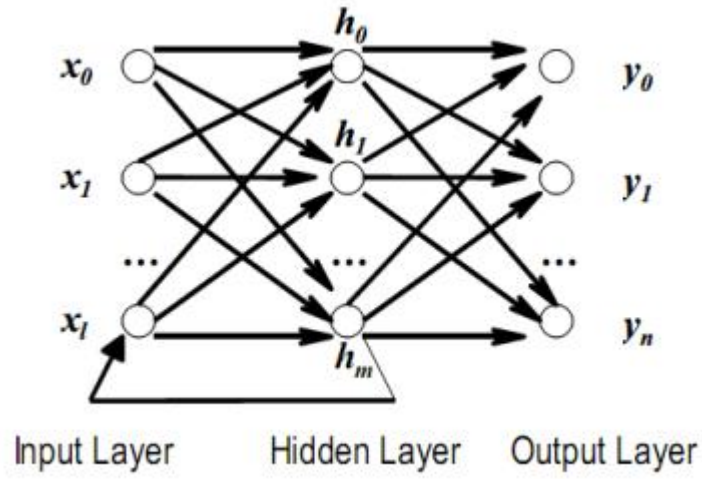
### 1. 4. 1. Mạng truyền thẳng:

Dòng dữ liệu từ đơn vị đầu vào đến đơn vị đầu ra chỉ được truyền thẳng. Việc xử lý dữ liệu có thể mở rộng ra nhiều lớp, nhưng không có các liên kết phản hồi. Nghĩa là, các liên kết mở rộng từ các đơn vị đầu ra tới các đơn vị đầu vào trong cùng một lớp hay các lớp trước đó là không cho phép




### 1. 4. 2. Mạng hồi quy:

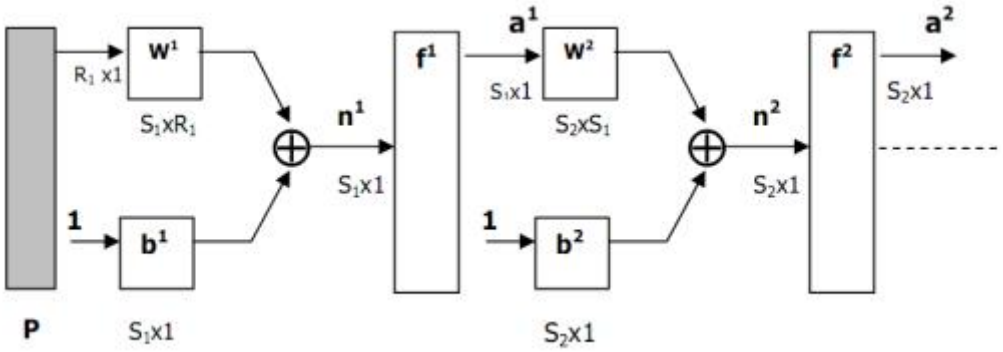
Có chứa các liên kết ngược. Khác với mạng truyền thẳng, các thuộc tính động của mạng mới quan trọng. Trong một số trường hợp, các giá trị kích hoạt của các đơn vị trải qua quá trình nới lỏng (tăng giảm số đơn vị và thay đổi các liên kết) cho đến khi mạng đạt đến một trạng thái ổn định và các giá trị kích hoạt không thay đổi nữa. Trong các ứng dụng khác mà cách chạy động tạo thành đầu ra của mạng thì những sự thay đổi các giá trị kích hoạt là đáng quan tâm.




## 1. 5. Mạng truyền thẳng và lan truyền ngược:

Một mạng truyền thẳng nhiều lớp bao gồm một lớp vào, một lớp ra và một hoặc nhiều lớp ẩn. Các nơron đầu vào thực chất không phải các nơron theo đúng nghĩa, bởi lẽ chúng không thực hiện bất kỳ một tính toán nào trên dữ liệu vào, đơn giản nó chỉ tiếp nhận các dữ liệu

vào và chuyển cho các lớp kế tiếp. Các nơron ở lớp ẩn và lớp ra mới thực sự thực hiện các tính toán, kết quả được định dạng bởi hàm đầu ra (hàm chuyển). Cụm từ "truyền thẳng" (feed forward) (không phải là trái nghĩa của lan truyền ngược) liên quan đến một thực tế là tất cả các nơron chỉ có thể được kết nối với nhau theo một hướng: tới một hay nhiều các nơron khác trong lớp kế tiếp (loại trừ các nơron ở lớp ra)




**Trong đó:**

***P***: Vector đầu vào (vector cột)

***Wi*** : Ma trận trọng số của các nơron lớp thứ i. (Si xRi : S hàng (nơron) - R cột (số đầu vào))

***bi*** : Vector độ lệch (bias) của lớp thứ i (Si x1: cho S nơron)

***ni*** : net input (Si x1)

***fi*** : Hàm chuyển (hàm kích hoạt)

***ai*** : net output (Si x1)

⊕: Hàm tổng thông thường.

Mỗi liên kết gắn với một trọng số, trọng số này được thêm vào trong quá trình tín hiệu đi qua liên kết đó. Các trọng số có thể dương, thể hiện trạng thái kích thích, hay âm, thể hiện trạng thái kiềm chế. Mỗi nơron tính toán mức kích hoạt của chúng bằng cách cộng tổng các đầu vào và đưa ra hàm chuyển. Một khi đầu ra của tất cả các nơron trong một lớp mạng cụ thể đã thực hiện xong tính toán thì lớp kế tiếp có thể bắt đầu thực hiện tính toán của mình bởi vì đầu ra của lớp hiện tại tạo ra đầu vào của lớp kế tiếp. Khi tất cả các nơron đã thực hiện tính toán thì kết quả được trả lại bởi các nơron đầu ra. Tuy nhiên, có thể là chưa đúng yêu cầu, khi đó một thuật toán huấn luyện cần được áp dụng để điều chỉnh các tham số của mạng.

**Thuật toán lan truyền ngược:**

Cần có một sự phân biệt giữa kiến trúc của một mạng và thuật toán học của nó, các mô tả trong các mục trên mục đích là nhằm làm

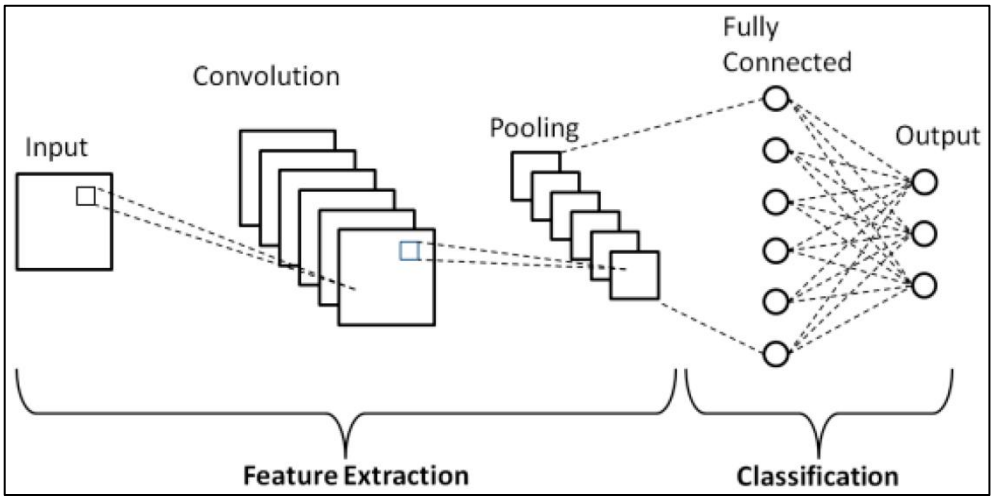
rõ các yếu tố về kiến trúc của mạng và cách mà mạng tính toán các đầu ra từ tập các đầu vào. Sau đây là mô tả của thuật toán học sử dụng để điều chỉnh hiệu năng của mạng sao cho mạng có khả năng sinh ra được các kết quả mong muốn.

Về cơ bản có hai dạng thuật toán để luyện mạng: học có thầy và học không có thầy. Các mạng nơron truyền thẳng nhiều lớp được luyện bằng phương pháp học có thầy. Phương pháp này căn bản dựa trên việc yêu cầu mạng thực hiện chức năng của nó và sau đó trả lại kết quả, kết hợp kết quả này với các đầu ra mong muốn để điều chỉnh các tham số của mạng, nghĩa là mạng sẽ học thông qua những sai sót của nó.

Thuật toán lan truyền ngược là dạng tổng quát của thuật toán trung bình bình phương tối thiểu (Least Means Square-LMS). Thuật toán này thuộc dạng thuật toán xấp xỉ để tìm các điểm mà tại đó hiệu năng của mạng là tối ưu. Chỉ số tối ưu (performance index) thường được xác định bởi một hàm số của ma trận trọng số và các đầu vào nào đó mà trong quá trình tìm hiểu bài toán đặt ra.

## 2. Mạng nơ-ron tích chập:

- Neural network không xử lý tốt với đầu vào là số lượng ảnh lớn
- Mô hình CNN xử lý đầu vào nhằm giảm thiểu số lượng thông tin phải xử lý trong neural network.
- Kiến trúc của CNN:




- **Convolution (tích chập):** xử lý ảnh đầu vào với bộ lọc (kernel). Ảnh sau khi xử lý bao gồm các Feature map.

- **Pooling (tổng hợp)**: đơn giản hóa (giảm phân giải của ảnh nhưng vẫn giữ lại được đặc trưng tốt nhất của ảnh). Có 3 loại: max, average, min
- **Fully-Connected (Neural network):** tất cả các nút của lớp này sẽ kết nối với tất cả các nút của lớp trước đó

## 3. Các khái niệm:

### 3. 1. Hàm mất mát:

#### 3. 1. 1. Định nghĩa:

Hàm tổn thất là số liệu giúp mạng hiểu liệu nó đang học đúng hướng. Để đơn giản hóa về hàm mất mát, đó như là điểm kiểm tra trong kỳ kiểm tra để đánh giá độ chính xác của việc học.

Giả sử bạn đang phát triển một mô hình để dự đoán liệu một học sinh sẽ đậu hay rớt và cơ hội đậu hay rớt được xác định bởi xác suất. Vì vậy, 1 sẽ chỉ ra rằng anh ấy sẽ vượt qua với 100% chắc chắn và 0 sẽ chỉ ra rằng anh ta chắc chắn sẽ thất bại.

Mô hình học hỏi từ dữ liệu và dự đoán điểm là 0,87 cho sinh viên để vượt qua. Vì vậy, độ mất mát thực tế ở đây sẽ là 1,00 - 0,87 = 0,13. Nếu nó lặp lại bài tập với một số cập nhật thông số để cải thiện và bây giờ đạt được mức độ mất mát là 0,40, nó sẽ hiểu rằng những thay đổi nó có được thực hiện không giúp mạng tìm hiểu một cách thích hợp. Ngoài ra, mức độ mất mát mới 0,05 sẽ chỉ ra rằng các bản cập nhật hoặc thay đổi từ việc học đang đi đúng hướng.

#### 3. 1. 2. Một số hàm mất mát:

- *Mean Square Error:* Trung bình của bình phương độ chênh lệch giữa giá trị dự đoán và giá trị thực tế. Bình phương độ chênh lệch sẽ dễ dàng dẫn đến sự thay đổi của mô hình hơn với sự khác biệt cao hơn vì giả sử độ chênh lệch là 3 thì hàm mất mát là 9 nhưng nếu độ chênh lệch là 9 thì hàm mất mát sẽ là 81.

$$\sum_{n=1}^{k} \frac{(Actual - Predicted)^2}{k}$$

- *Mean Absolute Error:* Trung bình của trị tuyệt đối độ chênh lệch giữa giá trị thực tế và giá trị dự đoán

$$\sum_{n=1}^{k} |Actual - Predicted|$$

**Với bài toán phân loại:**

- *Binary cross-entropy:* Giá trị mất mát khi dự đoán phân loại có đầu ra chỉ (Đúng/Sai) hoặc (Có/Không) (giá trị nhị phân)

$$\text{Loss} = -\left[\, y * \log(p) + (1-y) * \log(1-p) \,\right]$$

- *Categorical cross-entropy*: Giá trị mất mát khi dự đoán phân loại có đầu ra không phải dạng nhị phân. Ví dụ: (Loại 1, loại 2, loại 3, …) hoặc (Có, Không, Có thể, ...)

$$\text{Loss} = -\sum_{i}^{n} y_i' log_2\, y_i$$

## 3. 2. Optimizers:

### 3. 2. 1. Định nghĩa:

Về cơ bản, thuật toán tối ưu là cơ sở để xây dựng mô hình neural network với mục đích "học " được các features ( hay pattern) của dữ liệu đầu vào, từ đó có thể tìm 1 cặp weights và bias phù hợp để tối ưu hóa model. Nhưng vấn đề là "học" như thế nào? Cụ thể là weights và bias được tìm như thế nào! Đâu phải chỉ cần random (weights, bias) 1 số lần hữu hạn và hy vọng ở 1 bước nào đó ta có thể tìm được lời giải.

Tóm lại, một mạng có trọng số ngẫu nhiên và cấu trúc là điểm khởi đầu cho một mô hình. Mô hình dự đoán tại thời điểm này, nhưng nó hầu như luôn không có giá trị. Các mạng lấy một mẫu đào tạo và sử dụng các giá trị của nó làm đầu vào cho tế bào thần kinh trong lớp đầu tiên, sau đó tạo ra một đầu ra với hàm kích hoạt. Đầu ra bây giờ trở thành đầu vào cho lớp tiếp theo, và như thế. Đầu ra của lớp cuối cùng sẽ là dự đoán cho mẫu đào tạo. Đây là nơi mà hàm mất mát được sử dụng.

Bước tiếp theo cho mô hình là để giảm tổn thất. Làm thế nào nó biết những bước hoặc cập nhật nó nên thực hiện trên giảm bớt sự mất mát? Hàm tối ưu hóa giúp nó hiểu bước này. Hàm tối ưu hóa là một phép toán học thuật toán sử dụng đạo hàm, đạo hàm

riêng và quy tắc chuỗi trong phép tính để hiểu mức độ thay đổi mà mạng sẽ thấy trong mất chức năng do tạo ra một thay đổi nhỏ về trọng lượng của các tế bào thần kinh

**Tóm lại:**

- Là phần quan trọng trong việc huấn luyện model.
- Mỗi khi huấn luyện, weight và bias được lấy bất kì. Hàm loss cho biết việc lấy giá trị weight và bias đó là tốt hay là xấu. Và bước tiếp theo cho việc huấn luyện mô hình là giảm thiểu giá trị loss.
- Hàm tối ưu hóa sẽ thấy được sự thay đổi của hàm loss khi mà thay đổi nhỏ weight
- **Batch**: là một tập các mẫu trong dataset. **Iterations**: số lượng batch để hoàn thành 1 epoch. **Epoch**: là khi thực hiện hết tất cả các batch

### 3. 2. 2. Một số hàm tối ưu:

#### 3. 2. 2. 1. Gradient descent:

Tư tưởng của thuật toán là thay đổi weight của mạng qua mỗi epoch theo công thức được đơn giản hóa:

*weight = weight - learning rate * Loss*

Trong đó:

***learning rate***: tham số được định nghĩa ở kiến trúc của mạng

#### 3. 2. 2. 2. Stochastic gradient descent:

Thay vì cập nhật weight sau mỗi epoch thì thuật toán SGD cập nhật weight sau mỗi batch.Điều này khiến cho weight được cập nhật quá thường xuyên và làm cho hàm loss trở nên bất ổn định nhưng việc hội tụ lại đạt tốc độ nhanh hơn. SGD thường được sử dụng cho những mô hình yêu cầu sự thay đổi liên tục, tức là online learning.

#### 3. 2. 2. 3. Momentum (động lượng):

Nhược điểm của *Gradient descent* là việc hội tụ hay tìm ra nghiệm phụ thuộc vào điểm xuất phát cũng như learning rate. Nên việc lựa chọn điểm xuất phát và learning rate không tối ưu sẽ khiến mô hình không hội tụ hoặc rơi vào nghiệm cục bộ không mong muốn.

Để khắc phục nhược điểm của *Gradient Descent* thì mô hình sử dụng *Gradient Descent với Momentum.* Momentum được định

nghĩa như vận tốc để giúp Gradient Descent vượt qua nghiệm cục bộ. Việc sử dụng *Gradient Descent với Momentum* tăng độ chính xác của mô hình nhưng sẽ tốn thời gian hơn trong huấn luyện

#### 3. 2. 2. 4. Adam:

Adam, viết tắt của Adaptive Moment Estimation, cho đến nay là rình tối ưu hóa phổ biến và được sử dụng rộng rãi trong Deep Learning. Trong hầu hết các trường hợp, ta có thể mù quáng chọn trình tối ưu hóa Adam và quên các lựa chọn thay thế tối ưu hóa.

Kỹ thuật tối ưu hóa này tính toán tỷ lệ học tập thích ứng cho mỗi tham số. Nó xác định động lượng và phương sai của gradient tổn thất và tận dụng hiệu ứng kết hợp để cập nhật các thông số trọng lượng.

Công thức toán học đơn giản hóa là:

*weights= weights– (Động lượng và Phương sai)*

### 3. 3. Tăng cường dữ liệu (Data augmentation):

Khi dữ liệu của tập huấn luyện quá ít, mô hình có thể sẽ bị *overfitting* hoặc *underfitting*:

- **Overfitting** là hiện tượng khi mô hình quá khớp với bộ dữ liệu huấn luyện nhưng trên tập dữ liệu chưa từng được huấn luyện thì kết quả dự đoán sẽ rất tệ. Hiện tượng xảy ra khi mô hình tìm được hàm mục tiêu khớp với chỉ riêng tập huấn luyện
- **Underfitting** là hiện tượng khi mô hình được huấn luyện có độ chính xác không cao ở bộ dữ liệu huấn luyện hoặc bất kì bộ dữ liệu khác

Để khắc phục hiện tượng mô hình xảy ra overfitting hoặc underfitting, thì cần mở rộng bộ dữ liệu huấn luyện hoặc thay đổi thuật toán dành cho huấn luyện.

Data augmentation là kỹ thuật làm tăng tính đa dạng của bộ dữ liệu bằng cách thực hiện các phép biến đổi ngẫu nhiên (phép dịch, quay, lật, thay đổi kích thước,...)

### 3. 4. Transfer learning:

Mô hình đã được huấn luyện trước là mô hình đã được huấn luyện trước đó trên bộ dữ liệu lớn. Việc sử dụng mô hình đã được

huấn luyện trước đó làm điểm bắt đầu để huấn luyện mô hình cho một mục đích, nhiệm vụ khác, việc này là transfer learning.

Transfer learning làm tăng độ chính xác của mô hình, giảm thời gian huấn luyện thậm chí tránh overfitting do mô hình trước đó đã được huấn luyện trên tập dữ liệu rất lớn. Có 2 loại transfer learning:

- **Trích xuất đặc trưng**: Sẽ chỉ loại bỏ đi lớp Fully Connected ở cuối cùng của mô hình và giữ lại toàn bộ các lớp tích chập của mô hình đã được huấn luyện. Việc này sẽ giữ lại tất cả các đặc trưng của ảnh để tiếp tục huấn luyện trên bộ dữ liệu mới.
- **Tinh chỉnh (Fine tuning)**: cũng giống như loại trên, ta chỉ loại bỏ đi lớp Fully Connected cuối cùng của mô hình nhưng thay vào đó ta sẽ thêm cho mạng nhưng lớp mới. Điều này tương tự như lấy mô hình được huấn luyện trước làm đầu vào cho mạng tích chập thêm vào.

Trong đồ án, em sử dụng tinh chỉnh để huấn luyện mô hình sử dụng transfer learning. Em bổ sung thêm các lớp mới đằng sau để gia tăng độ chính xác của mô hình.

## 4. Giới thiệu chung về Keras:

### 4. 1. Keras là gì ?

Keras là một mạng nơ-ron API cấp cao được viết bằng Python và có thể phát triển một mô hình học sâu đầy đủ chức năng với ít hơn 15 dòng code. Đơn giản hóa thì Keras giúp người dùng phát triển mô hình học sâu một cách nhanh chóng và cung cấp hàng tấn tính linh hoạt khi vẫn là một API cấp cao (có thể chạy trên các API cấp thấp như TensorFlow, CNTK, Theano).

Cho đến nay, Keras thường sử dụng TensorFlow làm backend và khi code được viết trên Keras sẽ được chuyển thành TensorFlow khi chạy.

Trong đồ án, em sẽ sử dụng Keras để huấn luyện các mô hình học sâu.

### 4. 2. Sequential

Mô hình *Sequentail* thích hợp cho phép dễ dàng tạo các lớp chồng lên nhau khi mà mỗi lớp chỉ có chính xác một tensor đầu vào và một tensor đầu ra.

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(10, input_dim=15))
model.add(Activation('relu'))
```

Đoạn mã trên tạo ra mạng tích chập gồm 10 nơ-ron nhận đầu vào là 15 nơ-ron với sau cùng là hàm kích hoạt ReLU.

## 4. 3. Các layers:

### 4. 3. 1. Conv2D:

Là lớp tích chập được tạo ra để thực hiện phép chập với đầu vào của lớp hiện tại

```
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1),
    padding="valid", data_format=None, dilation_rate=(1, 1),
    groups=1, activation=None, use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs)
```

Đầu ra của lớp tích chập được gọi là Features Map (trích xuất các đặc trưng của ảnh)

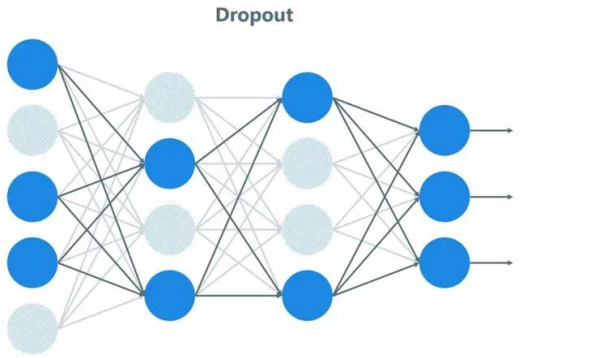
### 4. 3. 2. Pooling2D:

Keras bao gồm các lớp pooling với chức năng tính toán giá trị lớn nhất hoặc giá trị trung bình trong pool_size để giữ lấy đặc trưng nôi bật nhất của ảnh cũng như giảm kích thước ảnh đầu vào. Việc sử dụng pooling có thể giảm số lượng phép toán cần tính mà vẫn giữ được đặc trưng của ảnh.

Có 2 loại pooling trong keras: global pooling và pooling thông thường. Điểm khác biệt duy nhất là global pooling sẽ được tính toán trên toàn bộ kênh của lớp trước đó và đầu ra là một kênh duy nhất (tương tự lớp Flatten) và có thể sử dụng thay thế các lớp fully connected ở trên đỉnh của features map.

### 4. 3. 3. Dropout:

Lớp Dropout (bỏ học) trong mô hình học máy giúp mô hình giảm overfitting. Dropout sẽ ngẫu nhiên bỏ qua các nơ-ron của lớp trước đó theo tỷ lệ xác định (dropout_rate).




Sử dụng Dropout làm mô hình được huấn luyện trên mạng con khác nhau thay vì huấn luyện trên 1 mạng mẹ tại mỗi vòng lặp. Việc tồn tại các nơ-ron bị nhiễu mang lại kết quả sai cho mô hình nên việc bỏ học có thể loại bỏ đi những nơ-ron nhiễu để huấn luyện và làm giảm overfitting.

#### 4. 3. 4. Dense:

Lớp Dense là lớp thường xuyên được sử dụng, lớp kết nối tất cả các nơ-ron của lớp hiện tại với tất cả các nơ-ron của lớp trước đó.

```
keras.layers.Dense(units, activation=None, use_bias=True,
                   kernel_initializer='glorot_uniform',
                   bias_initializer='zeros',
                   kernel_regularizer=None,
                   bias_regularizer=None,
                   activity_regularizer=None,
                   kernel_constraint=None,
                   bias_constraint=None)
```

Trong đó:

*units* là chiều của không gian đầu ra

activation là hàm kích hoạt được sử dụng (nếu để trống thì hàm kích hoạt sẽ là tuyến tính a(x) = x)

Thông thường, khi sử Dense em chỉ thay đổi các tham số như *units* và *activation,* các tham số còn lại để mặc định. Các tham số khác chỉ quan trọng trong các trường hợp cụ thể.

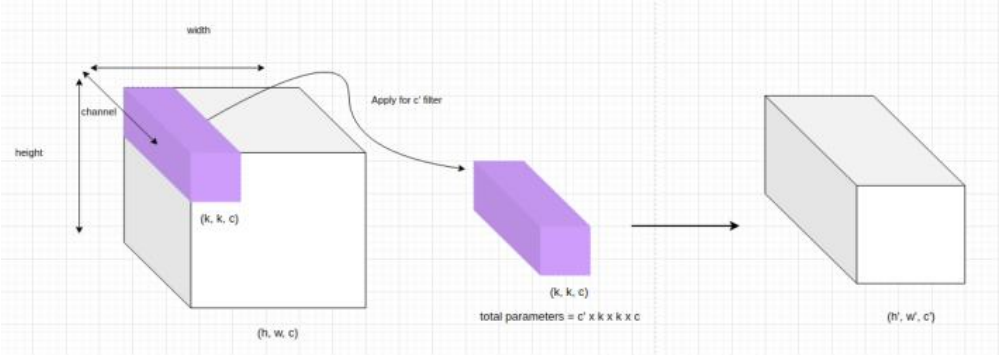
#### 4. 3. 5. BatchNormalization:

## 5. Giới thiệu về MobileNet:

### 5. 1. Cấu trúc mạng MobileNet:

Để phát triển được những ứng dụng AI trên các thiết bị có tài nguyên hạn chế như mobile, IoT thì chúng ta cần hiểu về tài nguyên của những thiết bị này để lựa chọn model phù hợp cho chúng. Những mô hình ưa chuộng được sử dụng thường là những model có số lượng tính toán ít và độ chính xác cao. MobileNet là một trong những lớp mô hình như vậy.

Như chúng ta đã biết tích chập 2 chiều thông thường sẽ được tính toán trên toàn bộ chiều sâu (channel). Do đó số lượng tham số của mô hình sẽ gia tăng đáng kể phụ thuộc vào độ sâu của layer trước đó.



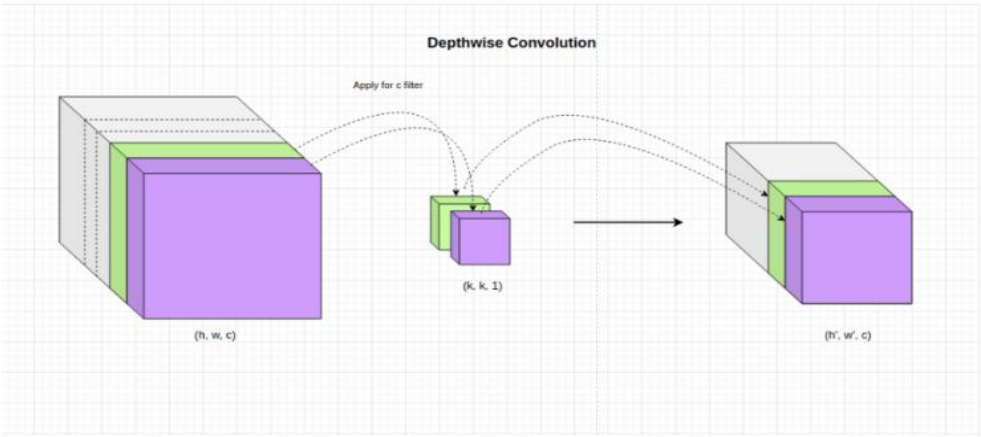

Chẳng hạn ở trên chúng ta có một đầu vào kích thước $h \times w \times c$, tích chập thông thường sẽ cần $k \times k \times c$ tham số để thực hiện tích chập trên toàn bộ độ sâu của layers. Mỗi một bộ lọc sẽ tạo ra một ma trận output kích thước $h' \times w' \times 1$. Áp dụng c′ bộ lọc khác nhau ta sẽ tạo ra đầu ra có kích thước $h' \times w' \times c'$ (các ma trận output khi áp dụng tích chập trên từng bộ lọc sẽ được concatenate theo chiều sâu). Khi đó số lượng tham số cần sử dụng cho một tích chập thông thường sẽ là: $c' \times k \times k \times c$.

Số lượng tham số sẽ không đáng kể nếu độ sâu của input channel c là nhỏ, thường là ở các layers ở vị trí đầu tiên. Tuy nhiên khi độ sâu tăng tiến dần về những layers cuối cùng của mạng CNN thì số lượng tham số của mô hình sẽ là một con số không hề nhỏ. Sự gia tăng tham số này tạo ra những mô hình cồng kềnh làm chiếm dụng bộ nhớ và ảnh hưởng tới tốc độ tính toán. Alexnet và VGGNet là những mô hình điển hình có số lượng tham số rất lớn do chỉ áp dụng những tích chập 2 chiều thông thường.

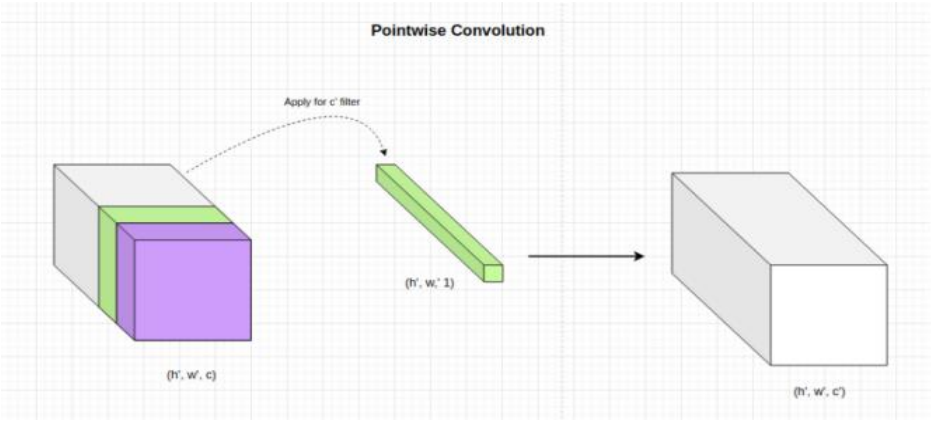
Mô hình thuộc họ MobileNet chỉ có vài triệu tham số nhưng độ chính xác tốt hơn AlexNet vài chục triệu tham số. Điểm mấu chốt tạo nên sự khác biệt này đó là MobileNet lần đầu tiên áp dụng kiến trúc *tích chập tách biệt chiều sâu.*

➢ **Tích chập tách biệt chiều sâu (Depthwise Separable Convolution):** 2 phần *tích chập chiều sâu* và *tích chập điểm*

- **Tích chập chiều sâu:** Chia khối tensor 3D thành những lát cắt ma trận theo độ sâu -> tích chập -> ghép lại. Lợi ích:
  - Nhận diện đặc trưng: Quá trình học và nhận diện đặc trưng sẽ được tách biệt theo từng bộ lọc. Nếu đặc trưng trên các channels là khác xa nhau thì sử dụng các bộ lọc riêng cho channel sẽ chuyên biệt hơn trong việc phát hiện các đặc trưng. Chẳng hạn như đầu vào là ba kênh RGB thì mỗi kênh áp dụng một bộ lọc khác nhau chuyên biệt.
  - Giảm thiểu khối lượng tính toán: Để tạo ra một điểm pixel trên output thì tích chập thông thường cần sử dụng $k \times k \times c$ phép tính trong khi tích chập chiều sâu tách biệt chỉ cần $k \times k$ phép tính.
  - Giảm thiểu số lượng tham số : Ở tích chập chiều sâu cần sử dụng c×k×k tham số. Số lượng này ít hơn gấp c′ lần so với tích chập chiều sâu thông thường.




- **Tích chập điểm:** Có tác dụng thay đổi độ sâu của output bước trên từ c sang c′. Chúng ta sẽ áp dụng c′ bộ lọc kích thước 1×1×c. Như vậy kích thước width và height không thay đổi mà chỉ độ sâu thay đổi.



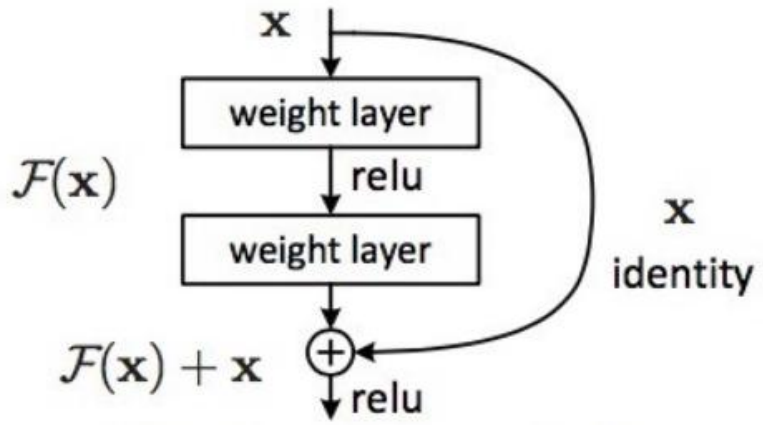

## 5. 2. Cấu trúc mạng MobileNetV2:

MobileNetV2 có một số điểm cải tiến so với MobileNetV1 giúp cho nó có độ chính xác cao hơn, số lượng tham số và số lượng các phép tính ít hơn.

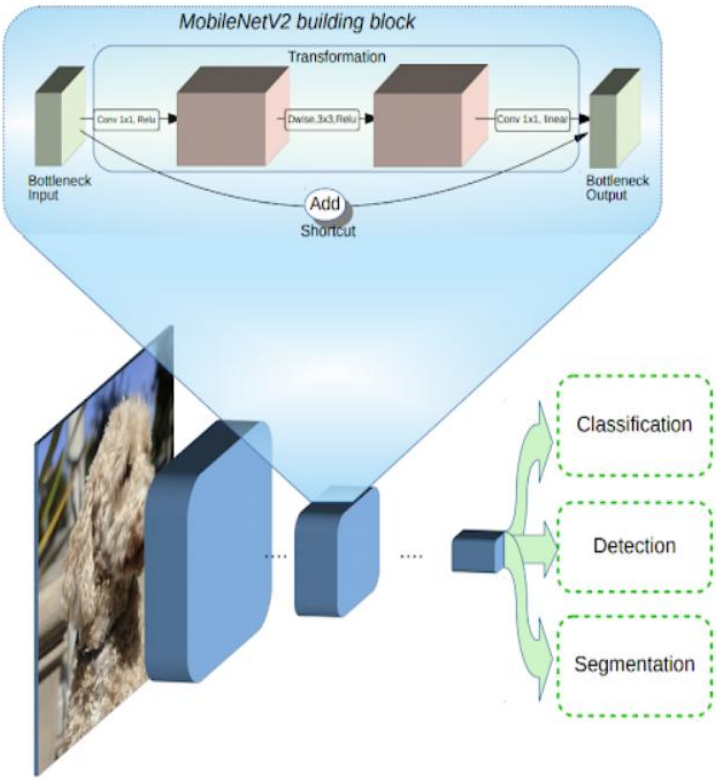
MobileNetV2 tiếp tục sử dụng MobileNetV1 làm nền tảng nhưng với 2 sự thay đổi mới:

+ Inverted Residual Block
+ Loại bỏ non-linear

MobileNetV2 cũng sử dụng các kết nối tắt. Các khối ở layer trước được cộng trực tiếp vào các layer liền sau (Residual Block)




Tuy nhiên kết nối tắt ở MobileNetV2 được điều chỉnh sao cho số kênh (hoặc chiều sâu) ở input và output của mỗi residual block được thắt hẹp lại. Chính vì thế nó được gọi là các bottleneck layers (bottleneck là một thuật ngữ thường được sử dụng trong deep learning để ám chỉ các kiến trúc thu hẹp kích thước theo một chiều nào đó).

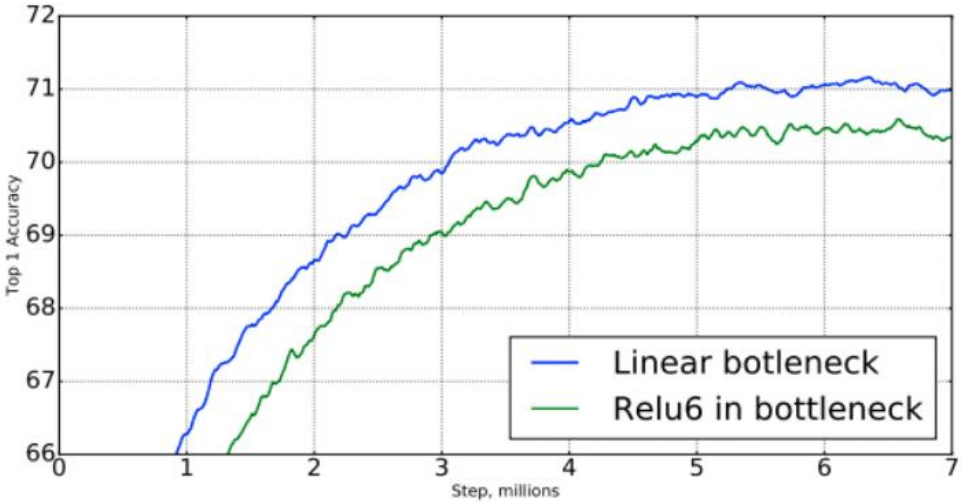



Kiến trúc residual này ngược lại so với các kiến trúc residual truyền thống vì kiến trúc residual truyền thống có số lượng kênh ở input và output của một block lớn hơn so với các layer trung gian. Chính vì vậy nó còn được gọi là kiến trúc *inverted residual block*.

Tác giả cho rằng các layer trung gian trong một block sẽ làm nhiệm vụ biến đổi phi tuyến nên cần dày hơn để tạo ra nhiều phép biến đổi hơn. Kết nối tắt giữa các block được thực hiện trên những bottleneck input và output chứ không thực hiện trên các layer trung gian. Do đó các layer bottleneck input và output chỉ cần ghi nhận kết quả và không cần thực hiện biến đổi phi tuyến.

Ở giữa các layer trong một block *inverted residual block* chúng ta cũng sử dụng những biến đổi *tích chập tách biệt chiều sâu* để giảm thiểu số lượng tham số của mô hình. Đây cũng chính là bí quyết giúp họ các model MobileNet có kích thước giảm nhẹ.

Một trong những thực nghiệm được tác giả ghi nhận đó là việc sử dụng các biến đổi phi tuyến (như biến đổi qua ReLu hoặc sigmoid) tại input và output của các *residual block* sẽ làm cho thông tin bị mất mát. Cụ thể cùng xem kết quả thực nghiệm bên dưới:




Chính vì thế trong kiến trúc của *residual block* tác giả đã loại bỏ hàm phi tuyến tại layer input và output và thay bằng các phép chiếu tuyến tính.

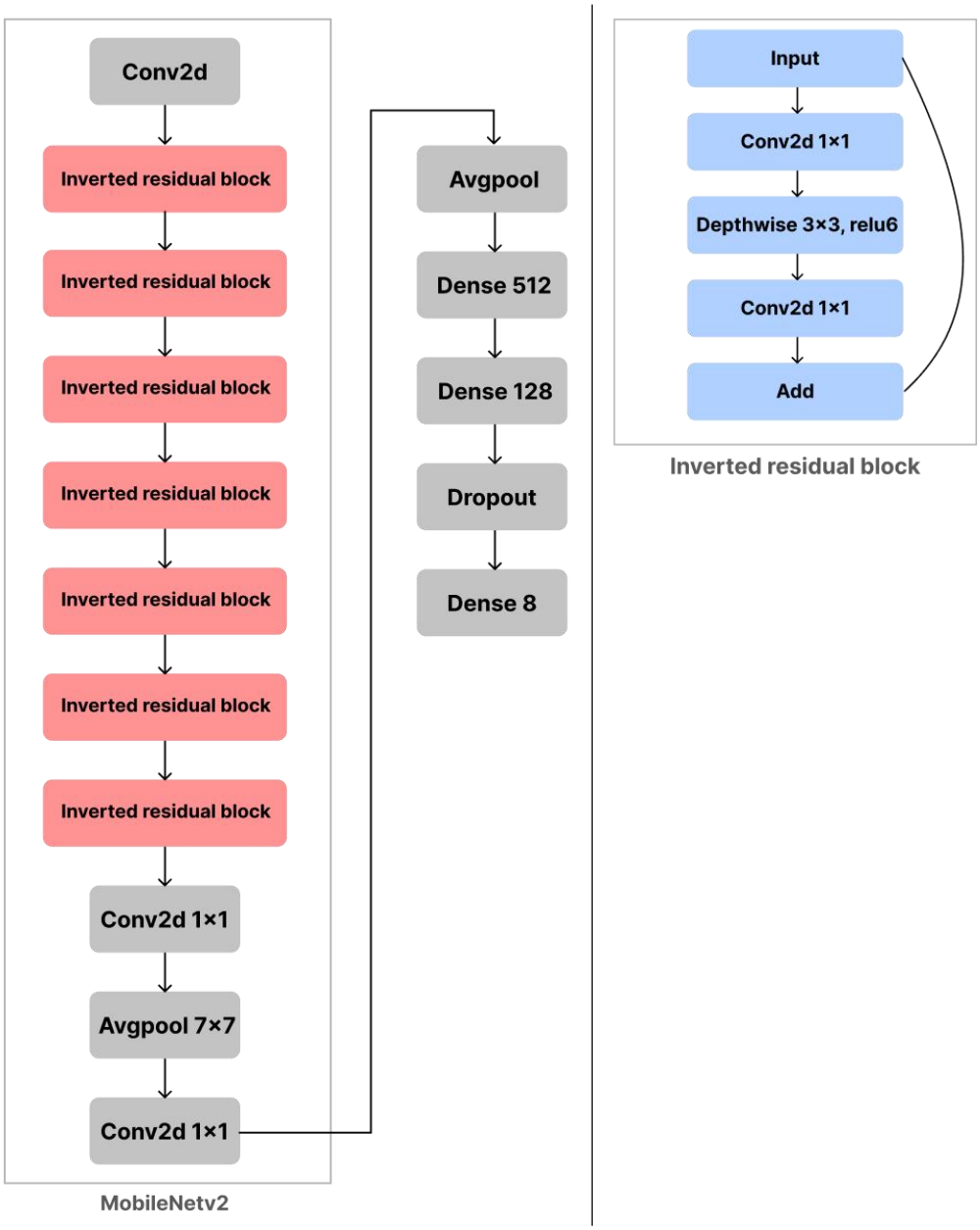
Với mạng MobileNetV3, có sự thay đổi thêm Squeeze and Excitation (SE) vào block Residual để tạo thành một kiến trúc có độ chính xác cao hơn. Nhưng sự hiệu quả của MobileNetV3 với MobileNetV2 chỉ rõ rệt ở bài toán phân loại nên trong đồ án em sử dụng MobileNetV2 để huấn luyện mô hình.

# Chương III: Triển khai mô đun phát hiện và làm phẳng ảnh

## 1. Mạng sử dụng trong huấn luyện:

### 1. 1. Mạng tích chập sử dụng mobilenetv2 làm lõi:

**Cấu trúc mạng:**



**Các tham số:**
Tổng tham số của mô hình cho huấn luyện: *2,983,112*

| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

**Trong đó:**

t là yếu tố mở rộng
c là kích thước của output
n là số lần lặp lại *'bottleneck'* trong *'inverted residual block'*
s là stride trong tích chập tách biệt chiều sâu

**Bottleneck:**

```
def _bottleneck(inputs, filters, kernel, t, alpha, s, r=False):
    channel_axis = 1 if K.image_data_format() == 'channels_first' else -1
    # Depth
    tchannel = K.int_shape(inputs)[channel_axis] * t
    # Width
    cchannel = int(filters * alpha)

    x = _conv_block(inputs, tchannel, (1, 1), (1, 1))

    x = DepthwiseConv2D(kernel, strides=(s, s), depth_multiplier=1, padding='same')(x)
    x = BatchNormalization(axis=channel_axis)(x)
    x = Activation(relu6)(x)

    x = Conv2D(cchannel, (1, 1), strides=(1, 1), padding='same')(x)
    x = BatchNormalization(axis=channel_axis)(x)

    if r:
        x = Add()([x, inputs])

    return x
```

## Inverted residual block:

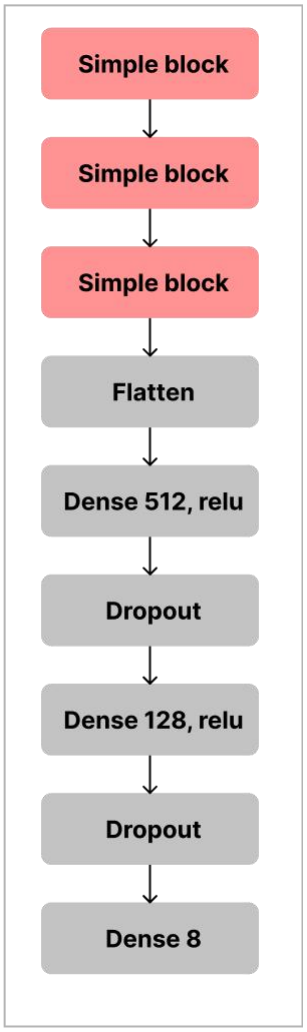
```
def _inverted_residual_block(inputs, filters, kernel, t, alpha, strides, n):
    x = _bottleneck(inputs, filters, kernel, t, alpha, strides)

    for i in range(1, n):
        x = _bottleneck(x, filters, kernel, t, alpha, 1, True)

    return x
```

## MobileNetv2:

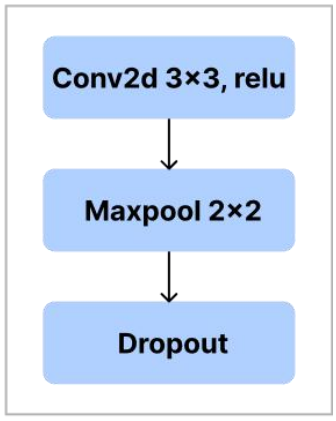
```
def MobileNetv2(input_shape, k, alpha=1.0):
    inputs = Input(shape=input_shape)

    first_filters = _make_divisible(32 * alpha, 8)
    x = _conv_block(inputs, first_filters, (3, 3), strides=(2, 2))

    x = _inverted_residual_block(x, 16, (3, 3), t=1, alpha=alpha, strides=1, n=1)
    x = _inverted_residual_block(x, 24, (3, 3), t=6, alpha=alpha, strides=2, n=2)
    x = _inverted_residual_block(x, 32, (3, 3), t=6, alpha=alpha, strides=2, n=3)
    x = _inverted_residual_block(x, 64, (3, 3), t=6, alpha=alpha, strides=2, n=4)
    x = _inverted_residual_block(x, 96, (3, 3), t=6, alpha=alpha, strides=1, n=3)
    x = _inverted_residual_block(x, 160, (3, 3), t=6, alpha=alpha, strides=2, n=3)
    x = _inverted_residual_block(x, 320, (3, 3), t=6, alpha=alpha, strides=1, n=1)

    if alpha > 1.0:
        last_filters = _make_divisible(1280 * alpha, 8)
    else:
        last_filters = 1280

    x = _conv_block(x, last_filters, (1, 1), strides=(1, 1))
    x = GlobalAveragePooling2D()(x)
    x = Reshape((1, 1, last_filters))(x)
    x = Dropout(0.3, name='Dropout')(x)
    x = Conv2D(k, (1, 1), padding='same')(x)

    x = Dense(k)(x)

    model = Model(inputs, x)
    # plot_model(model, to_file='images/MobileNetv2.png', show_shapes=True)

    return model
```

## 1. 2. Mạng nơ-ron tích chập đơn giản:

**Cấu trúc mạng:**




Simple CNN




Simple block

**Các tham số:**

Tổng tham số của mô hình huấn luyện: *3,433,056*

| Input | Operator | c |
|---|---|---|
| $64^2$ x 3 | Conv2d 3x3 | 32 |
| $64^2$ x 32 | MaxPool 2x2 | 32 |
| $31^2$ x 32 | Conv2d 3x3 | 64 |
| $30^2$ x 64 | MaxPool 2x2 | 64 |
| $15^2$ x 64 | Conv2d 3x3 | 128 |
| $14^2$ x 128 | MaxPool 2x2 | 128 |
| $7^2$ x 128 | Flatten | |

**Trong đó:**

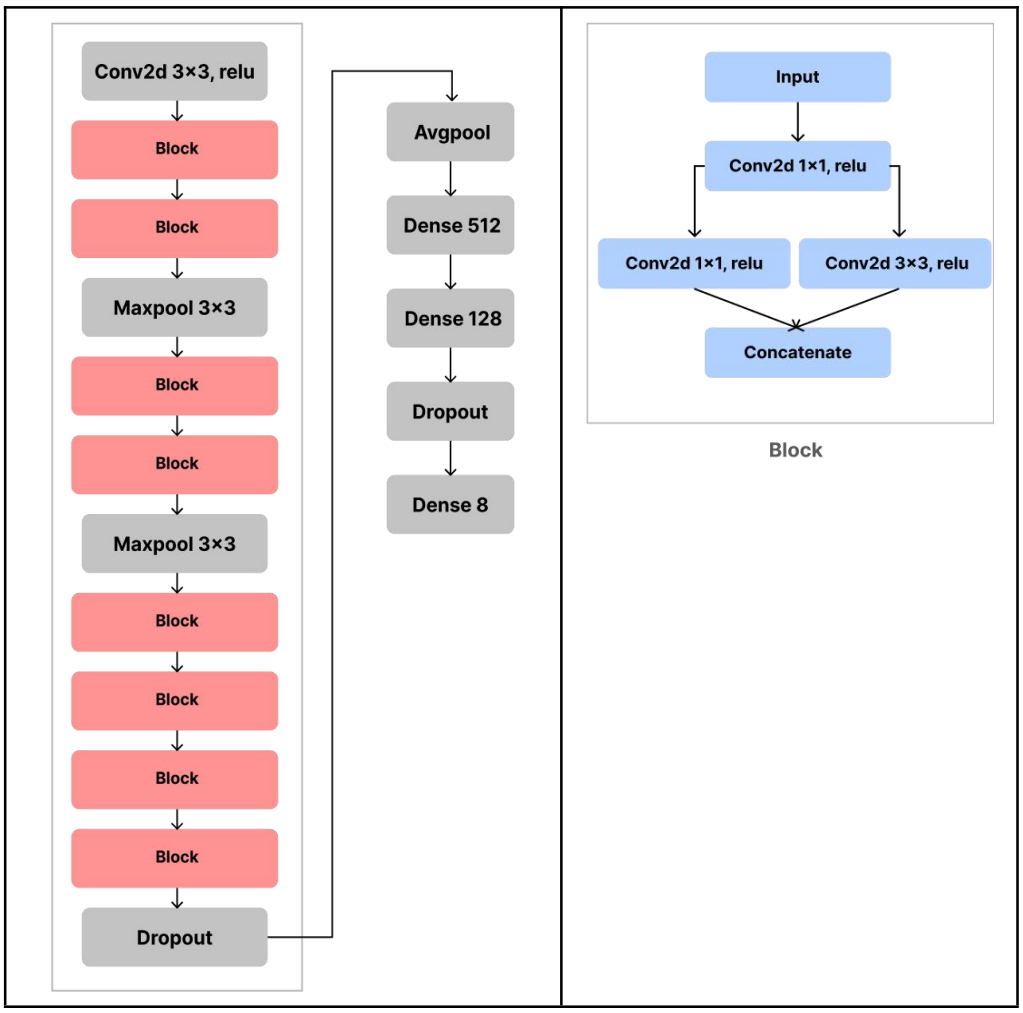
c là kích thước của output

**Mạng CNN đơn giản:**

```
def SimpleCNN(input_shape = (64, 64, 3), classes = 8):
    inputs = Input(input_shape)

    x = Conv2D(32,(3, 3), input_shape = input_shape)(inputs)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size = (2,2))(x)
    x = Dropout(0.1)(x)

    x = Conv2D(64,(2,2))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size = (2,2))(x)
    x = Dropout(0.1)(x)

    x = Conv2D(128,(2,2))(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Dropout(0.1)(x)

    x = Flatten()(x)

    x = Dense(512)(x)
    x = Activation('relu')(x)
    x = Dropout(0.1)(x)

    x = Dense(128)(x)
    x = Activation('relu')(x)
    x = Dropout(0.1)(x)

    x = Dense(classes)(x)

    return Model(inputs, x)
```

## 1. 3. Mạng nơ-ron tích chập 2:

**Cấu trúc mạng:**




**Các tham số:**

Tổng tham số của mô hình cho huấn luyện: 1,051,848

| Input | Operator | f | r | c |
|---|---|---|---|---|
| $64^2$ x 3 | Conv2d 3x3 | - | - | 64 |
| $32^2$ x 64 | MaxPool 3x3 | - | - | - |
| $15^2$ x 64 | Block 1 | 16 | 4 | 128 |
| $15^2$ x 128 | Block 2 | 16 | 4 | 128 |
| $15^2$ x 128 | MaxPool 3x3 | - | - | - |
| $7^2$ x 128 | Block 3 | 32 | 4 | 256 |
| $7^2$ x 256 | Block 4 | 32 | 4 | 256 |
| $7^2$ x 256 | MaxPool 3x3 | - | - | - |
| $3^2$ x 256 | Block 5 | 48 | 4 | 384 |
| $3^2$ x 384 | Block 6 | 48 | 4 | 384 |
| $3^2$ x 384 | Block 7 | 64 | 4 | 512 |
| $3^2$ x 512 | Block 8 | 64 | 4 | 512 |
| | GlobalAvgPool | - | - | - |
| 512 | Dense | - | - | 512 |
| 512 | Dense | - | - | 128 |
| 128 | Dense | - | - | k |

**Trong đó:**

f là số lượng đầu ra của conv2d đầu trong block
r là hệ số
c là kích thước đầu ra của output (f * r * 2)

**Block**

```
def create_block(x, first_filter, name, ratio = 4):
    second_filter = ratio * first_filter

    squeeze    = Conv2D(first_filter,(1,1), activation='relu', padding='same')(x)
    expand_1x1 = Conv2D(second_filter, (1,1), activation='relu', padding='same')(squeeze)
    expand_3x3 = Conv2D(second_filter, (3,3), activation='relu', padding='same')(squeeze)

    axis = get_axis()
    x_ret = Concatenate(axis=axis, name='%s_concatenate'%name)([expand_1x1, expand_3x3])

#     x_ret = Add(name='%s_concatenate_bypass'%name)([x_ret, x])

    return x_ret
```

**Mạng CNN:**

```
def CNN(input_shape, nb_classes, dropout_rate=0.1):
    input_img = Input(shape=input_shape)

    x = Conv2D(64, (3,3), activation='relu', strides=(2,2), padding='same', name='conv1')(input_img)

    x = MaxPooling2D(pool_size=(3,3), strides=(2,2), name='maxpool1')(x)

    x = create_block(x, 16, name='block1')
    x = create_block(x, 16, name='block2')

    x = MaxPooling2D(pool_size=(3,3), strides=(2,2), name='maxpool3')(x)

    x = create_block(x, 32, name='block3')
    x = create_block(x, 32, name='block4')

    x = MaxPooling2D(pool_size=(3,3), strides=(2,2), name='maxpool5')(x)

    x = create_block(x, 48, name='block5')
    x = create_block(x, 48, name='block6')
    x = create_block(x, 64, name='block7')
    x = create_block(x, 64, name='block8')

    x = Dropout(dropout_rate)(x)

    x = GlobalAveragePooling2D(name='avgpool10')(x)
    x = Dense(512)(x)
    x = Dense(128)(x)
    x = Dropout(dropout_rate)(x)
    x = Dense(nb_classes)(x)

    return Model(inputs=input_img, outputs=x)
```

## 2. Hướng tiếp cận:

- 4 điểm góc của tài liệu là 4 điểm quan trọng cần tìm kiếm: *góc trên trái(TL), góc trên phải(TR), góc dưới trái(BL), góc dưới phải(BR)*
- 4 điểm (x1, y1), (x2, y2), (x3, y3), (x4, y4) là tọa độ của 4 điểm TL, TR, BR, BL
- 2 giai đoạn: dự đoán 4 góc của tài liệu sử dụng mô hình mạng tích chập được huấn luyện, đệ quy từng dự đoán để tìm kiếm kết quả chính xác

- **Giai đoạn 1:**

- Em sử dụng các mô hình mạng tích chập nêu trên để huấn luyện
- Huấn luyện trên tập dataset có kích thước 64 x 64 để giảm thời gian huấn luyện và tài nguyên
- Dự đoán 4 góc của tài liệu
- *Region extractor (trích xuất 4 miền tương ứng với mỗi điểm trên):* TL', TR', BR', BL' là 4 điểm dự đoán ở trên. Ta sẽ trích xuất các vùng chứa các điểm dự đoán tương ứng:
  - Cắt ảnh dọc theo trục x của điểm trên/dưới dự đoán và trục y của điểm trái/phải dự đoán
  - Xét tại điểm TR', biên trái sẽ là đường thẳng song song với trục y đi qua trung điểm của đoạn TL'TR'. Biên dưới sẽ là đường thẳng song song với trục x đi qua trung điểm của đoạn TR'BR'. 2 biên còn lại sẽ được lấy đối xứng qua điểm TR'. **Tương tự với các điểm khác**

- **Giai đoạn 2:**
  - Mỗi vùng (region) sẽ được đưa vào mạng tích chập để tiếp tục dự đoán góc chính xác trong vùng đó và lặp lại quá trình đến khi đạt được điều kiện dừng
  - Mỗi bước sẽ crop ảnh đi một số RF (Retain Factor) ($0 < RF < 1$). Tại mỗi lần lặp, giữ lại một phần ảnh gần điểm dự đoán nhất với hệ số RF theo cả chiều ngang và dọc. Nghĩa là, một bức ảnh có kích thước (H, W), tại lần lặp thứ n thì kích thước của ảnh sẽ là (H x , W x ). Nếu kích thước của ảnh xuống dưới 10x10 thì quá trình sẽ được dừng lại.
- **Giai đoạn 3:** Làm phẳng ảnh sử dụng kỹ thuật chuyển đổi phối cảnh trong opencv2

## 3. Quá trình huấn luyện

### 3. 1. Tập dữ liệu dùng để huấn luyện:

**Dataset dùng cho huấn luyện mô hình 1:**

**Dataset gồm 2 dataset:**

1. SmartDoc 2015 - Challenge 1

2. Dataset tương tự dataset 1 nhưng đa dạng hơn về background

**SmartDoc 2015 - Challenge 1:**

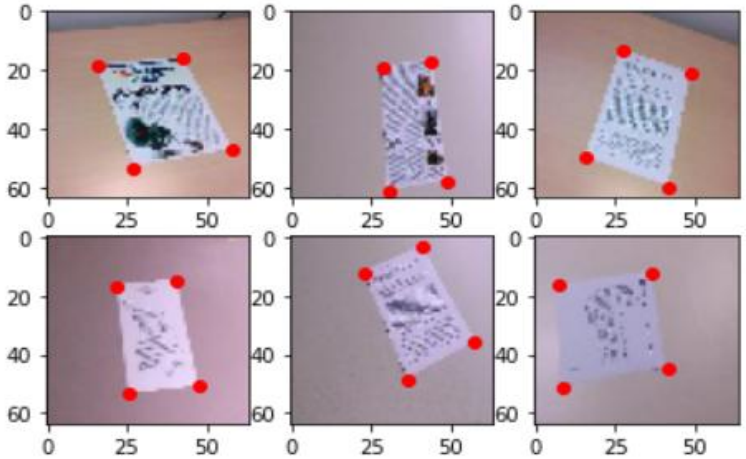
6 loại tài liệu khác nhau đến từ cơ sở dữ liệu công cộng và năm hình ảnh tài liệu cho mỗi lớp. Bao gồm các kiểu khác nhau để chúng bao gồm các lược đồ và nội dung bố cục tài liệu khác nhau (hoặc hoàn toàn bằng văn bản hoặc có nội dung đồ họa cao).

Mỗi mô hình tài liệu này được in bằng tia laser màu trên giấy thường định dạng A4 và đã tiến hành chụp chúng bằng máy tính bảng Google Nexus 7. Dataset bao gồm các video clip nhỏ khoảng 10 giây cho mỗi tài liệu trong số 30 tài liệu trong 5 tình huống nền khác nhau. Các video được quay bằng độ phân giải Full HD 1920x1080 ở tốc độ khung hình thay đổi. Vì được quay video bằng cách cầm tay và di chuyển máy tính bảng, nên các khung hình video có hiện tượng biến dạng thực tế như lấy nét và mờ chuyển động, phối cảnh, thay đổi độ sáng và thậm chí là bị che khuất một phần các trang tài liệu. Tổng kết, cho đến nay, cơ sở dữ liệu bao gồm 150 video clip với khoảng 24.000 khung hình.

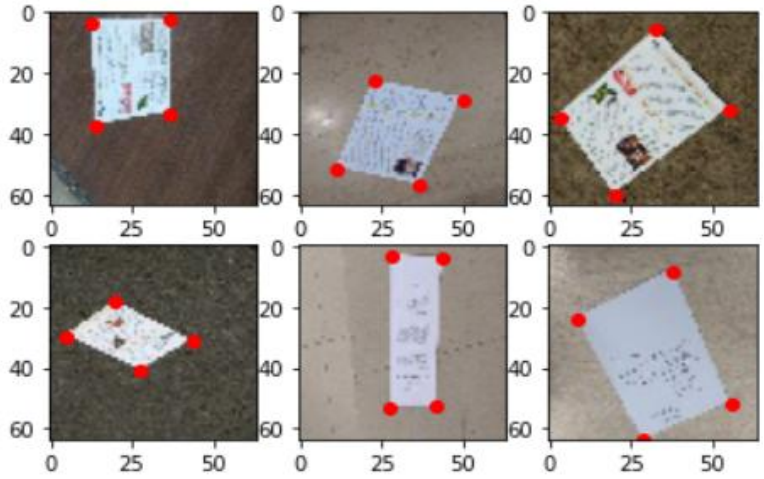
Chú thích của tập dữ liệu là tọa độ 4 điểm của 4 góc tài liệu trong khung hình

**Trong mô hình huấn luyện em sử dụng:**

Tập dữ liệu sẽ được thay đổi kích thước về mức 64 x 64. Với tập dữ liệu bao gồm 16.765 ảnh cho tập huấn luyện và 6.158 ảnh cho tập test.



Với tập dữ liệu 2 gồm 14.208 ảnh cho tập huấn luyện và 7.104 ảnh cho tập test.



Ground truth của các ảnh được cho vào tệp csv tương ứng với 8 giá trị cho từng ảnh tương ứng cho tọa độ của 4 góc cần phát hiện

**Tập dữ liệu dùng cho huấn luyện mô hình 2:**

Với mô hình 2, tập dữ liệu được tạo bằng cách cắt riêng từng góc của tập dữ liệu dùng cho việc huấn luyện mô hình 1. Bởi vậy, việc thí nghiệm độc lập cho từng kiểu dữ liệu là không cần thiết cho mô hình 2 vì kết quả của thí nghiệm giống với các thí nghiệm tương ứng khi huấn luyện mô hình 1.

Với việc huấn luyện mô hình 2, tập dữ liệu bao gồm 100 000 ảnh với kích thước 64x64 với 50 vòng lặp và Adam(0.0001).

**Tập dữ liệu dùng cho việc kiểm tra kết quả dự đoán:**

Tập dữ liệu bao gồm 25 ảnh thu được trực tiếp từ máy ảnh của điện thoại. Em chụp với nhiều góc và màu nền cũng như độ phức tạp khác nhau để kiểm tra mức độ hiệu quả của mô hình với ảnh thực tế.

## 3. 2. Quá trình huấn luyện:

### 3. 2. 1. Huấn luyện với mạng tích chập sử dụng MobileNetv2 làm lõi

## Quá trình huấn luyện:

- Các bước:

*1. Load thông tin của tệp csv chứa ground truth của tập dữ liệu*

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras.utils import Sequence
import numpy as np
import csv

train_path = "/content/docTrain"
# test_path = "D:/Dataset/r-cnn/smartDocData_DocTestC/smartDocData_DocTestC/"

import pandas as pd
import numpy as np

doc_train = pd.read_csv(train_path + "/total_gt.csv", header=None)
# doc_test = pd.read_csv(test_path + "gt2.csv", header=None)

# tập train nhận 90% dataset và 10% cho validation
train, val = np.split(doc_train.sample(frac=1), [int(0.9*len(doc_train))])
```

*2. Load tập dữ liệu với ImageDataGenerator*

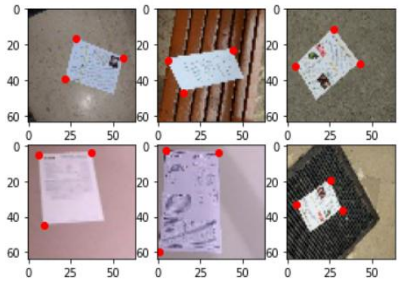
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        # rotation_range=30,
        # shear_range=0.3,
        # width_shift_range=0.1,
        # height_shift_range=0.1,
        # zoom_range=0.25,
)

valid_datagen = ImageDataGenerator(
        rescale=1./255,
)
```

```
# tạo tập train
train_g = train_datagen.flow_from_dataframe(
    train, directory=train_path,
    x_col=0, y_col=[1, 2, 3, 4, 5, 6, 7, 8],
    target_size=(64, 64), batch_size=32,
    shuffle=True,
    class_mode="raw")

# tạo tập validation
valid_g = valid_datagen.flow_from_dataframe(
    val, directory=train_path,
    x_col=0, y_col=[1, 2, 3, 4, 5, 6, 7, 8],
    target_size=(64, 64), batch_size=32,
    shuffle=False,
    class_mode="raw")
```

```
Found 27875 validated image filenames.
Found 3098 validated image filenames.
```

*3. Trực quan hóa dữ liệu*

```
import matplotlib.pyplot as plt

count = 231
for i in range(0,6):
  plt.subplot(count)
  count += 1

  images, labels = train_g.next()

  plt.plot(labels[0][0] * 64, labels[0][1] * 64, 'ro', 10)
  plt.plot(labels[0][2] * 64, labels[0][3] * 64, 'ro', 10)
  # plt.plot(labels[0][4] * 64, labels[0][5] * 64, 'ro', 10)
  plt.plot(labels[0][6] * 64, labels[0][7] * 64, 'ro', 10)

  plt.imshow(images[0])
```



## 4. Khởi tạo mạng nơ-ron để huấn luyện

```
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from keras import Sequential

base = tf.keras.applications.MobileNetV2(input_shape=(64,64,3),include_top=False,weights='imagenet')
base.trainable = True

model = Sequential()
model.add(base)
# model.add(Flatten())
model.add(GlobalAveragePooling2D())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(8))

model.summary()

model.compile(loss='mean_squared_error',
              optimizer=tf.keras.optimizers.Adam(0.0001),
              metrics=['accuracy'])
```

## 5. Huấn luyện mô hình

```
history = model.fit(train_g, validation_data=valid_g, epochs=100, callbacks=callbacks)
```

```
Epoch 1/100
872/872 [==============================] - 27s 21ms/step - loss: 0.0277 - accuracy: 0.4399 - val_loss: 0.0581 - val_accuracy: 0.5781
Epoch 2/100
872/872 [==============================] - 18s 21ms/step - loss: 0.0164 - accuracy: 0.5485 - val_loss: 0.0342 - val_accuracy: 0.6501
Epoch 3/100
872/872 [==============================] - 18s 20ms/step - loss: 0.0124 - accuracy: 0.6006 - val_loss: 0.0192 - val_accuracy: 0.7085
Epoch 4/100
872/872 [==============================] - 18s 20ms/step - loss: 0.0100 - accuracy: 0.6377 - val_loss: 0.0135 - val_accuracy: 0.7298
Epoch 5/100
872/872 [==============================] - 18s 21ms/step - loss: 0.0084 - accuracy: 0.6670 - val_loss: 0.0084 - val_accuracy: 0.7460
Epoch 6/100
872/872 [==============================] - 18s 20ms/step - loss: 0.0072 - accuracy: 0.6890 - val_loss: 0.0061 - val_accuracy: 0.7792
Epoch 7/100
872/872 [==============================] - 18s 21ms/step - loss: 0.0064 - accuracy: 0.7173 - val_loss: 0.0068 - val_accuracy: 0.8002
```

- Em huấn luyện từng dataset với 100 epochs
- Sử dụng thuật toán tối ưu Adam với learning rate = 0.0001, loss=”mean_squared_error” (MSE)
- Kích thước ảnh được thay đổi từ kích thước gốc 1920x1080 về 64x64 nhưng giữ nguyên tỷ lệ chuẩn của tài liệu trong ảnh để việc huấn luyện vẫn hiệu quả so với kích thước gốc nhưng giảm thời gian và tài nguyên cho huấn luyện
- Em thực hiện huấn luyện trên GPU của Google Colab
- Em đã thực hiện các thí nghiệm sau:
    - **Thí nghiệm 1:** Huấn luyện trên tập dataset 1
    - **Thí nghiệm 2:** Huấn luyện trên tập dataset 2
    - **Thí nghiệm 3:** Huấn luyện trên cả 2 tập dataset với data augmentation:
    - **Thí nghiệm 4:**
        - Huấn luyện mô hình khi trộn lẫn cả 2 dataset
        - Thí nghiệm này cần nhiều vòng lặp hơn cho việc hội tụ do độ đa dạng về mặt tài nguyên và độ nhiễu cao hơn khi huấn luyện trên từng tập riêng biệt
        - Thí nghiệm thực hiện 100 epochs với thuật toán tối ưu Adam và learning rate = 0.0001

**Kết quả thí nghiệm:**

+ **Thí nghiệm 1 và thí nghiệm 2:**

Mô hình 1 đạt độ chính xác 87% với dataset tương ứng và 54% khi kiểm tra trên tệp dữ liệu chưa từng thấy

Mô hình 2 đạt độ chính xác 52% với dataset tương ứng và 28% khi kiểm tra trên tệp dữ liệu chưa từng thấy

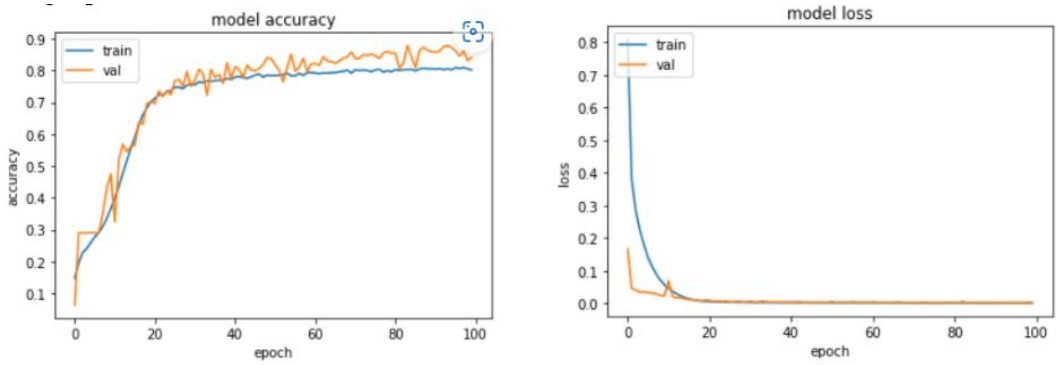
| | Tập dữ liệu 1 | Tập dữ liệu 2 | Tập kiểm thử |
|---|---|---|---|
| Mô hình 1 | 0.8795 | 0.5431 | |
| Mô hình 2 | 0.2840 | 0.5294 | |

Kết quả khi kiểm tra mô hình với các dữ liệu test tương ứng cho dataset tương ứng. Như kết quả cho thấy, khi mô hình dự đoán cho input là một dữ liệu chưa từng thấy khi huấn luyện thì kết quả không khả quan, độ chính xác đạt rất thấp

Tập dữ liệu 2 rất khó hội tụ khi thực hiện ít vòng lặp

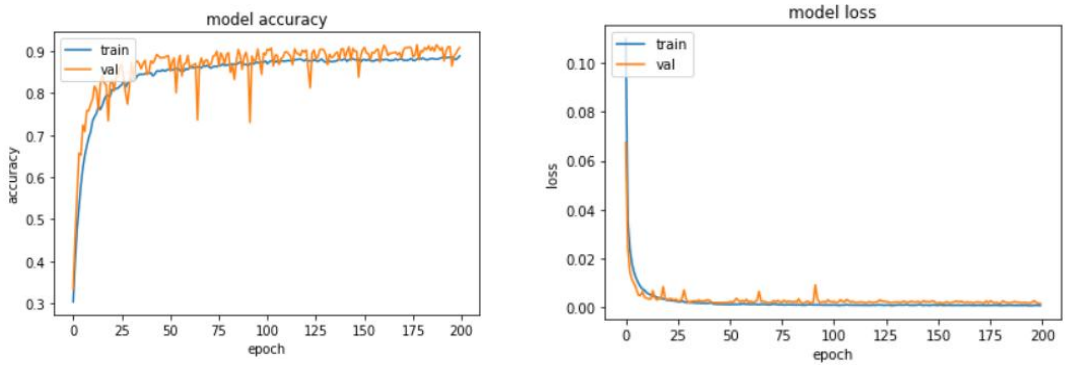
+ **Thí nghiệm 3:** Data augmentation không mang lại hiệu quả cao với 2 tập dữ liệu trên

+ **Thí nghiệm 4:** Mô hình đạt độ chính xác 80% trên tập huấn luyện sau 100 vòng lặp



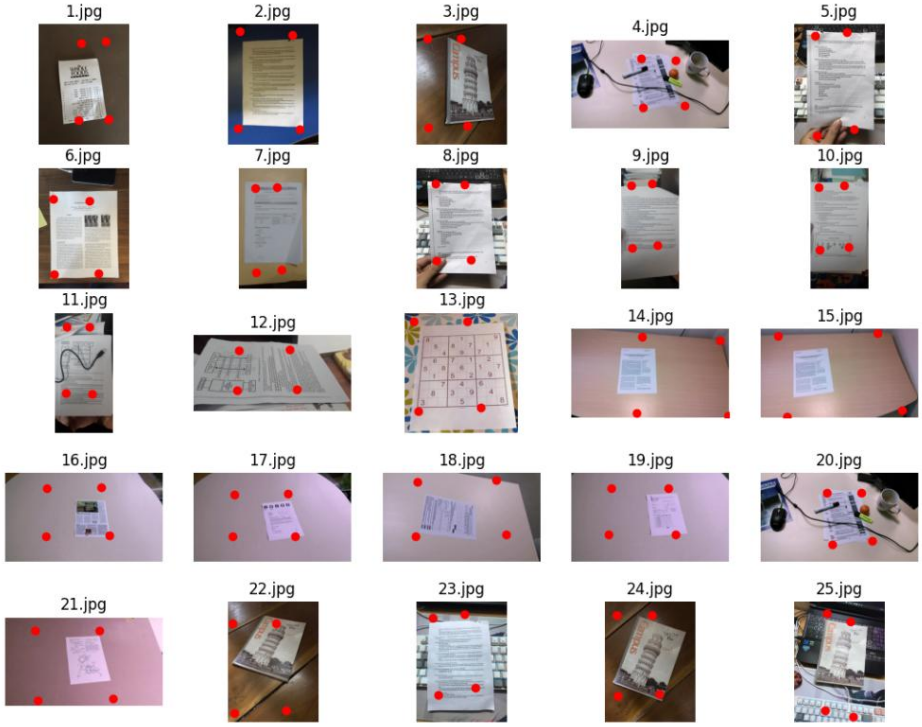

Áp dụng transfer learning với pre-weight là imagenet, epochs = 200 với cùng mạng nơ-ron: kết quả mô hình đạt ngưỡng 80% tại vòng lặp thứ 25 (nhanh hơn rất nhiều so với 100 vòng lặp khi không sử dụng transfer learning). Ở vòng lặp 100, mô hình đạt độ chính xác 87% và ở vòng lặp 200, mô hình đạt độ chính xác 89% trên tập huấn luyện

```
Epoch 190/200
872/872 [==============================] - 30s 34ms/step - loss: 9.6130e-04 - accuracy: 0.8831 - val_loss: 0.0023 - val_accuracy: 0.9022
Epoch 191/200
872/872 [==============================] - 30s 34ms/step - loss: 9.5385e-04 - accuracy: 0.8849 - val_loss: 0.0019 - val_accuracy: 0.9087
Epoch 192/200
872/872 [==============================] - 30s 34ms/step - loss: 9.5773e-04 - accuracy: 0.8831 - val_loss: 0.0021 - val_accuracy: 0.8764
Epoch 193/200
872/872 [==============================] - 33s 38ms/step - loss: 0.0010 - accuracy: 0.8843 - val_loss: 0.0021 - val_accuracy: 0.9003
Epoch 194/200
872/872 [==============================] - 30s 35ms/step - loss: 8.8907e-04 - accuracy: 0.8849 - val_loss: 0.0019 - val_accuracy: 0.9099
Epoch 195/200
872/872 [==============================] - 30s 35ms/step - loss: 8.7107e-04 - accuracy: 0.8855 - val_loss: 0.0019 - val_accuracy: 0.9106
Epoch 196/200
872/872 [==============================] - 30s 34ms/step - loss: 8.7706e-04 - accuracy: 0.8856 - val_loss: 0.0022 - val_accuracy: 0.8644
Epoch 197/200
872/872 [==============================] - 30s 34ms/step - loss: 9.3322e-04 - accuracy: 0.8822 - val_loss: 0.0028 - val_accuracy: 0.8880
Epoch 198/200
872/872 [==============================] - 30s 35ms/step - loss: 0.0011 - accuracy: 0.8795 - val_loss: 0.0021 - val_accuracy: 0.8932
Epoch 199/200
872/872 [==============================] - 33s 37ms/step - loss: 9.3242e-04 - accuracy: 0.8836 - val_loss: 0.0018 - val_accuracy: 0.9028
Epoch 200/200
872/872 [==============================] - 30s 35ms/step - loss: 8.9879e-04 - accuracy: 0.8882 - val_loss: 0.0017 - val_accuracy: 0.9087
```

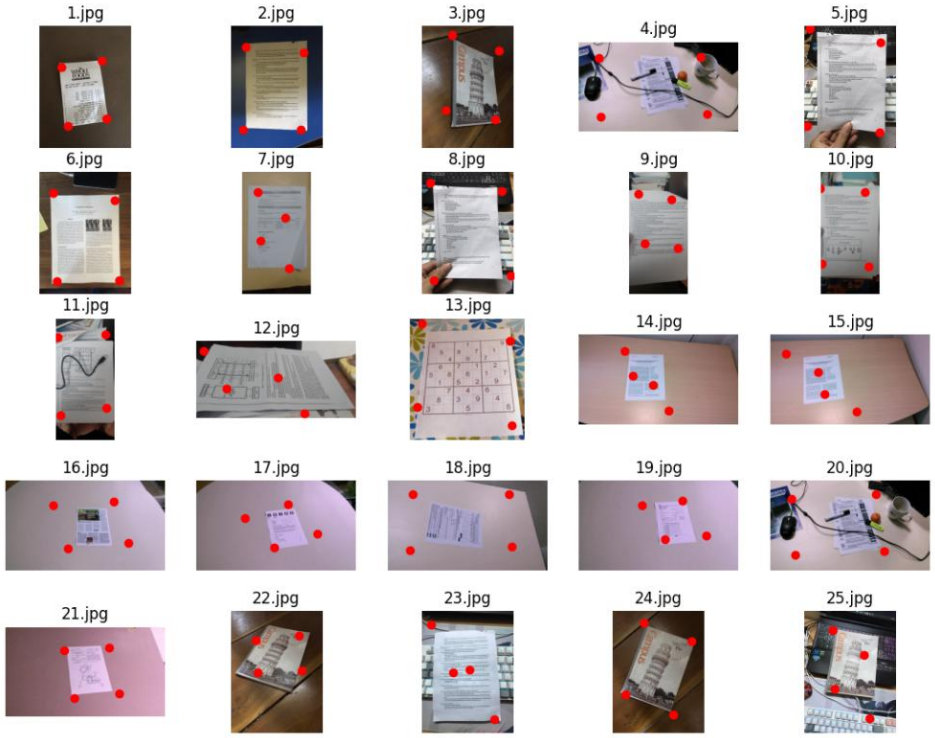


## Kết quả dự đoán:

## Thí nghiệm 1:

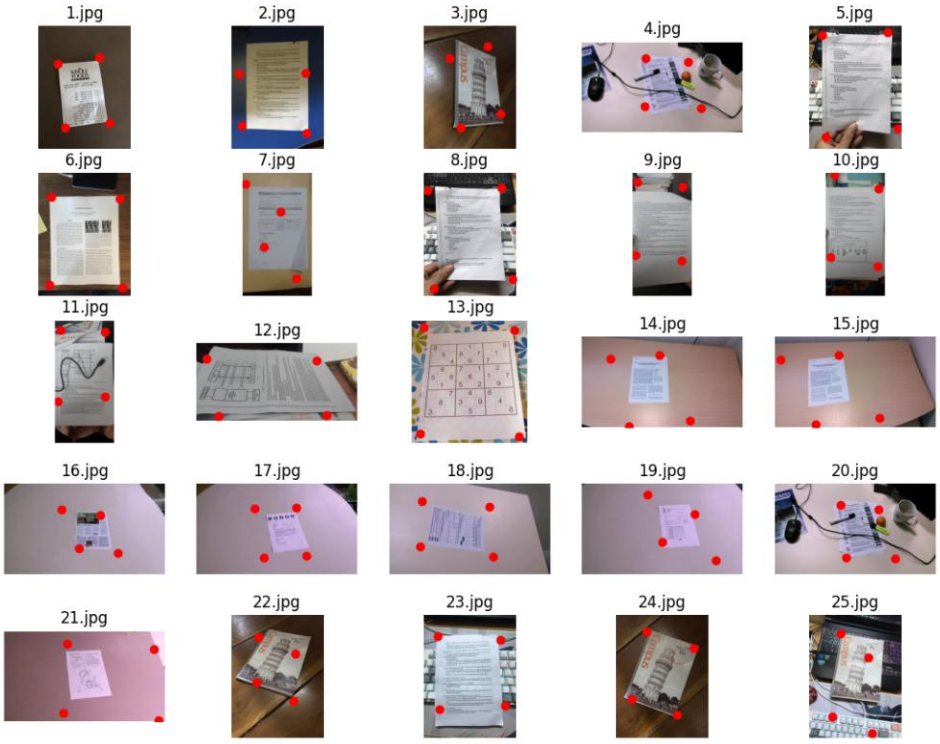



## Thí nghiệm 2:

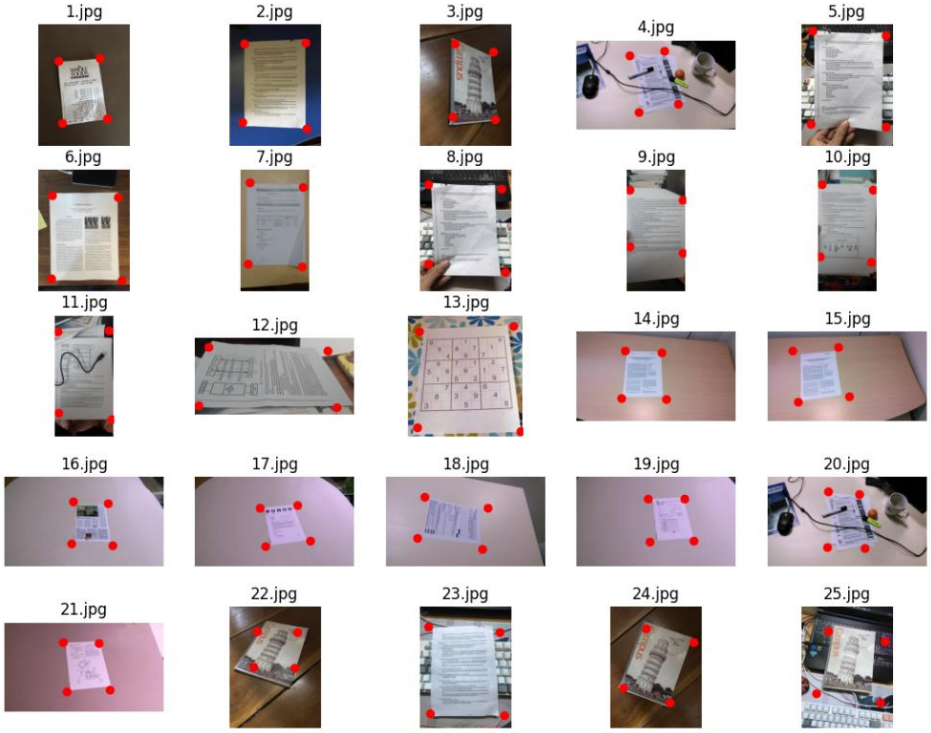


**Thí nghiệm 4:**

Khi không sử dụng transfer learning:

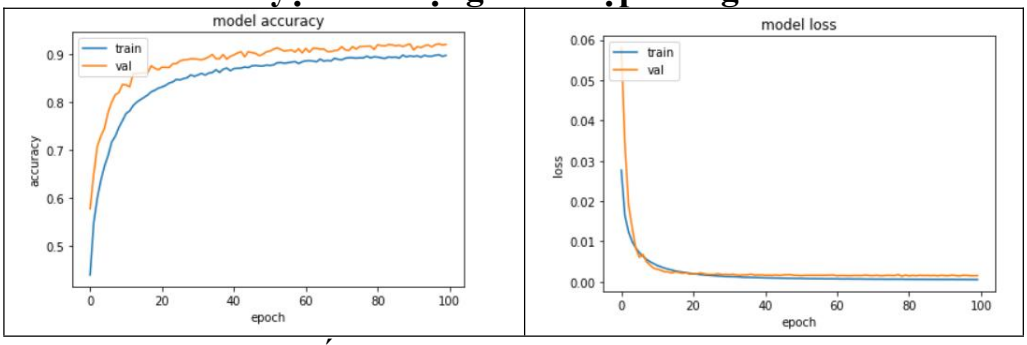



Khi sử dụng transfer learning:



Dựa trên kết quả của 4 thí nghiệm, kết quả của thí nghiệm 4 với transfer learning đạt độ chính xác lớn nhất trên tập dữ liệu thực tế

### 3. 2. 2. Huấn luyện với mạng tích chập đơn giản



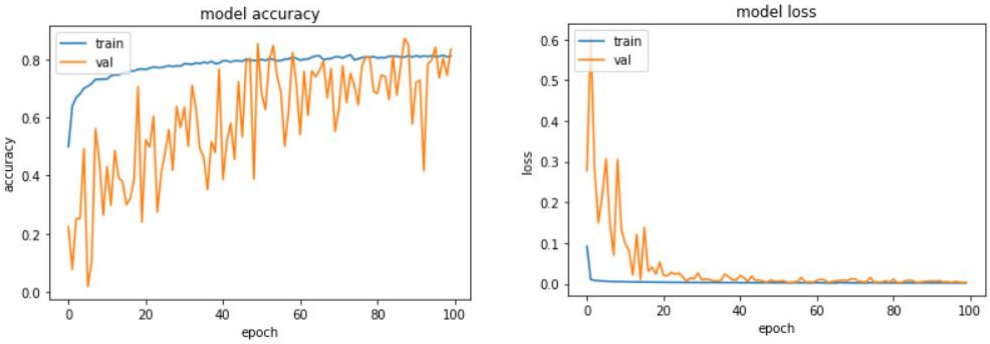

Mô hình được huấn luyện tương tự thí nghiệm 4 với mạng MobileNet v2 và thu được độ chính xác trên tập huấn luyện là 90%

### 3. 2. 3. Huấn luyện với mạng tích chập

Thay thế MobileNetv2 bằng mạng nơ-ron tích chập trên để huấn luyện.

Với mạng tích chập này chỉ huấn luyện với tập dữ liệu đã được trộn từ 2 tập dữ liệu ban đầu. Huấn luyện với 100 epochs, hàm tối ưu adam với lr = 0.0001 và hàm mất mát MSE

Kết quả:



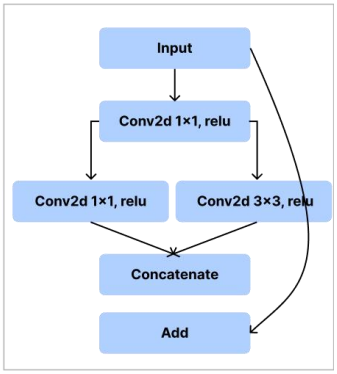

Mô hình đạt độ chính xác 81% trên tập huấn luyện sau 100 epochs

Qua kết quả, mô hình được huấn luyện với mạng tích chập này xảy ra hiện tượng overfitting ở 40 epochs đầu, và có xu hướng khớp ở các epochs sau

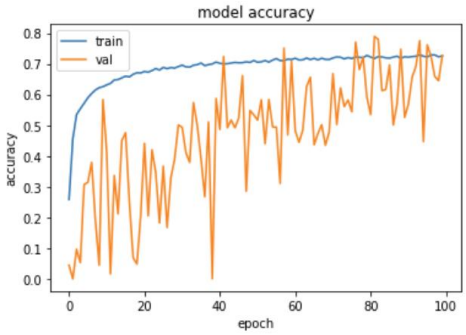
Khi em sử dụng dropout = 0.4, mô hình đạt độ chính xác 77% và hiện tượng overfitting chỉ xảy ra ở 20 epochs đầu.

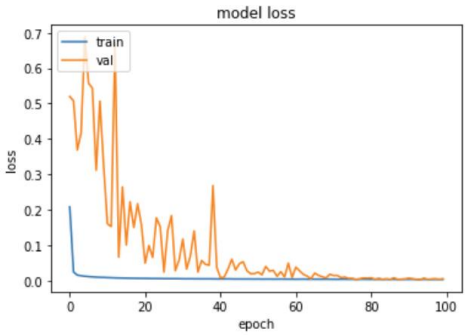
**Áp dụng residual block của mạng ResNet vào mạng tích chập đang huấn luyện:**

Khi thay đổi cấu trúc mạng bằng cách áp dụng residual block với các khối block trong mạng tích chập này thì hiện tượng overfitting không được cải thiện thậm chí còn tăng lên. Độ chính xác giảm từ 81% -> 71%. Nhưng kích thước của mô hình giảm.



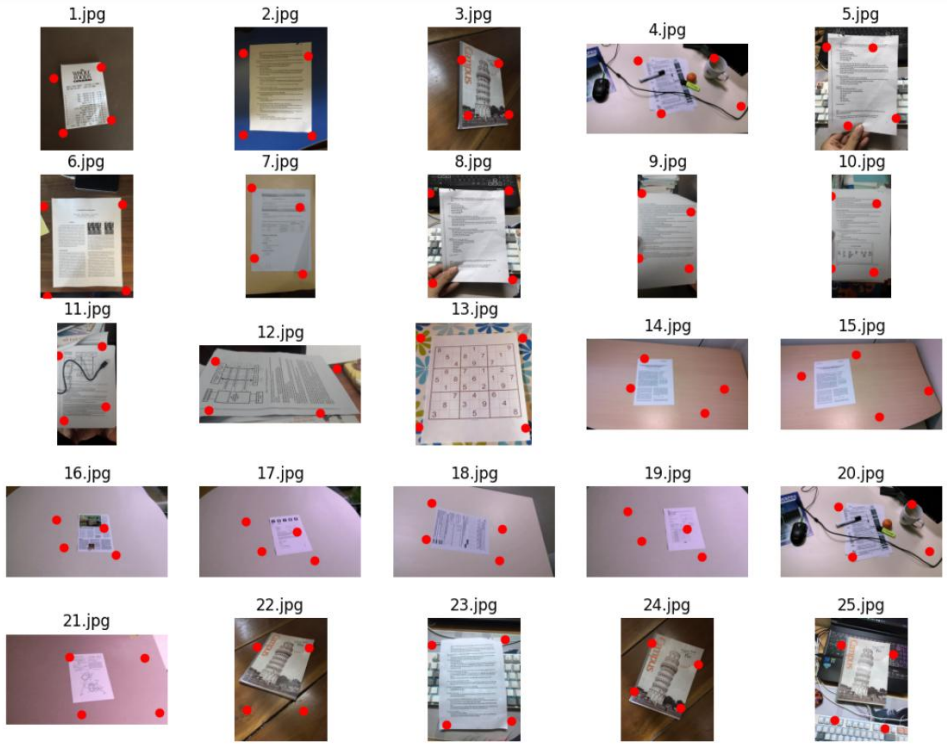

Block



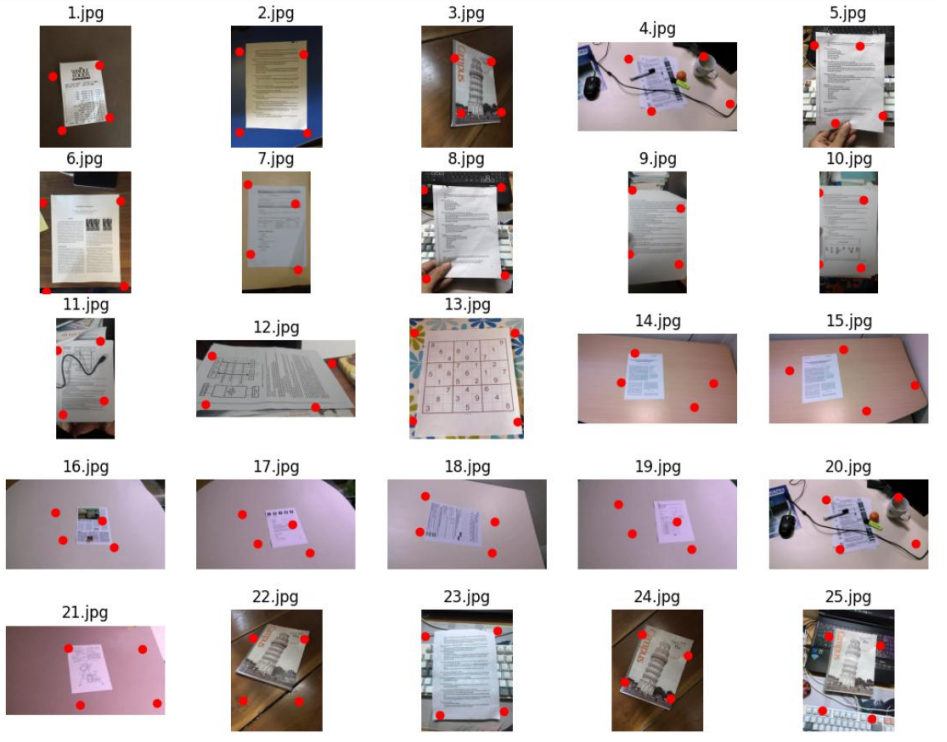




**Kết quả dự đoán:**

Không áp dụng Residual Block:



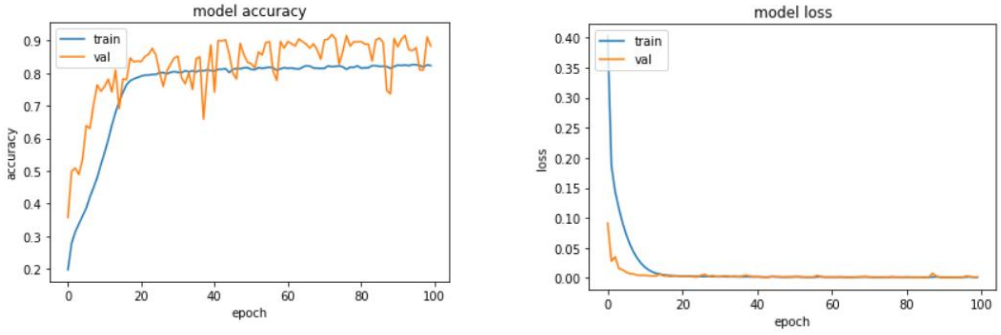

Áp dụng Residual Block:



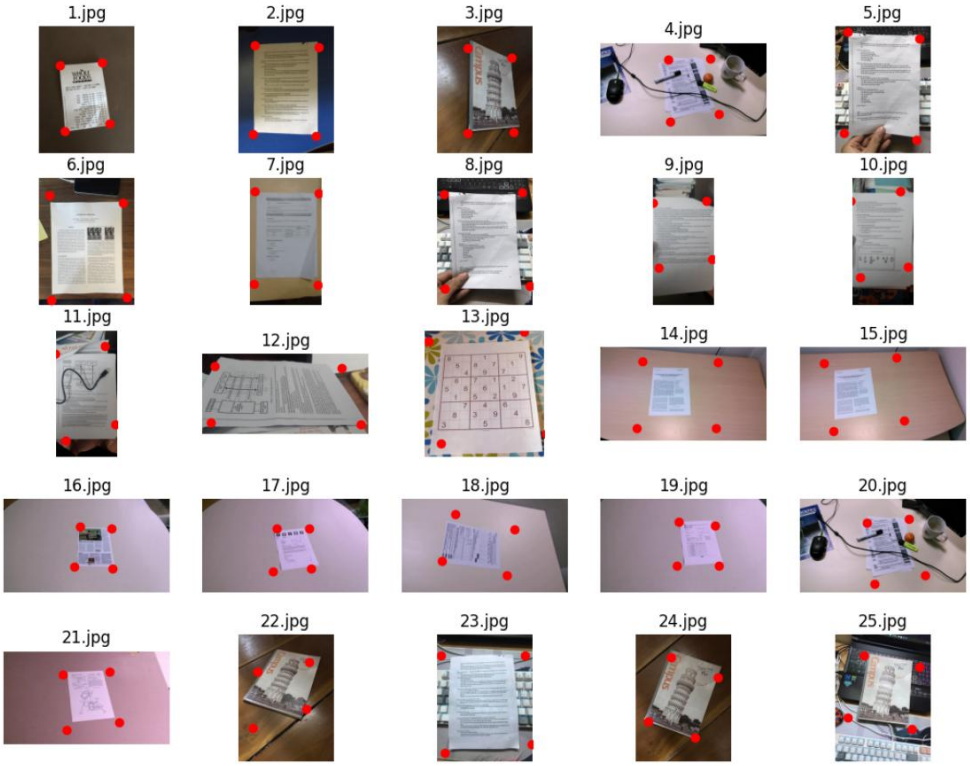

Mạng tích chập này luôn xảy ra hiện tượng overfitting khi huấn luyện trên tập dữ liệu trên. Với kết quả kiểm tra trên tập dữ liệu thực tế, mô hình trả ra kết quả không như mong đợi. Để cải thiện vấn đề, em áp dụng transfer learning với pre-trained model được huấn luyện trên imagenet để tiếp tục huấn luyện.

**Transfer learning:**

Khi áp dụng transfer learning vào mạng tích chập này, hiện tượng overfitting đã không còn xảy ra, độ chính xác đạt 82%




**Kết quả dự đoán:**




Độ chính xác đã được cải thiện, kết quả trên tập dữ liệu thực tế là chấp nhận được.

### 3. 2. 4. Đánh giá

Bảng so sánh:

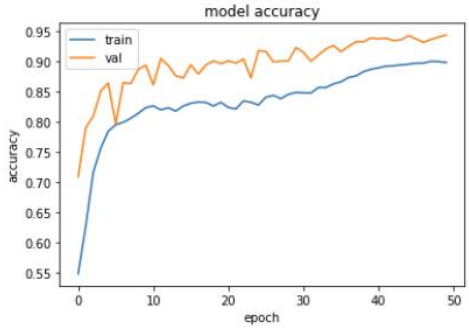
| **Mô hình** | MobileNetv2 | | Mạng tích chập v1 | Mạng tích chập v2 | | Mạng tích chập v2.2 | |
|---|---|---|---|---|---|---|---|
| **Số param** | *2,983,112* | | *3,433,056* | *1,054,408* | | *1,054,408* | |
| **Kích thước** | 35,4MB | | 38,6MB | 12,6MB | | 10,6MB | |
| **Thời gian huấn luyện** | 3720s | | 1809s | 2208s | | ~ 2208 s | |
| **Độ chính xác** | 80 % | 88 % | 90% | 81% | 82 % | 71% | |
| **Độ chính xác thực tế** | 50 % | 76 % | 40% | 35% | 55 % | 38% | |
| **Tốc độ dự đoán** | 105ms | | 120ms | 87ms | | | |
| **Transfer learning** | Không | Có | Không | Không | Có | Không | |

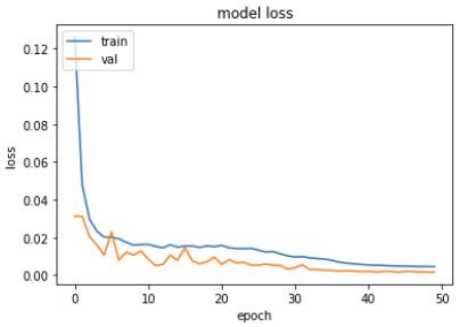
### 3. 3. Huấn luyện mô hình 2:

Em sử dụng kiến trúc trên để huấn luyện

Với mô hình 2, dataset được tạo bằng cách cắt riêng từng góc của dataset dùng cho việc huấn luyện mô hình 1. Bởi vậy, việc thí nghiệm độc lập cho từng kiểu dữ liệu là không cần thiết cho mô hình 2 vì kết quả của thí nghiệm giống với các thí nghiệm tương ứng khi huấn luyện mô hình 1.

Với việc huấn luyện mô hình 2, em thực hiện duy nhất 1 thí nghiệm trên tập dữ liệu gồm 100 000 ảnh với kích thước 64x64 với 50 vòng lặp và Adam(0.0001).




## Kết hợp 2 mô hình:

## Hàm đệ quy khi thực hiện dự đoán góc:

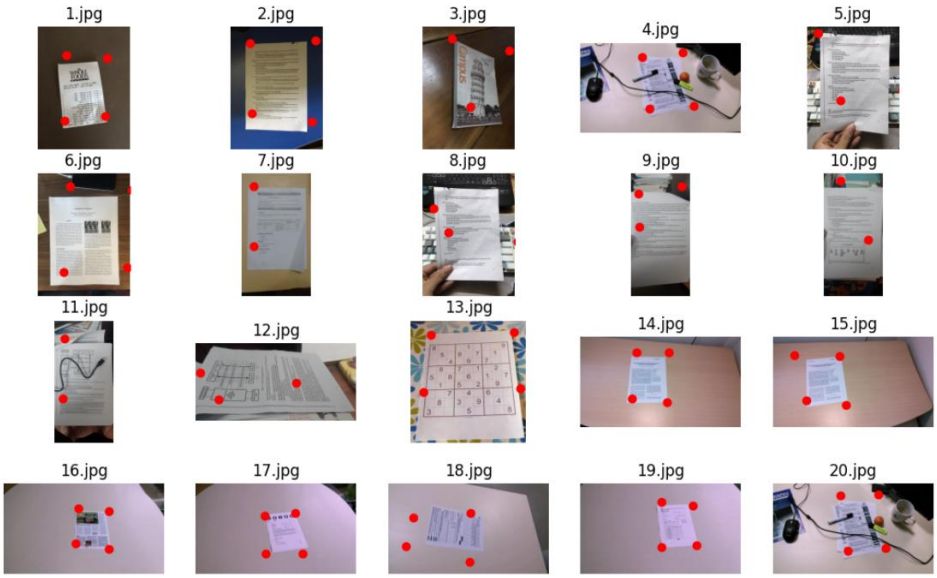
```
while (width > 10 and height > 10):
  (pre_x, pre_y) = corner_model.predict(oImg)[0] * 64

  width = int(width * factor)
  height = int(height * factor)

  left = max(int(pre_x - width/2), left)
  top = max(int(pre_y - height/2), top)
  right = min(int(pre_x + width/2), right)
  bottom = min(int(pre_y + height/2), bottom)

  width = right - left
  height = bottom - top

  oImg = oImg[:, top:bottom, left:right]
  oImg = cv2.resize(img_resize[top:bottom, left:right], (64, 64))
  oImg = oImg.reshape(-1, 64, 64, 3).astype('float32') / 255
  # oImg = np.pad(oImg, ((0, 0), (top, 32 - bottom), (left, 32 - right), (0, 0)), 'constant')

  res = (pre_x/64, pre_y/64)
return res
```

**Kết hợp 2 mô hình:**

Dựa trên kết quả của mô hình 1, phân ảnh ra thành 4 vùng và thực hiện tìm góc chính xác theo thuật toán đã nêu trên

```
tlx_trx = int((tlx + trx)/2)
tly_try = int((tly + Try)/2)

tlx_blx = int((tlx + blx)/2)
tly_bly = int((tly + bly)/2)

trx_brx = int((trx + brx)/2)
try_bry = int((Try + bry)/2)

blx_brx = int((blx + brx)/2)
bly_bry = int((bly + bry)/2)


tl_img = img_resize[max(2 * int(tly) - tly_bly, 0) : tly_bly,  max(2 * int(tlx) - tlx_trx, 0) : tlx_trx]
top_left = detect_corner(
    tl_img, factor
)

tr_img = img_resize[max((2 * int(Try) - try_bry), 0) : try_bry,  tlx_trx : min((2 * int(trx) - tlx_trx), width)]
top_right = detect_corner(
    tr_img, factor
)

br_img = img_resize[try_bry : min((2 * int(bry) - tly_bly), height),  blx_brx : min((2 * int(brx) - blx_brx), width)]
bottom_right = detect_corner(
    br_img, factor
)

bl_img = img_resize[tly_bly : min((2 * int(bly) - tly_bly), height),  max((2 * int(blx) - blx_brx), 0) : blx_brx]
bottom_left = detect_corner(
    bl_img, factor
)
```

**Kết quả khi kết hợp 2 mô hình:**




Kết quả dự đoán phụ thuộc vào chất lượng của mô hình 2. Dựa trên kết quả thu được từ tập dữ liệu kiểm tra thì việc sử dụng mô hình 2 trong bài toán không mang lại hiệu quả, thậm chí còn làm giảm độ chính xác của mô hình ban đầu.

4. **Kết hợp các thành phần thành mô-đun hoàn chỉnh:**