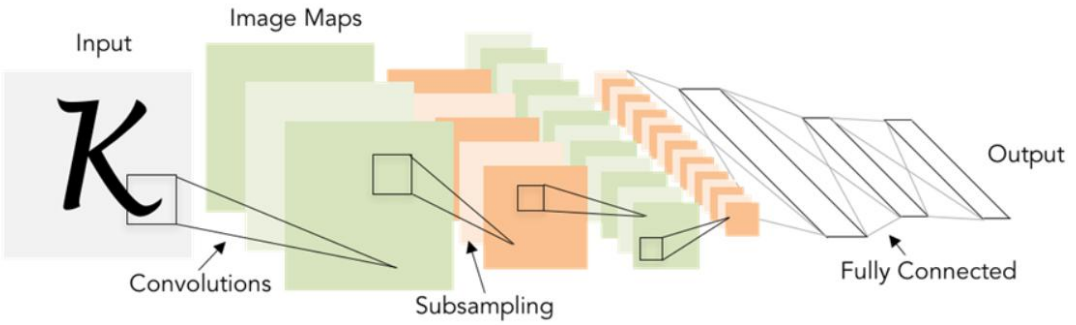
1. Vì sao cần CNN?

- Nếu xài Fully connected:
    + Ảnh sẽ mất cấu trúc không gian (vd ảnh 32x32x7 -> 1 vector 3072)
    + Mỗi neuron nhìn toàn bộ ảnh (Total)
- Nếu xài CNN:
    + Neuron nhìn từng vùng nhỏ (Local)
    + cùng 1 filter trượt khắp ảnh -> cùng bộ trọng số dùng ở mọi vị trí (weight sharing)

Mạng CNN vẫn sẽ có các FC cuối cùng để đổi feature thành class scores -> 1 mảng 1 chiều



2. Image features truyền thống vs ConvNets

Feature truyền thống: giống code tay phân biệt các tính năng/ tính chất đặc biệt, người ta dựa trên trực giác/kiến thức domain để chọn đặc trưng nào quan trọng, tách rời 2 việc trích xuất đặc trưng và phân loại feature cố định + train classifier (các đặc trưng có thể như color histogram, HOG, Bag of Word, …)

ConvNets: làm hết end to end, học từ data

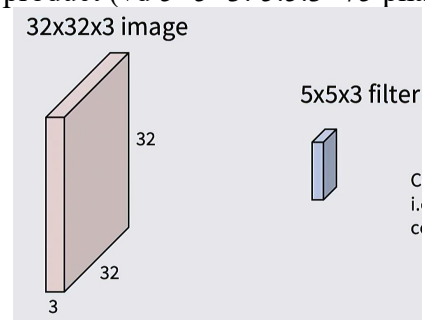
3. Convolution layer

3.1. Input/Filter/Output cơ bản

Input $H \times W \times C_{in}$ (vd $32 \times 32 \times 3$)

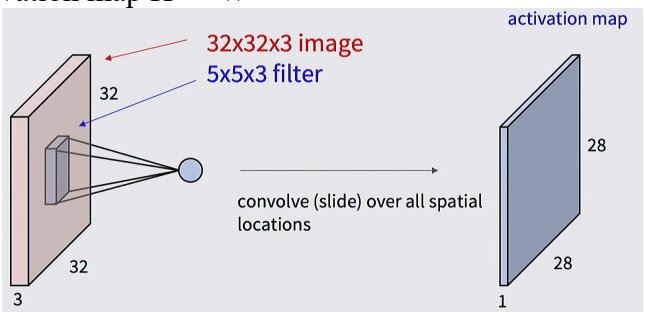
Filter / kernel $K \times K \times C_{in}$: Filter luôn có chiều sâu bằng Input

Mỗi vị trí trượt -> 1 dot product (vd $5 \times 5 \times 3$: 5.5.3=75 phần tử + bias → ra 1 số)

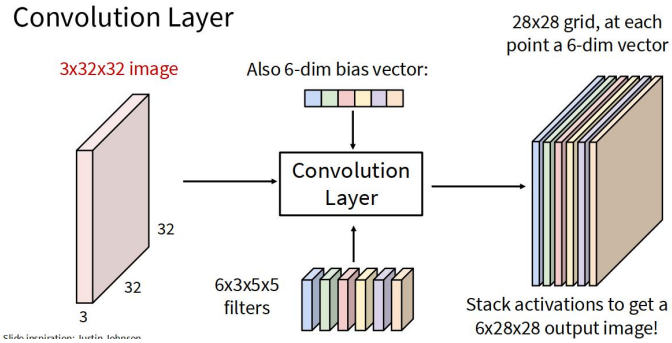


3.2. Nhiều filter -> nhiều activation maps

1 filter -> 1 activation map $H' \times W'$



C_{out} filters -> output sẽ có dạng $H' \times W' \times C_{out}$



Mỗi filter có thể sẽ có 1 bias riêng và sẽ được cập nhật dựa trên gradient descent (vd hình trên sẽ có 6 cái bias)

Tương tự với các filter các trọng số weight trong filter lúc đầu sẽ được chọn random xong sẽ được cập nhật qua gradient descent và backpropagation

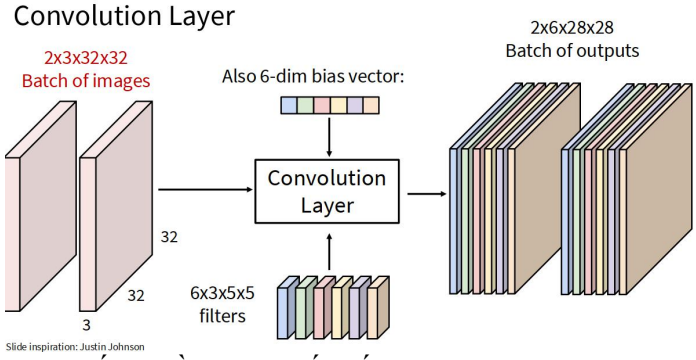
Hyperparameter (các số set up trước): số filter, kích thước filter, số output channel

Parameter (sẽ được cập nhật trong lúc train): value trong các filter và bias

Batch:

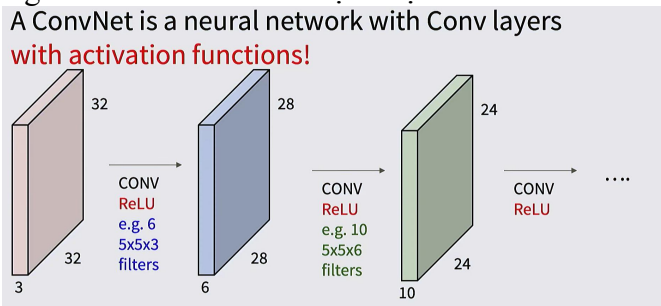
+ Input: $N \times H \times W \times C_{in}$

+ Output: $N \times H \times W \times C_{out}$



Thường người ta sẽ sắp nhiều Conv nối tiếp nhau

-> Problem: giông như NN(FC) thì nếu xếp nhiều cái với nhau thì nó sẽ chỉ thành 1 cái liear bình thường -> cần non-linear để học được nhiều hơn



Mỗi conv layer sẽ học:

+ Layer đầu: thường học edges / oriented gradients / đối lập màu

+ Layer sâu: khó nhìn hơn, học cấu trúc lớn hơn (mắt, chữ, …)

Quan trọng: muốn học tốt thì vẫn cần filter tốt

Trick question: có thể tái tạo lại ảnh sau khi filter ko? Có thể -> gradient descent do that

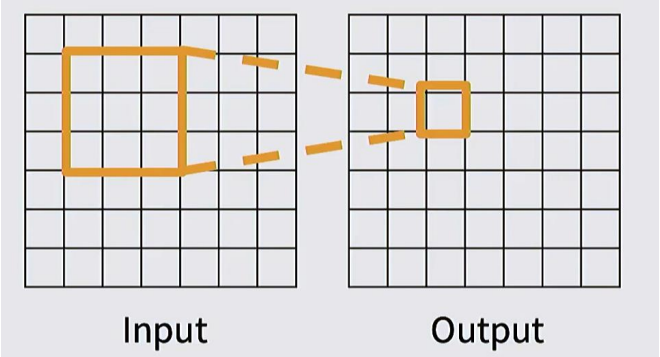
4. Công thức

Với kernel K, padding P, stride S:

$$H' = \frac{H - K + 2P}{S} + 1$$
$$W' = \frac{W - K + 2P}{S} + 1$$

5. Receptive Field: mỗi neuron nhìn được bao nhiêu vùng input?

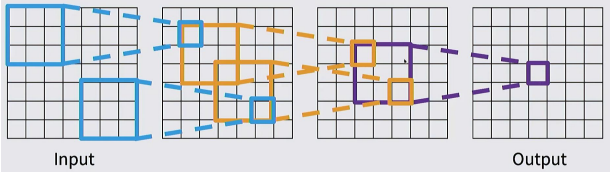
Receptive Field: mỗi pixel ở feature map cuối cùng phụ thuộc vào một miếng (patch) nào đó của ảnh gốc



Với kernel K×K, mỗi output phụ thuộc vào receptive field K×K trên input

Với nhiều conv stride 1:

+ Mỗi layer thêm K−1 vào kích thước receptive field
+ Với L layers: RF=1+L.(K−1)



Thấy rõ, mỗi pixel ở output phụ thuộc vào patch 3x3 của output tới ô cuối đã cần 1 + 4.(3-1) = 9 x 9 ô pixel ban đầu rồi

Vấn đề: ảnh lớn -> cần nhiều layers để thấy toàn ảnh vì cần nhiều conv layer -> downsample trong network bằng stride/pooling.

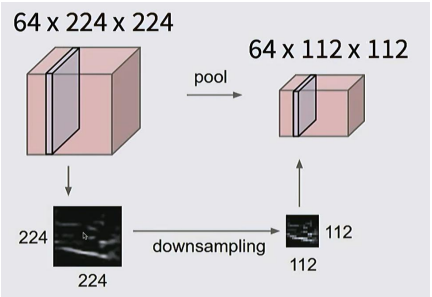
6. Downsampling (Subsampling): Strided Convolution và Pooling

6.1. Strided Convolution

Vd: stride 2 đổ lên sẽ làm giảm kích thước không gian (nhanh hơn) Input 7x7, filter 3x3, stride 2 -> output 3x3 thay vì 5x5 như stride = 1

6.2. Pooling layers

Downsample trừng channle

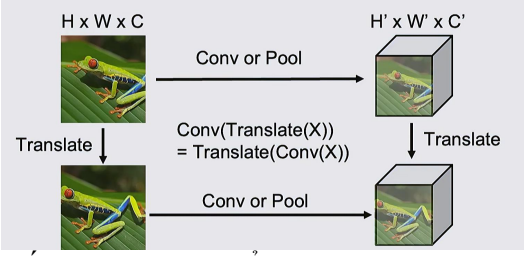


Sẽ có nhiều hướng để giảm: sẽ chọn average, max, min từ các kernel và stride hợp lý (vd hình trên có thể là 2x2 và stride = 2 => giảm 1 nửa) hoặc tạo **invariance** với dịch chuyển nhỏ

Thường thì pooling không cần pooling vì nó không có ý nghĩa lắm ví dụ như max pooling

7. Tính chất quan trọng: Translation Equivariance

Conv/Pool thỏa: Conv(Translate(X))=Translate(Conv(X))



-> Feature không biến mất, nó chỉ di chuyển vị trí theo input

Caution: Các ảnh vào cùng size