# Phân loại hình ảnh CIFAR-10 trong TensorFlow

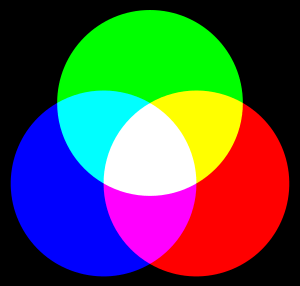
## Một số lý thuyết cơ bản

### Giới thiệu về xử lý ảnh

#### Ảnh trong máy tính

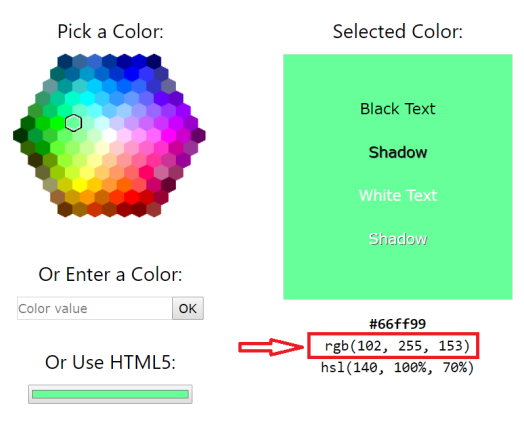
##### Hệ màu RGB

RGB viết tắt của red (đỏ), green (xanh lục), blue (xanh lam), là ba màu chính của ánh sáng khi tách ra từ lăng kính. Khi trộn ba màu trên theo tỉ lệ nhất định có thể tạo thành các màu khác nhau.



*Thêm đỏ vào xanh lá cây tạo ra vàng; thêm vàng vào xanh lam tạo ra trắng. Nguồn*[*wiki*](https://vi.wikipedia.org/wiki/M%C3%B4_h%C3%ACnh_m%C3%A0u_RGB#/media/File:AdditiveColorMixiing.svg)*.*

Ví dụ khi bạn chọn màu ở [đây](https://www.w3schools.com/colors/colors_picker.asp). Khi bạn chọn một màu thì sẽ ra một bộ ba số tương ứng **(r,g,b)**



*màu được chọn là rgb(102, 255, 153), nghĩa là r=102, g=255, b=153.*

Với mỗi bộ 3 số r, g, b nguyên trong khoảng [0, 255] sẽ cho ra một màu khác nhau. Do có 256 cách chọn r, 256 cách chọn màu g, 256 cách chọn b => tổng số màu có thể tạo ra bằng hệ màu RGB là: 256 \* 256 \* 256 = 16777216 màu !!!

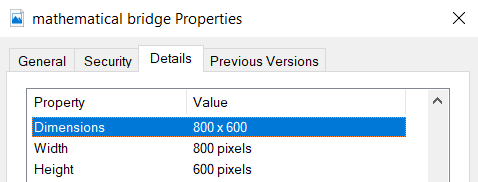
##### Ảnh màu

Ví dụ về ảnh màu

**

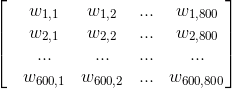
*Mathematical bridge, Cambridge*

Khi bạn kích chuột phải vào ảnh trong máy tính, bạn chọn properties (mục cuối cùng), rồi chọn tab Detail



Bạn sẽ thấy chiều dài ảnh là 800 pixels (viết tắt px), chiều rộng 600 pixels, kích thước là 800 \* 600. Theo [wiki](https://vi.wikipedia.org/wiki/Pixel), pixel (hay điểm ảnh) là một khối màu rất nhỏ và là đơn vị cơ bản nhất để tạo nên một bức ảnh kỹ thuật số.

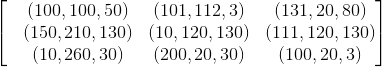
Vậy bức ảnh trên kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một [ma trận](https://nttuan8.com/bai-1-linear-regression-va-gradient-descent/#Ma_tran) kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột).



Trong đó mỗi phần tử *wij*​ là một pixel.

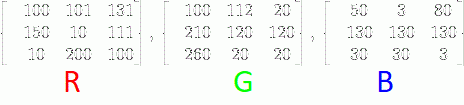
Như vậy có thể hiểu là mỗi pixel thì biểu diễn một màu và bức ảnh trên là sự kết hợp rất nhiều pixel. Hiểu đơn giản thì in bức ảnh ra, kẻ ô vuông như chơi cờ ca rô với 800 đường thẳng ở chiều dài, 600 đường ở chiều rộng, thì mỗi ô vuông là một pixel, biểu diễn một chấm màu.

Tuy nhiên để biểu diễn 1 màu ta cần 3 thông số (r,g,b) nên gọi *wij*​=(*rij*​,*gij*​,*bij*​) để biểu diễn dưới dạng ma trận thì sẽ như sau:



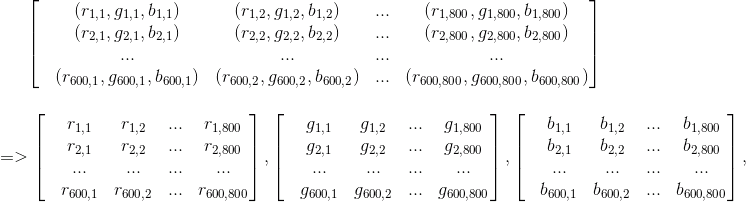
*Ảnh màu kích thước 3\*3 biểu diễn dạng ma trận, mỗi pixel biểu diễn giá trị (r,g,b)*

Để tiện lưu trữ và xử lý không thể lưu trong 1 ma trận như thế kia mà sẽ tách mỗi giá trị trong mỗi pixel ra một ma trận riêng.



*Tách ma trận trên thành 3 ma trận cùng kích thước: mỗi ma trận lưu giá trị từng màu khác nhau red, green, blue*

Tổng quát

*Tách ma trận biểu diễn màu ra 3 ma trận, mỗi ma trận lưu giá trị 1 màu.*

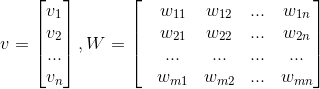
Mỗi ma trận được tách ra được gọi là 1 channel nên ảnh màu được gọi là 3 channel: channel red, channel green, channel blue.

**Tóm tắt**: Ảnh màu là một ma trận các pixel mà mỗi pixel biểu diễn một điểm màu. Mỗi điểm màu được biểu diễn bằng bộ 3 số (r,g,b). Để tiện cho việc xử lý ảnh thì sẽ tách ma trận pixel ra 3 channel red, green, blue.

##### Tensor là gì

Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột.

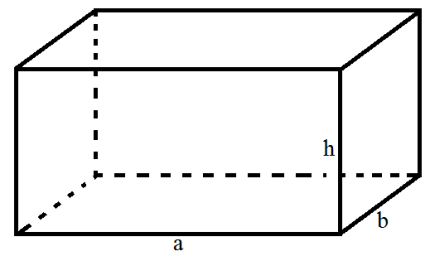
Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng \* số cột.



*Vector v kích thước n, ma trận W kích thước m\*n*

Khi dữ liệu nhiều hơn 2 nhiều thì sẽ được gọi là tensor, ví dụ như dữ liệu có 3 chiều.

Để ý thì thấy là ma trận là sự kết hợp của các vector cùng kích thước. Xếp n vector kích thước m cạnh nhau thì sẽ được ma trận m\*n. Thì tensor 3 chiều cũng là sự kết hợp của các ma trận cùng kích thước, xếp k ma trận kích thước m\*n lên nhau sẽ được tensor kích thước m\*n\*k.

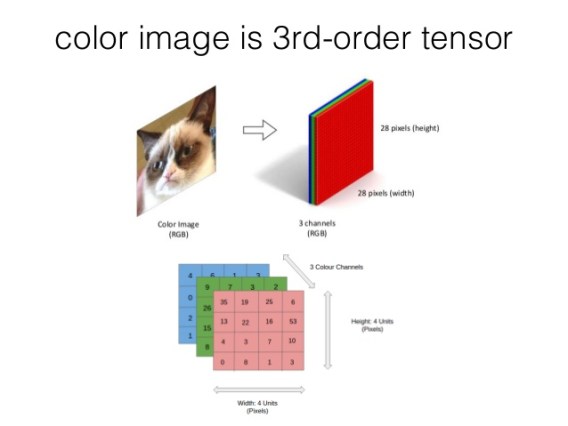


*Hình hộp chữ nhật kích thước a\*b\*h*

Tưởng tượng mặt đáy là một ma trận kích thước a \* b, được tạo bởi b vector kích thước a. Cả hình hộp là tensor 3 chiều kích thước a\*b\*h, được tạo bởi xếp h ma trận kích thước a\*b lên nhau.

Do đó biểu diễn ảnh màu trên máy tính ở phần trên sẽ được biểu diễn dưới dạng tensor 3 chiều kích thước 600\*800\*3 do có 3 ma trận (channel) màu red, green, blue kích thước 600\*800 chồng lên nhau.

Ví dụ biểu diễn ảnh màu kích thước 28\*28, biểu diễn dưới dạng tensor 28\*28\*3



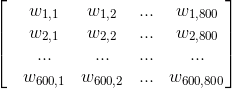
*Nguồn: https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors*

##### Ảnh xám

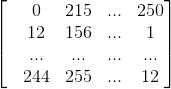


*Ảnh xám của mathematical bridge*

Tương tự ảnh màu, ảnh xám cũng có kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một [ma trận](https://nttuan8.com/bai-1-linear-regression-va-gradient-descent/#Ma_tran) kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột).



Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ [0,255] thay vì (r,g,b) như trong ảnh màu. Do đó khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ.



*Biểu diễn ảnh xám*

Giá trị 0 là màu đen, 255 là màu trắng và giá trị pixel càng gần 0 thì càng tối và càng gần 255 thì càng sáng.

##### Chuyển hệ màu của ảnh

Mỗi pixel trong ảnh màu được biểu diễn bằng 3 giá trị (r,g,b) còn trong ảnh xám chỉ cần 1 giá trị x để biểu diễn.

Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng [công thức](https://pillow.readthedocs.io/en/3.2.x/reference/Image.html#PIL.Image.Image.convert):

x = r \* 0.299 + g \* 0.587 + b \* 0.114.

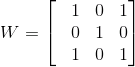
Tuy nhiên khi chuyển ngược lại, bạn chỉ biết giá trị x và cần đi tìm r,g,b nên sẽ không chính xác.

#### Phép tính convolution

##### Convolution

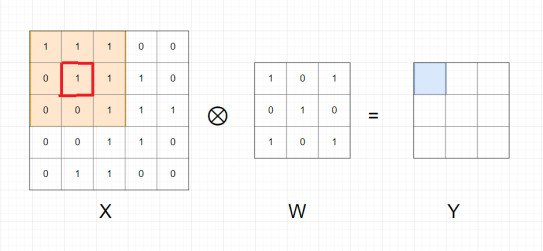
Để cho dễ hình dung ta sẽ lấy ví dụ trên ảnh xám, tức là ảnh được biểu diễn dưới dạng ma trận X kích thước m\*n.

Ta định nghĩa **kernel**là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9,… Ví dụ kernel kích thước 3\*3



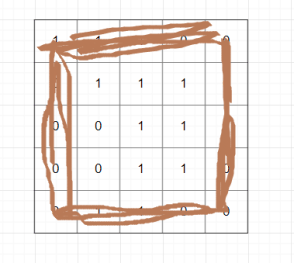
Kí hiệu phép tính convolution (⊗), kí hiệu *Y*=*X*⊗*W*

Với mỗi phần tử *xij​* trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử *xij*​ làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính [element-wise](https://nttuan8.com/bai-1-linear-regression-va-gradient-descent/#Element-wise_multiplication_matrix) của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

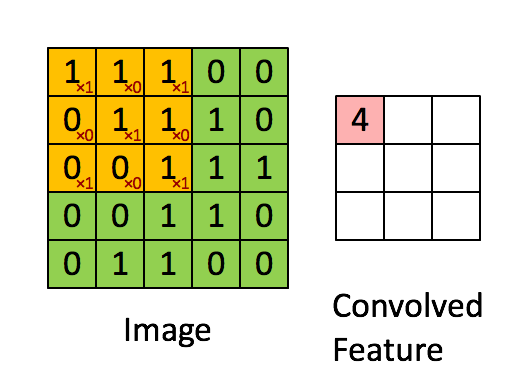


Ví dụ khi tính tại *x*22​ (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có *x*22​ làm trung tâm có màu nền da cam như trong hình. Sau đó tính *y*11​ = *sum*(*A*⊗*W*)=*x*11​∗*w*11​ + *x*12​∗*w*12​ + *x*13​∗*w*13​ + *x*21​∗*w*21 ​+ *x*22​∗*w*22​ + *x*23​∗*w*23​ + *x*31​∗*w*31​ + *x*32​∗*w*32​ + *x*33​∗*w*33​ = 4. Và làm tương tự với các phần tử còn lại trong ma trận.

Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như *x*11​? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X.



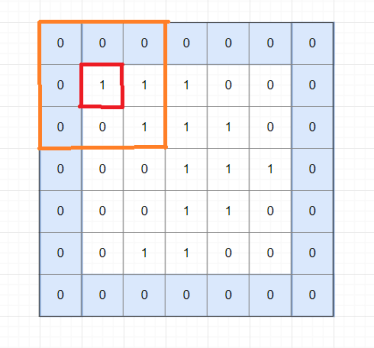
Nên bạn để ý thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là (m-k+1) \* (n-k+1).



*Các bước thực hiện phép tính convolution cho ma trận X với kernel K ở trên*

##### Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X => Tìm cách giải quyết cho các phần tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.



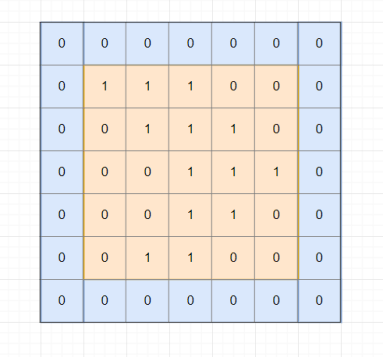
*Ma trận X khi thêm viền 0 bên ngoài*

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử *x*11​ , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu.

Phép tính này gọi là convolution với **padding=1**. Padding=k nghĩa là thêm k vector 0 vào mỗi phía của ma trận.

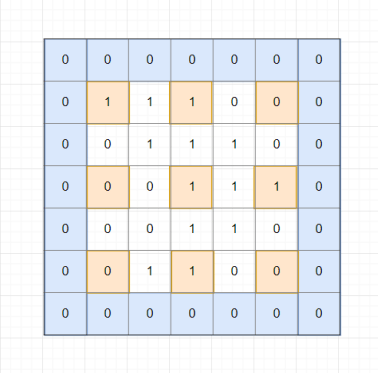
##### Stride

Như ở trên ta thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.



*stride=1, padding=1*

Tuy nhiên nếu **stride=k** (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử *x*1+*i*∗*k* , 1+*j*∗*k*​. Ví dụ k = 2.

**

*padding=1, stride=2*

Hiểu đơn giản là bắt đầu từ vị trí *x*11​ sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X.

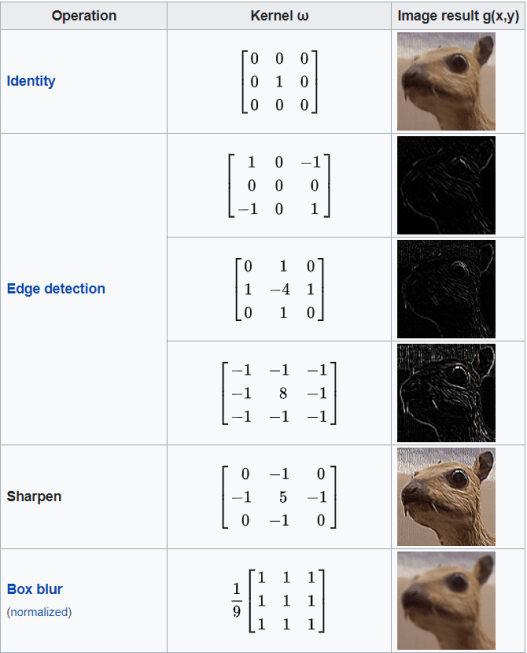
Kích thước của ma trận Y là 3\*3 đã giảm đi đáng kể so với ma trận X.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước m\*n với kernel kích thước k\*k, stride = s, padding = p ra ma trận Y kích thước

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

##### Ý nghĩa của phép tính convolution

Mục đích của phép tính convolution trên ảnh là làm mở, làm nét ảnh; xác định các đường;… Mỗi kernel khác nhau thì sẽ phép tính convolution sẽ có ý nghĩa khác nhau. Ví dụ:



*Nguồn: https://en.wikipedia.org/wiki/Kernel\_(image\_processing)*

### Giới thiệu về mạng CNN

**Convolutional Neural Network** (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến.

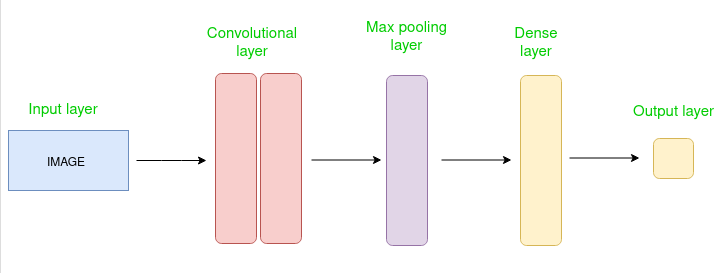
Trước khi đi sâu vào Mạng nơ-ron tích chập, chúng ta hãy xem lại một số khái niệm về Mạng nơ-ron. Trong Mạng nơ-ron thông thường, có ba loại lớp:

1. **Lớp đầu vào:** Đó là lớp mà chúng ta cung cấp đầu vào cho mô hình của mình. Số lượng tế bào thần kinh trong lớp này bằng với tổng số đặc trưng trong dữ liệu của chúng ta (số pixel trong trường hợp hình ảnh).
2. **Lớp ẩn:** Đầu vào từ lớp Đầu vào sau đó được đưa vào lớp ẩn. Có thể có nhiều lớp ẩn tùy thuộc vào mô hình và kích thước dữ liệu của chúng ta. Mỗi lớp ẩn có thể có số lượng tế bào thần kinh khác nhau, thường lớn hơn số lượng đặc trưng. Đầu ra từ mỗi lớp được tính bằng phép nhân ma trận của đầu ra của lớp trước đó với các trọng số có thể học được của lớp đó và sau đó bằng phép cộng các độ lệch có thể học được, sau đó là hàm kích hoạt làm cho mạng trở nên phi tuyến tính.
3. **Lớp đầu ra:** Đầu ra từ lớp ẩn sau đó được đưa vào một hàm logistic như sigmoid hoặc softmax để chuyển đổi đầu ra của mỗi lớp thành điểm xác suất của mỗi lớp.

Dữ liệu sau đó được đưa vào mô hình và đầu ra từ mỗi lớp được lấy. Bước này được gọi là **feedforward(lan truyền thẳng)**, sau đó chúng ta tính toán lỗi bằng cách sử dụng hàm mất mát, một số hàm mất mát phổ biến là cross-entropy, lỗi mất bình phương, v.v. Hàm mất mát đo lường mức độ hoạt động của mạng. Sau đó, chúng tôi lan truyền ngược vào mô hình bằng cách tính đạo hàm. Bước này được gọi là **Backpropagation(lan truyền ngược)**, về cơ bản được sử dụng để giảm thiểu tổn thất.

#### Kiến trúc CNN

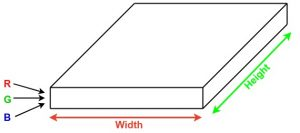
Mạng nơ-ron tích chập bao gồm nhiều lớp như lớp đầu vào(input layer), lớp tích chập(Convolutional layer), lớp tổng hợp(Pooling layer) và các lớp được kết nối đầy đủ(fully connected layers).



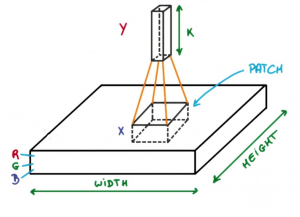
Lớp Convolutional áp dụng các bộ lọc cho hình ảnh đầu vào để trích xuất các đặc trưng, lớp Pooling lấy mẫu xuống hình ảnh để giảm tính toán và lớp fully connected đưa ra dự đoán cuối cùng. Mạng học các bộ lọc tối ưu thông qua lan truyền ngược và giảm dần độ dốc(**gradient descent**).

#### Cách thức hoạt động của mạng thần kinh tích chập(CNN)

Convolution Neural Networks hoặc covnets là các mạng nơ-ron chia sẻ các tham số của chúng. Hãy tưởng tượng bạn có một hình ảnh. Nó có thể được biểu diễn dưới dạng một hình khối có chiều dài, chiều rộng (kích thước của hình ảnh) và chiều cao (vì hình ảnh thường có các kênh màu đỏ, lục và lam).



Bây giờ hãy tưởng tượng lấy một mảng nhỏ của hình ảnh này và chạy một mạng thần kinh nhỏ trên đó, chẳng hạn như k đầu ra và biểu diễn chúng theo chiều dọc. Bây giờ hãy trượt mạng thần kinh đó trên toàn bộ hình ảnh, kết quả là chúng ta sẽ có được một hình ảnh khác với chiều rộng, chiều dài và chiều cao khác nhau. Thay vì chỉ có các kênh R, G và B, giờ đây chúng tôi có nhiều kênh hơn nhưng chiều rộng và chiều dài nhỏ hơn. Hoạt động này được gọi là Convolution. Nếu kích thước bản vá(**patch**) giống như kích thước của hình ảnh thì đó sẽ là một mạng thần kinh thông thường. Vì miếng vá nhỏ này, chúng tôi có ít trọng số hơn.



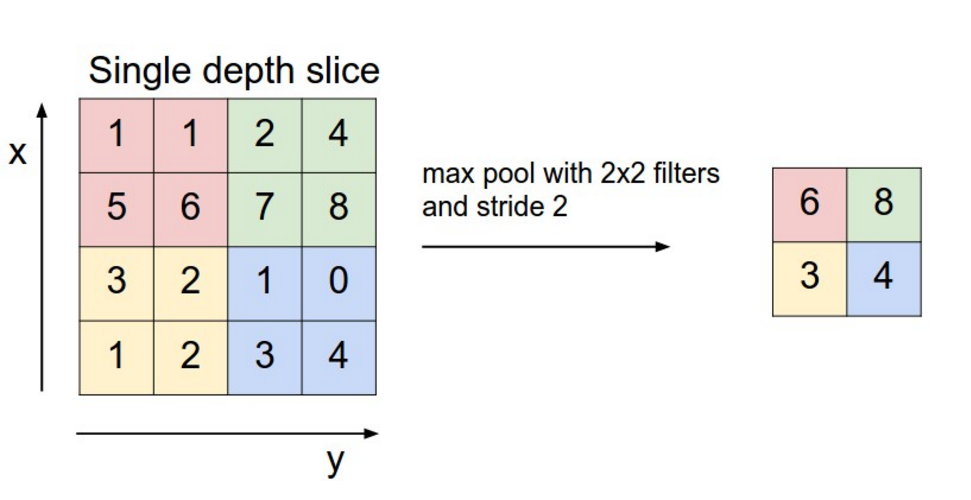
Bây giờ hãy nói về một chút toán học liên quan đến toàn bộ quá trình tích chập.

* Các lớp tích chập bao gồm một tập hợp các bộ lọc(**filters**) có thể học được (hoặc **kernels**(hạt nhân)) có chiều rộng và chiều dài nhỏ và cùng chiều cao với khối lượng đầu vào(3 nếu lớp đầu vào là đầu vào hình ảnh).
* Ví dụ: nếu chúng ta phải chạy tích chập trên một hình ảnh có kích thước 34x34x3. Kích thước có thể có của bộ lọc có thể là axax3, trong đó 'a' có thể là bất kỳ thứ gì như 3, 5 hoặc 7 nhưng nhỏ hơn so với kích thước hình ảnh.
* Trong quá trình chuyển tiếp, chúng tôi trượt từng bộ lọc qua toàn bộ khối lượng đầu vào theo từng bước trong đó mỗi bước được gọi là  [***stride***](https://www.geeksforgeeks.org/ml-introduction-to-strided-convolutions/)***(***[***sải chân***](https://www.geeksforgeeks.org/ml-introduction-to-strided-convolutions/)***)***(có thể có giá trị 2, 3 hoặc thậm chí 4 đối với chiều dài hình ảnh) và tính tích chấm giữa trọng số hạt nhân và bản vá từ khối lượng đầu vào.
* Khi chúng ta trượt các bộ lọc của mình, chúng tôi sẽ nhận được đầu ra 2-D cho mỗi bộ lọc và kết quả là chúng ta sẽ xếp chúng lại với nhau, chúng ta sẽ nhận được khối lượng đầu ra có độ sâu bằng với số lượng bộ lọc. Mạng sẽ học tất cả các bộ lọc.

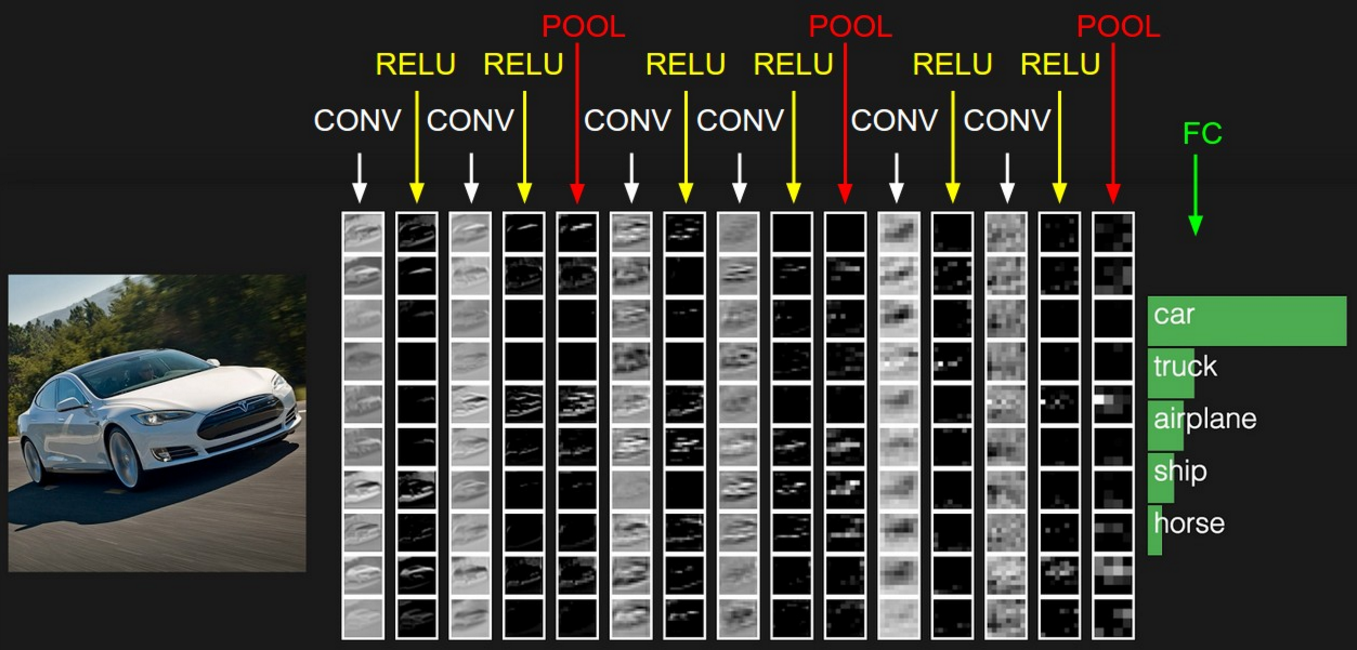
#### Các lớp được sử dụng để xây dựng ConvNets

Một kiến ​​trúc Convolution Neural Networks hoàn chỉnh còn được gọi là **covnets**. Một covnets là một chuỗi các lớp và mỗi lớp biến đổi tập này sang tập khác thông qua một hàm khả vi(differentiable function).   
**Các loại lớp:**  bộ dữ liệu  
Hãy lấy một ví dụ bằng cách chạy một covnets trên hình ảnh có kích thước 32x32x3.

* **Lớp đầu vào(Input Layers):** Đó là lớp mà chúng ta cung cấp đầu vào cho mô hình của mình. Trong CNN, Thông thường, đầu vào sẽ là một hình ảnh hoặc một chuỗi hình ảnh. Lớp này chứa đầu vào thô của hình ảnh với chiều rộng 32, chiều dài 32 và chiều cao 3.
* **Lớp tích chập(Convolutional Layers):** Đây là lớp được sử dụng để trích xuất đặc trưng từ tập dữ liệu đầu vào. Nó áp dụng một tập hợp các bộ lọc có thể học được gọi là hạt nhân(**kernels**) cho các hình ảnh đầu vào. Các bộ lọc/hạt nhân là các ma trận nhỏ hơn thường có hình dạng 2×2, 3×3 hoặc 5×5. nó trượt trên dữ liệu hình ảnh đầu vào và tính toán tích vô hướng giữa trọng số hạt nhân và bản vá(**patch**) hình ảnh đầu vào tương ứng. Đầu ra của lớp này được gọi là bản đồ đặc trưng. Giả sử chúng ta sử dụng tổng cộng 12 bộ lọc cho lớp này, chúng ta sẽ nhận được khối lượng đầu ra có kích thước 32x32x12.
* [**Lớp kích hoạt(**[**Activation Layer**](https://www.geeksforgeeks.org/activation-functions-neural-networks/)**):**](https://www.geeksforgeeks.org/activation-functions-neural-networks/) Bằng cách thêm một hàm kích hoạt vào đầu ra của lớp trước đó, các lớp kích hoạt sẽ bổ sung tính phi tuyến tính cho mạng. nó sẽ áp dụng hàm kích hoạt theo từng phần tử cho đầu ra của lớp tích chập. Một số hàm kích hoạt phổ biến là**RELU** : max(0, x),  **Tanh** ,**Leaky RELU** , v.v. Khối lượng không thay đổi do đó khối lượng đầu ra sẽ có kích thước 32x32x12.
* [**Lớp tổng hợp(****[Pooling layer](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/)):**](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/) Lớp này được chèn định kỳ vào covnets và chức năng chính của nó là giảm kích thước của ổ đĩa, giúp tính toán nhanh, giảm bộ nhớ và cũng ngăn ngừa overfitting. Hai loại lớp tổng hợp phổ biến là**tổng hợp tối đa(max pooling)** và**tổng hợp trung bình(average pooling)** . Nếu chúng ta sử dụng **tổng hợp tối đa** với bộ lọc 2x2 và stride là 2, khối lượng kết quả sẽ có kích thước 16x16x12.

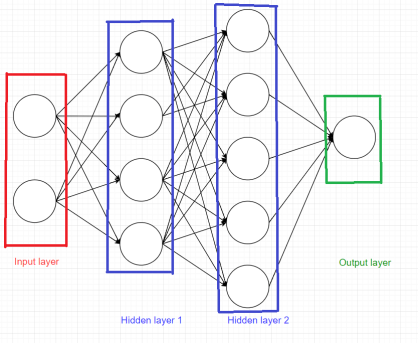


* **Làm phẳng(Flattening):** Các bản đồ đặc trưng kết quả được làm phẳng thành một vectơ một chiều sau các lớp tích chập và tổng hợp để chúng có thể được chuyển vào một lớp liên kết hoàn toàn để phân loại hoặc hồi quy.
* **Fully Connected Layers:** Nó lấy đầu vào từ lớp trước đó và tính toán nhiệm vụ phân loại hoặc hồi quy cuối cùng.



* **Lớp đầu ra(Output Layer):** Đầu ra từ các lớp được kết nối đầy đủ sau đó được đưa vào hàm hậu cần(logistic function) cho các tác vụ phân loại như sigmoid hoặc softmax để chuyển đổi đầu ra của mỗi lớp thành xác suất của mỗi lớp.

### Convolutional layer

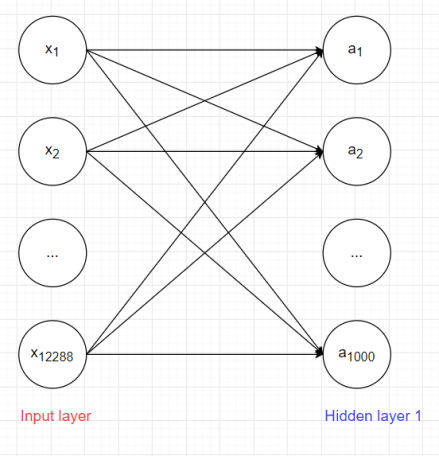


*Mô hình neural network.*

Mỗi hidden layer được gọi là **fully connected layer**, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là **fully connected neural network (FCN)**.

#### Vấn đề của fully connected neural network với xử lý ảnh

Như phần trên có nói về xử lý ảnh, thì ảnh màu 64\*64 được biểu diễn dưới dạng 1 tensor 64\*64\*3. Nên để biểu thị hết nội dung của bức ảnh thì cần truyền vào input layer tất cả các pixel (64\*64\*3 = 12288). Nghĩa là input layer giờ có 12288 nodes.



*Input layer và hidden layer 1*

Giả sử số lượng node trong hidden layer 1 là 1000. Số lượng weight W giữa input layer và hidden layer 1 là 12288\*1000 = 12288000, số lượng bias là 1000 => tổng số parameter là: 12289000. Đấy mới chỉ là số parameter giữa input layer và hidden layer 1, trong model còn nhiều layer nữa, và nếu kích thước ảnh tăng, ví dụ 512\*512 thì số lượng parameter tăng cực kì nhanh => Cần giải pháp tốt hơn !!!

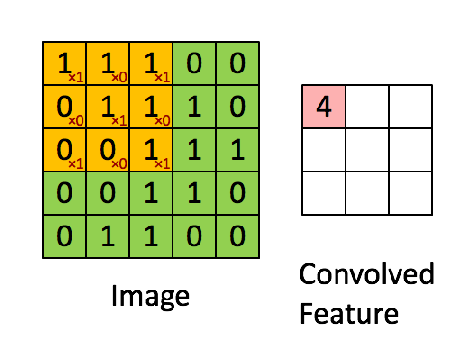
Nhận xét:

* Trong ảnh các pixel ở cạnh nhau thường có **liên kết với nhau hơn** là những pixel ở xa. Ví dụ như phép tính [convolution](https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/#Convolution) trên ảnh ở phần trên. Để tìm các đường trong ảnh, ta áp dụng sobel kernel trên mỗi vùng kích thước 3\*3. Hay làm nét ảnh ta áp dụng sharpen kernel cũng trên vùng có kích thước 3\*3.
* Trong phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh **chia sẻ** hệ số với nhau.

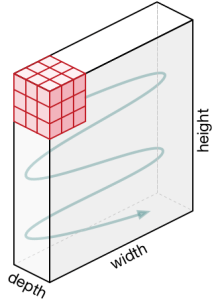
=> Áp dụng **phép tính convolution** vào layer trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.

#### Convolutional layer đầu tiên

Phần trên phép tính convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận

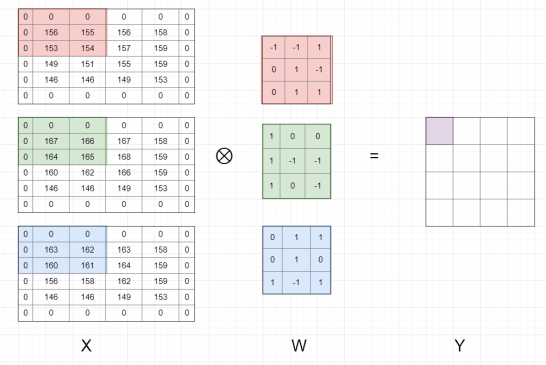


Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.



*Phép tính convolution trên ảnh màu với k=3.*

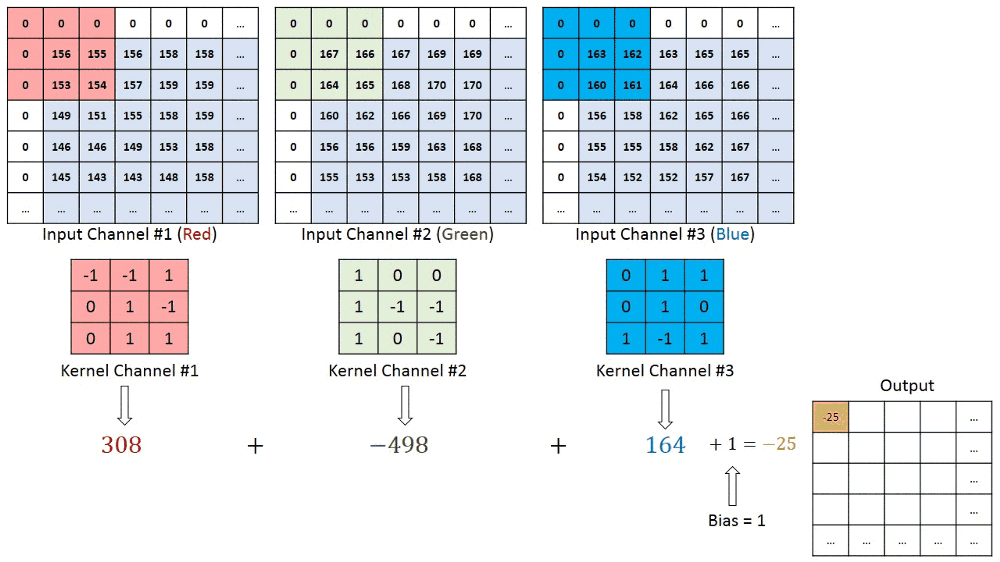
Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.



*Tensor X, W 3 chiều được viết dưới dạng 3 matrix.*

Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là *xijk*​.

*y*11​ = *b* + (*x*111​∗*w*111 ​+ *x*121​∗*w*121​ + *x*131​∗*w*131​ + *x*211​∗*w*211​ + *x*221​∗*w*221​ + *x*231​∗*w*231​ + *x*311​∗*w*311​ + *x*321​∗*w*321​ + *x*331​∗*w*331​) + (*x*112​∗*w*112​ + *x*122​∗*w*122​ + *x*132​∗*w*132​ + *x*212​∗*w*212​ + *x*222​∗*w*222​ + *x*232​∗*w*232​ + *x*312​∗*w*312​ + *x*322​∗*w*322​ + *x*332​∗*w*332​) + (*x*113​∗*w*113​ + *x*123​∗*w*123​ + *x*133​∗*w*133​ + *x*213​∗*w*213​ + *x*223​∗*w*223​ + *x*233​∗*w*233​ + *x*313​∗*w*313​ + *x*323​∗*w*323​ + *x*333​∗*w*333​) = −25

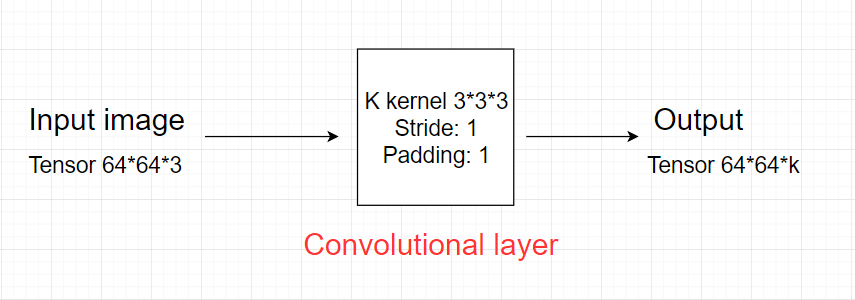
Thực hiện phép tính convolution trên ảnh màu

Nhận xét:

* Output Y của phép tính convolution trên ảnh màu là 1 matrix.
* Có 1 **hệ số bias** được cộng vào sau bước tính tổng các phần tử của phép tính element-wise

**Các quy tắc đối với**[**padding**](https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/#Padding)**và**[**stride**](https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/#Stride)**toàn hoàn tương tự như ở phần trên.**

Với mỗi kernel khác nhau ta sẽ học được những đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được **nhiều thuộc tính** của ảnh. Vì mỗi kernel cho ra output là 1 matrix nên k kernel sẽ cho ra k output matrix. Ta kết hợp k output matrix này lại thành 1 tensor 3 chiều có chiều sâu k.

*Convolutional layer đầu tiên*

Output của convolutional layer đầu tiên sẽ thành input của convolutional layer tiếp theo.

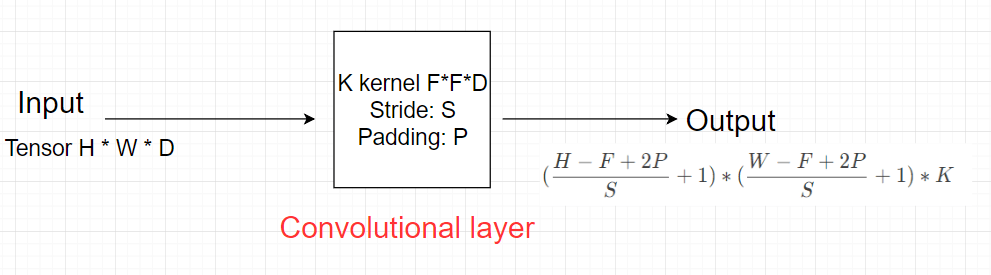
#### Convolutional layer tổng quát

Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D.

Kernel có kích thước F \* F \* D (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

Convolutional layer áp dụng K kernel.

=> Output của layer là tensor 3 chiều có kích thước:



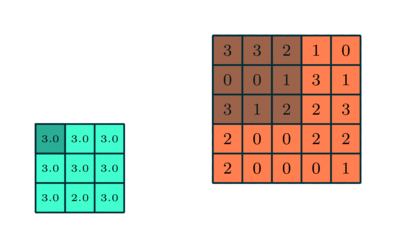
Lưu ý:

* Output của convolutional layer sẽ qua hàm **activation function** trước khi trở thành input của convolutional layer tiếp theo.
* Tổng số parameter của layer: Mỗi kernel có kích thước F\*F\*D và có 1 hệ số bias, nên tổng parameter của 1 kernel là F\*F\*D + 1. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là K \* (F\*F\*D + 1).

### Pooling layer

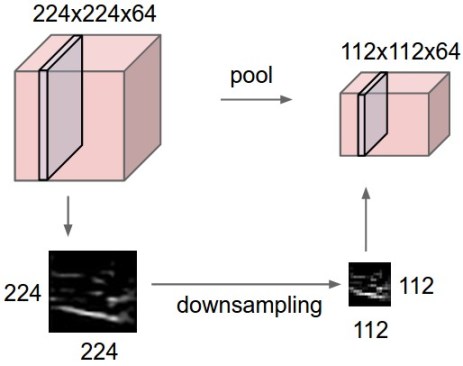
Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

Gọi pooling size kích thước K\*K. Input của pooling layer có kích thước H\*W\*D, ta tách ra làm D ma trận kích thước H\*W. Với mỗi ma trận, trên vùng kích thước K\*K trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.

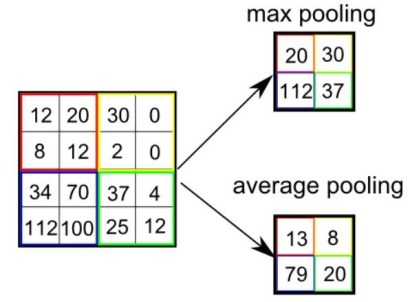


*max pooling layer với size=(3,3), stride=1, padding=0*

Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên .

  
*Sau pooling layer (2\*2).  
Source: http://cs231n.github.io/convolutional-networks/*

Có 2 loại pooling layer phổ biến là: max pooling và average pooling.

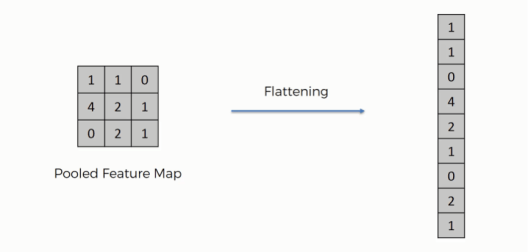


*Ví dụ về pooling layer*

**Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.**

### Fully connected layer

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,…) thì tensor của output của layer cuối cùng, kích thước H\*W\*D, sẽ được chuyển về 1 vector kích thước (H\*W\*D)



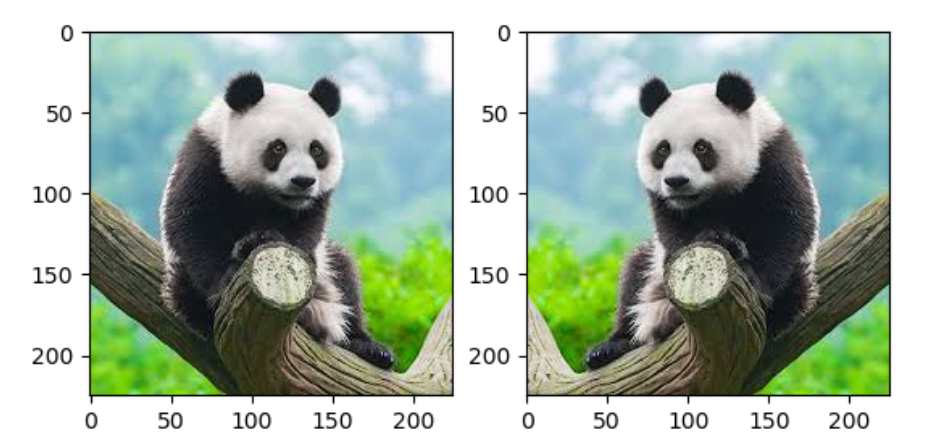
Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

### Một số kỹ thuật Deep learning sử dụng trong bài

#### Data augmentation

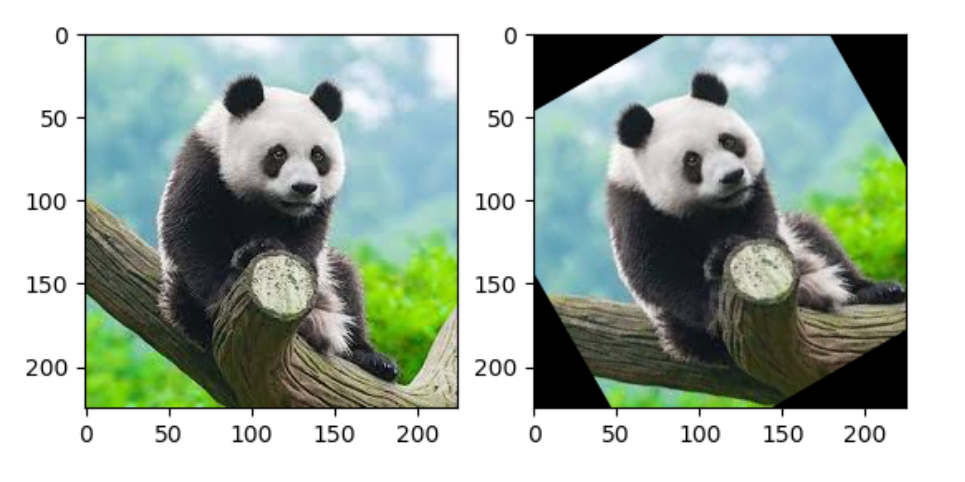
Đây là một kĩ thuật giải quyết vấn đề có ít dữ liệu cho việc training model. Augmentation là kĩ thuật tạo ra dữ liệu training từ dữ liệu mà ta đang có. Cùng xem một số kĩ thuật augmentation phổ biến với ảnh nhé.

**Flip**: Lật ngược ảnh theo chiều dọc hoặc chiều ngang



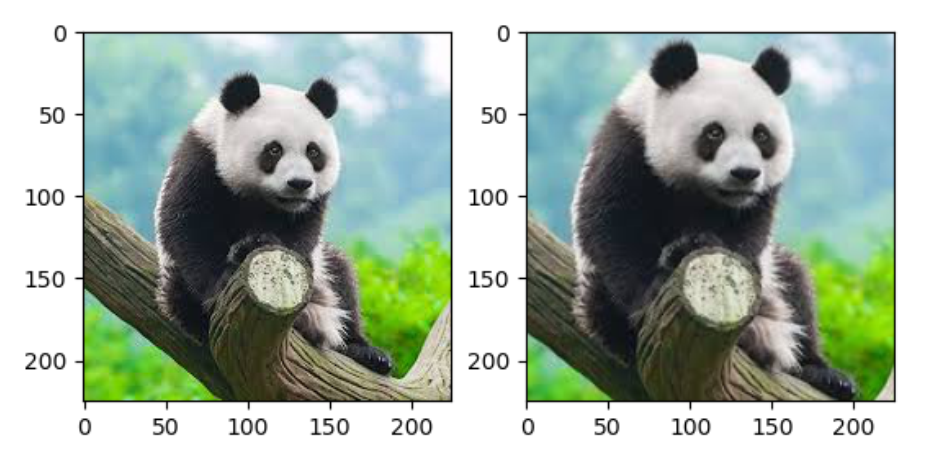
*lật ngược ảnh theo chiều dọc*

**Rotation**: Quay ảnh theo nhiều góc khác nhau



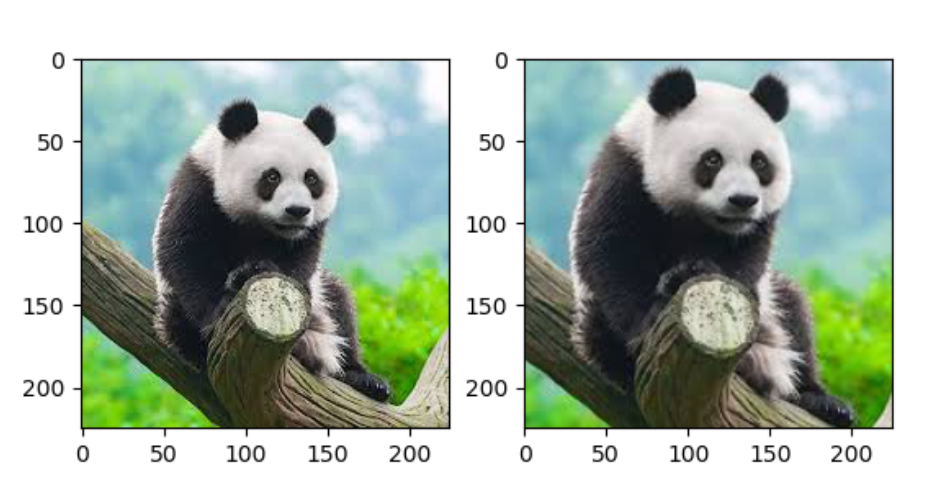
*Rotate ảnh 30 độ*

**Scale**: Phóng to hoặc thu nhỏ ảnh



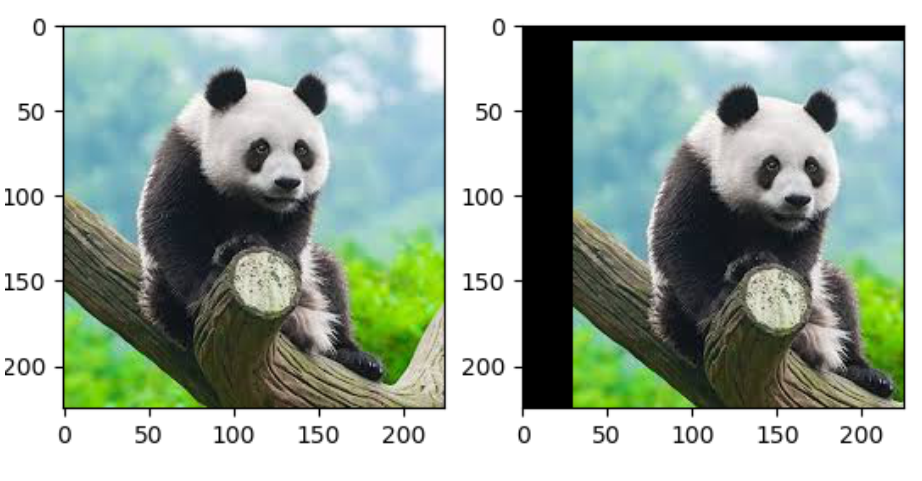
*Scale ảnh*

**Crop**: Cắt một vùng ảnh sau đó resize vùng ảnh đấy về kích thước ảnh ban đầu



*Crop ảnh*

**Translation**: dịch chuyển ảnh theo chiều x, y.



*Translate ảnh 30px theo phương x, 10px theo phương y*

Tuy nhiên khi rotate hoặc translation thì ảnh bị những khoảng đen mà thường ảnh thực tế không có các khoảng đen đấy nên có một số cách để xử lý như: lấy giá trị từ cạnh của ảnh mới để cho các pixel bị đen, gán các giá trị đen bằng giá trị của ảnh đối xứng qua cạnh,...

Nên áp dụng kiểu augmentation nào thì tùy thuộc vào ngữ nghĩa của ảnh trong bài toán đang giải quyết.

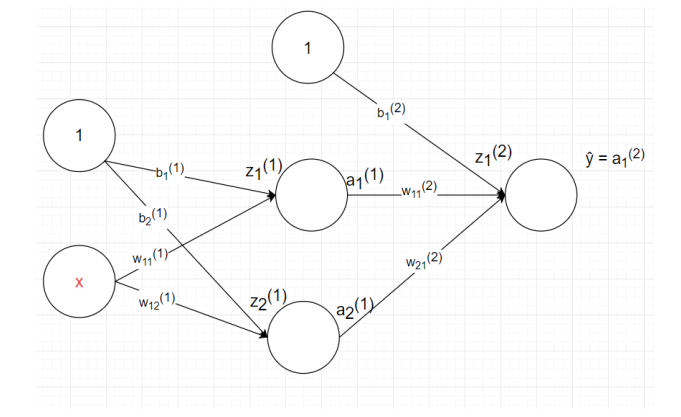
#### Activation function

##### Non-linear activation function

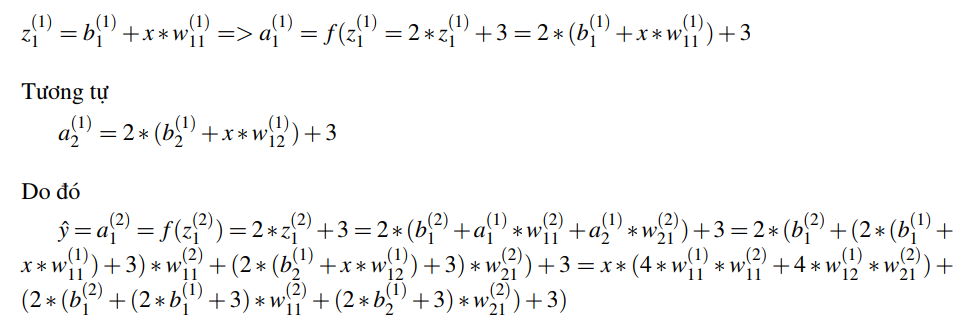
Hàm activation function được dùng sau bước tính tổng linear trong neural network hoặc sau convolutional layer trong CNN. Và hàm activation là non-linear function.

Linear function là gì? Theo wiki, "a linear function from the real numbers to the real numbers is a function whose graph is a line in the plane ", tóm lại linear function là một đường thẳng dạng y = a\*x + b. Vậy sẽ ra sao nếu hàm activation trong neural network là một linear function?

Giả sử hàm activation dạng y = f(x) = 2\*x + 3 và neural network như sau:

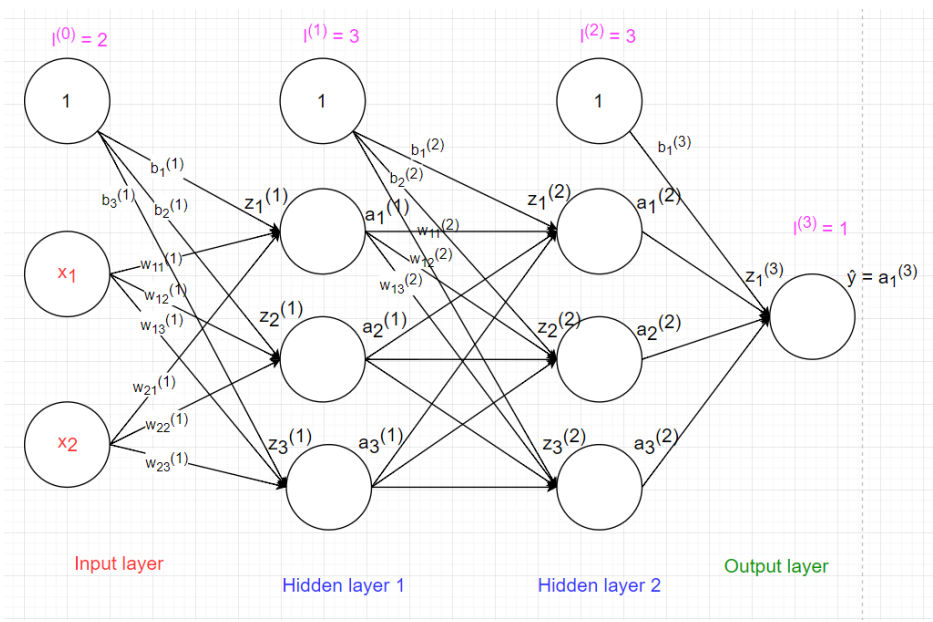


*Mô hình neural network, 1-2-1.*

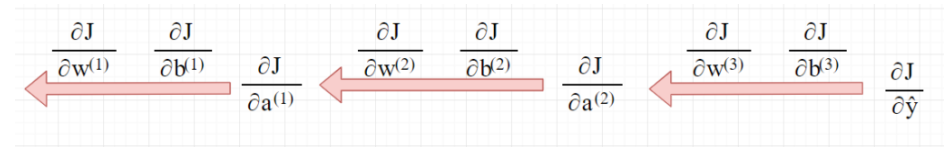
**Tóm lại hay nói cách khác mô hình neural network chỉ là mô hình linear regression đơn giản => **Hàm activation function phải là non-linear function**.

##### Vanishing và exploding gradient

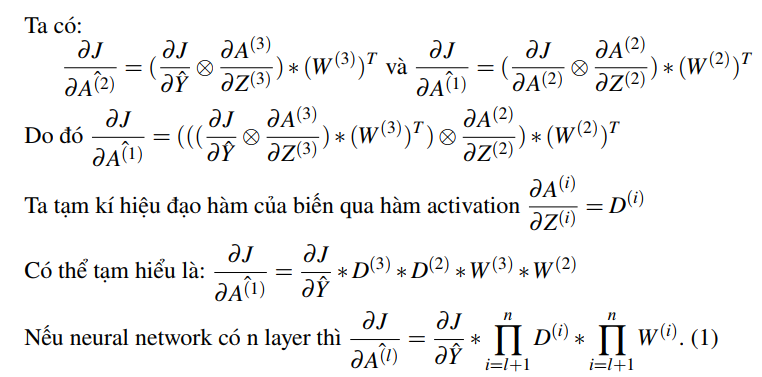
Backpropagation là thuật toán được dùng để tính đạo hàm các hệ số trong neural network với loss function đề rồi áp dụng gradient descent để tìm các hệ số.



*Mô hình neural network 2-3-3-1*



*Quá trình backpropagation*



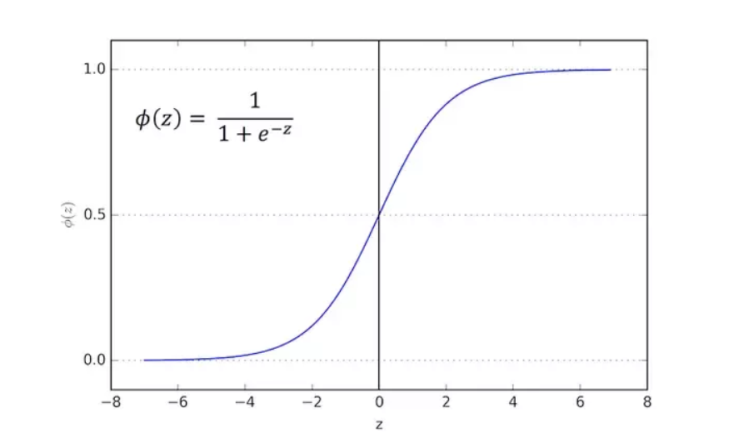
Nhận xét:

* Nếu các hệ số W và D đều nhỏ hơn 1 thì khi tính gradient ở các layer đầu ta sẽ phải nhân tíchbcủa rất nhiều số nhỏ hơn 1 nên giá trị sẽ tiến dần về 0 và bước cập nhật hệ số trong gradient descent trở nên vô nghĩa và các hệ số neural network sẽ không học được nữa. => **Vanishing gradient**
* Nếu các hệ số W và D đều lớn hơn 1 thì khi tính gradient ở các layer đầu ta sẽ phải nhân tích của rất nhiều số lớn hơn 1 nên giá trị sẽ tiến dần về vô cùng và bước cập nhật hệ số trong gradient descent trở nên không chính xác và các hệ số neural network sẽ không học được nữa. => **Exploding gradient**

Cách giải quyết vaninshing/exproding gradient là lựa chọn các giá trị khởi tạo cho hệ số phù hợp và chọn activation function phù hợp

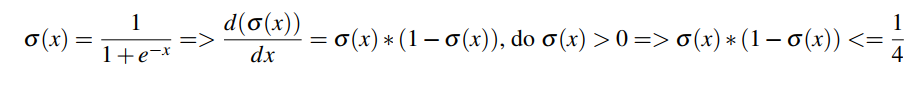
##### Một số hàm activation thông dụng

###### Sigmoid activation function

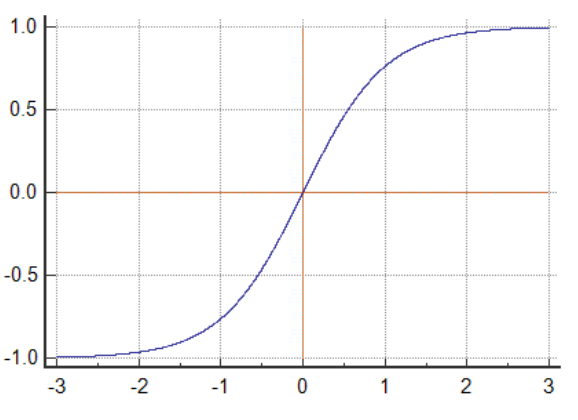


*Hàm sigmoid*

Đạo hàm hàm sigmoid

**Ví dụ nên nếu bạn nhìn vào công thức (1) ở trên thì ở những layer đầu tiên sẽ bị **vanishing gradient**.

###### Tanh activation function

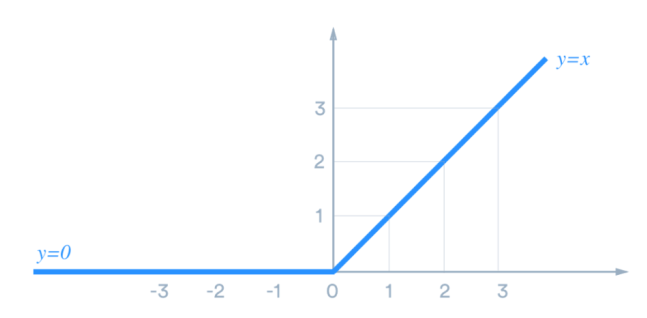


*Hàm tanh*

Hàm tanh:, giá trị g(x) trong đoạn (-1,1)

Đạo hàm hàm tanh: . Do đó khi dùng tanh activation function sẽbị vanishing gradient.

###### ReLU activation function



*Hàm ReLU*

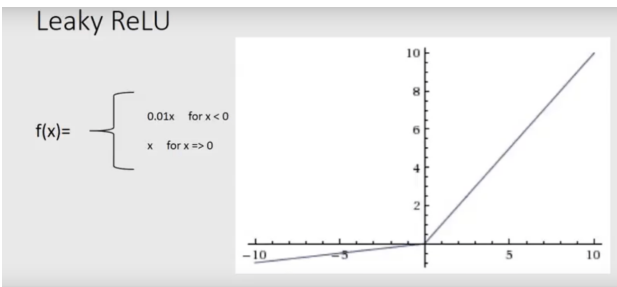
Hàm relu (rectified linear unit): *y* = *max*(0*;x*)

Nhận xét:

* Hàm ReLU activation đơn giản để tính => thời gian train model nhanh hơn.
* Đạo hàm là 1 với x >= 0 nên không bị vanishing gradient.

Tuy nhiên với các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng đấy gọi là "Dying ReLU". Nếu các node bị chuyển thành 0 thì sẽ không có ý nghĩa với bước linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent. => Leaky ReLU ra đời.

###### Leaky ReLU



*Hàm Leaky ReLU*

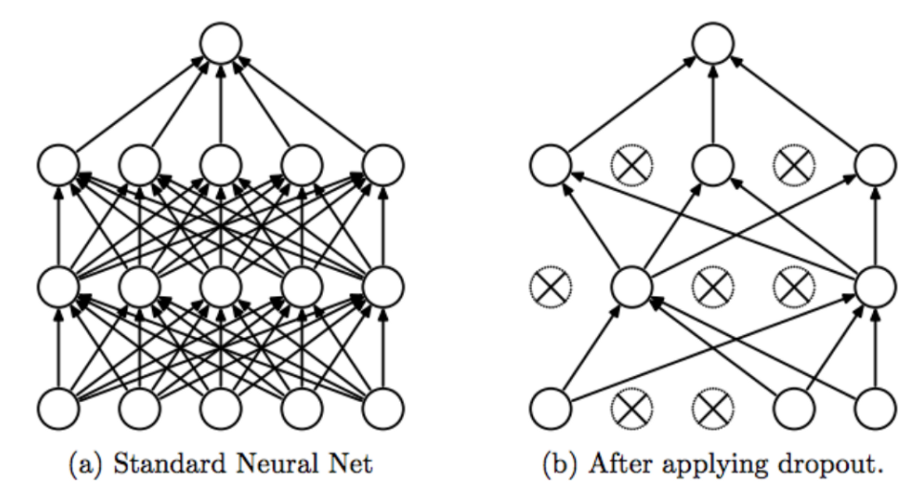
Hàm Leaky ReLU có các điểm tốt của hàm ReLU và giải quyết được vấn đề Dying ReLU bằng cách xét một độ dốc nhỏ cho các giá trị âm thay vì để giá trị là 0.

Lời khuyên: Mặc định nên dùng ReLU làm hàm activation. Không nên dùng hàm sigmoid.

#### Dropout

##### Dropout là gì

Dropout với hệ số p nghĩa là trong quá trình train model, với mỗi lần thực hiện cập nhật hệ số trong gradient descent ta ngẫu nhiên loại bỏ p% số lượng node trong layer đấy, hay nói cách khác là giữ lại (1-p%) node. Mỗi layer có thể có các hệ số dropout p khác nhau.



*So sánh model dropout và neural network thông thường*

Ví dụ mô hình neural network 1-2-1: 1 input layer, 2 hidden layer và 1 output layer. Ví dụ như hidden layer 1, ta dùng dropout với p = 0.6, nên chỉ giữ lại 2 trên 5 node cho mỗi lần cập nhật.

##### Dropout hạn chế việc overfitting

Overfitting là mô hình đang dùng quá phức tạp so với mô hình thật của dữ liệu. Khi ta dùng dropout như hình trên thì rõ ràng mô hình bên phải đơn giản hơn => tránh overfitting.

Thêm vào đó, vì mỗi bước khi train model thì ngẫu nhiên (1-p%) các node bị loại bỏ nên model không thể phụ thuộc vào bất kì node nào của layer trước mà thay vào đó có xu hướng trải đều weight, giống như trong L2 regularization => tránh được overfitting.

##### Lời khuyên khi dùng dropout

* Hệ số p nên ở khoảng [0.2, 0.5] . Nếu p quá nhỏ thì không có tác dụng chống overfitting, tuy nhiên nếu p quá lớn thì gần như loại bỏ layer đấy và có thể dẫn đến underfitting.
* Nên dùng model lớn, phức tạp hơn vì ta có dropout chống overfitting.
* Dropout chỉ nên dùng cho Fully Connected layer, ít khi được dùng cho ConvNet layer
* Hệ số p ở các layer nên tỉ lệ với số lượng node trong FC layer đó.

#### Batch Normalize

Một trong những giả định chính trong được đưa ra trong quá trình huấn luyện một mô hình học máy đó là phân phối của dữ liệu được giữ nguyên trong suốt quá trình training. Đối với các mô hình tuyến tính, đơn giản là ánh xạ input với output thích hợp, điều kiện này luôn được thỏa mãn. Tuy nhiên, trong trường hợp Neural Network với các lớp được xếp chồng lên nhau, ảnh hưởng của các hàm activation non-linear, điều kiện trên không còn đúng nữa.

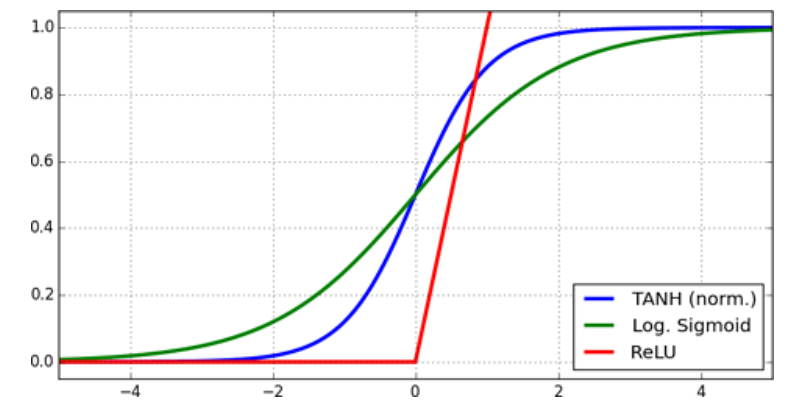
Trong kiến trúc neural network, đầu vào của mỗi lớp phụ thuộc nhiều vào tham số của toàn bộ các lớp trước đó. Hậu quả là trong quá trình backprop, các trọng số của một lớp được cập nhật dẫn đến những thay đổi về mặt dữ liệu sau khi đi qua lớp đó, những thay đổi này bị khuyếch đại khi mạng trở nên sâu hơn và cuối cùng làm phân phối của bản đồ đặc trưng (feature map) thay đổi, đây được gọi là hiện tượng **covariance shifting**. Khi huấn luyện, các lớp luôn phải điều chỉnh trọng số để đáp ứng những thay đổi về phân phối dữ liệu nhận được từ các lớp trước, điều này làm chậm quá trình hội tụ của mô hình.

##### Phân tích nguyên nhân

Vấn đề 1 : Khi dữ liệu chứa nhiều thành phần lớn hơn hoặc nhỏ hơn 0 và không phân bố quanh giá trị trung bình 0 (Non zero mean), kết hợp với việc phương sai lớn (high variance) làm cho dữ liệu chứa nhiều thành phần rất lớn hoặc rất nhỏ. Trong quá trình cập nhật trọng số bằng gradient descent, giá trị của dữ liệu ảnh hưởng trực tiếp lên giá trị đạo hàm (gradient), do đó làm giá trị gradient trở nên quá lớn hoặc qúa nhỏ, như chúng ta đã biết điều này không hề tốt chút nào. Hiện tượng trên xuất hiện khá phổ biến, phụ thuộc nhiều vào việc khởi tạo trọng số, và có xu hướng nghiêm trọng hơn khi mạng ngày càng sâu.

=> Cần một bước normalize các thành phần dữ liệu về cùng mean và chuẩn hóa variance.

Vấn đề 2 : Các hàm activation non-linear như sigmoid, relu, tanh,... đều có ngưỡng hay vùng bão hòa. Khi lan truyền thẳng, nếu dữ liệu có các thành phần quá lớn hoặc quá nhỏ, sau khi đi qua các hàm activation, các thành phần này sẽ rơi vào vùng bão hòa và có đầu ra giống nhau. Điều này dẫn đến luồng dữ liệu sau đó trở nên giống nhau khi lan truyền trong mạng (covariance shifting), lúc này các lớp còn lại trong mạng không còn phân biệt được các đặc trưng khác nhau. Ngoài ra, đạo hàm tại ngưỡng của các hàm activation bằng 0, điều này cũng khiến mô hình bị vanishing gradient. => Cần một bước normalize dữ liệu trước khi đi qua hàm activation.

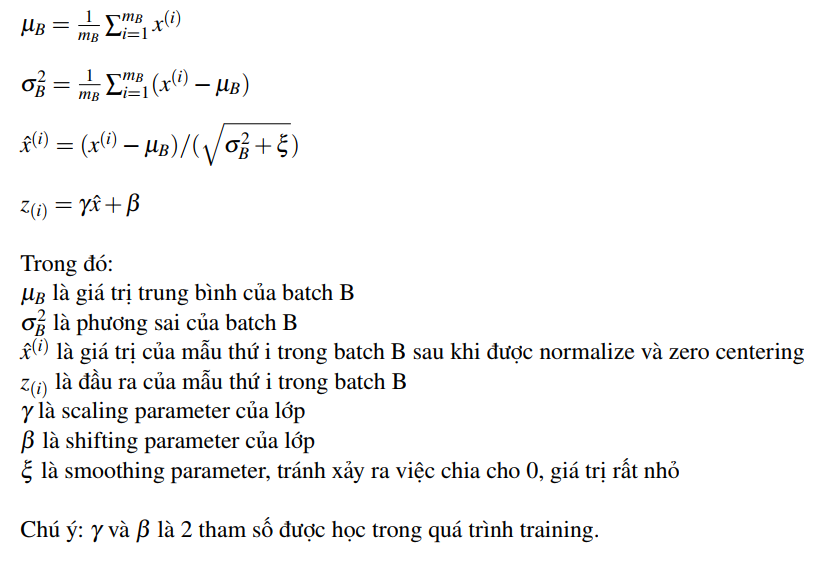


*Đồ thị các hàm activation*

##### Batch Normalization ra đời

Batch normalization thực hiện việc chuẩn hóa (normalizing) và zero centering (mean substracting) dữ liệu trước khi đưa qua hàm activation (giá trị trung bình (mean) sẽ được đưa về 0 và phương sai (variance) sẽ được đưa về 1). Để thực hiện 2 công việc trên, batch normalization tính toán phương sai và độ lệch chuẩn của dữ liệu dựa trên các batchs, rồi sử dụng 2 tham số và tinh chỉnh đầu ra.

Batch normalization:

**

##### Hiệu quả của batch normalization

* Batch normalization đưa dữ liệu về zero mean và chuẩn hóa variance trước khi đưa qua activation function nhờ đó giải quyết các vấn đề vanishing gradient hay exploding gradient.
* Batch normalization cho phép learning rate lớn trong quá trình huấn luyện.
* Batch-Norm giảm thiểu sự ảnh hưởng của quá trình khởi tạo trọng số ban đầu.
* Batch-Norm chuẩn hóa dữ liệu đầu ra của các layer giúp model trong quá trình huấn luyện không bị phụ thuộc vào một thành phần trọng số nhất định. Do đó, Batch-norm còn được sử dụng như một regularizer giúp giảm overfitting

## Bài toán phân loại hình ảnh CIFAR-10

### Xây dựng bài toán

Bộ dữ liệu này chứa các hình ảnh có độ phân giải thấp (32 x 32) với 10 lớp khác nhau của tập dữ liệu này là

* Airplane
* Car
* Bird
* Cat
* Deer
* Dog
* Frog
* Horse
* Ship
* Truck

Chúng ta sẽ dùng bộ dữ liệu này để xây dựng mô hình có thể phân loại chính xác 10 lớp của bộ dữ liệu.

### Chuẩn bị dữ liệu

CIFAR-10 là bộ dữ liệu về thị giác máy tính rất phổ biến. Bộ dữ liệu này đã được nghiên cứu kỹ lưỡng trong nhiều loại mô hình Deep learning phục vụ cho nhận dạng đối tượng.

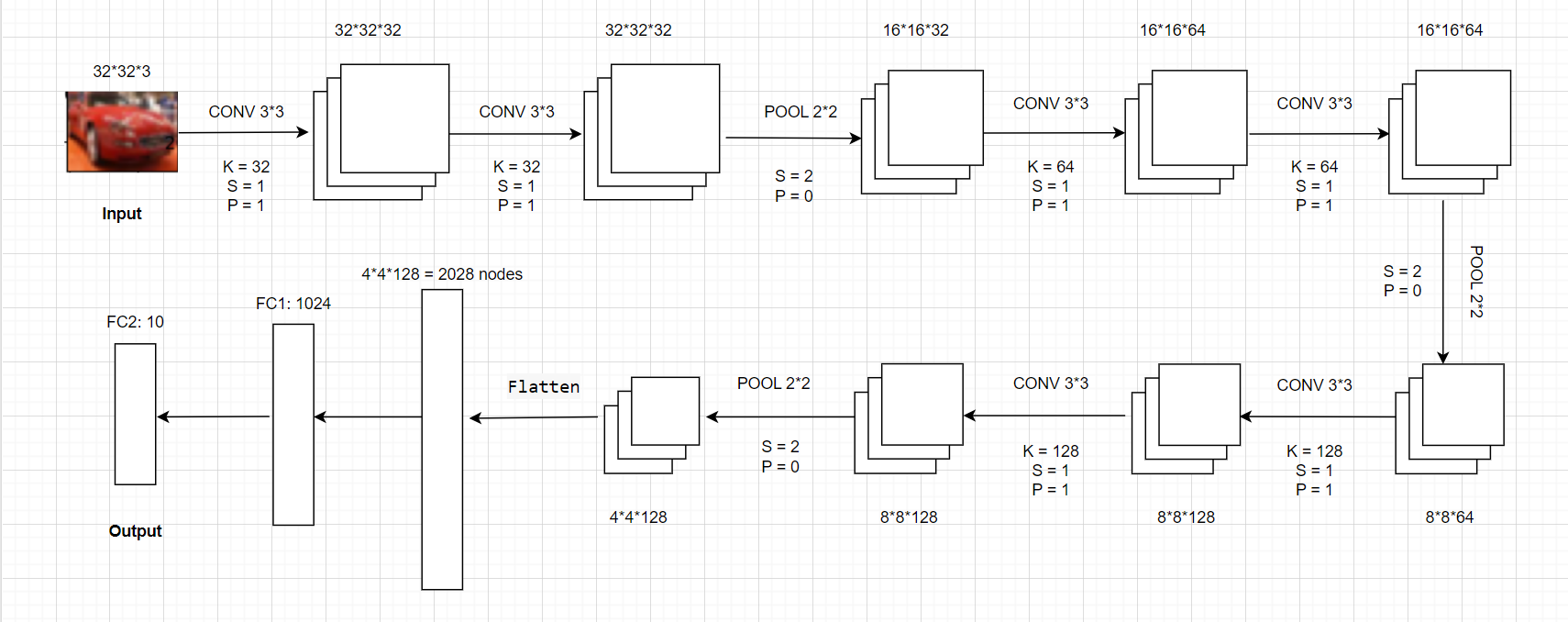
Bộ dữ liệu này bao gồm 60.000 hình ảnh có 50000 hình ảnh train và 10000 hình ảnh test được chia thành 10 target classes, với mỗi class chứa 6000 hình ảnh có kích thước 32 x 32.

CIFAR-10 đã có sẵn trong datasets module của Keras, chúng ta chỉ việc import trực tiếp từ keras.datasets.

### Xây dựng model

Vì input của model là ảnh nên nghĩ ngay đến convolutional neural network (CNN).

Mô hình chung bài toán CNN: Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.



*Model cho bài toán*

Input của model là ảnh màu kích thước 32\*32\*3.