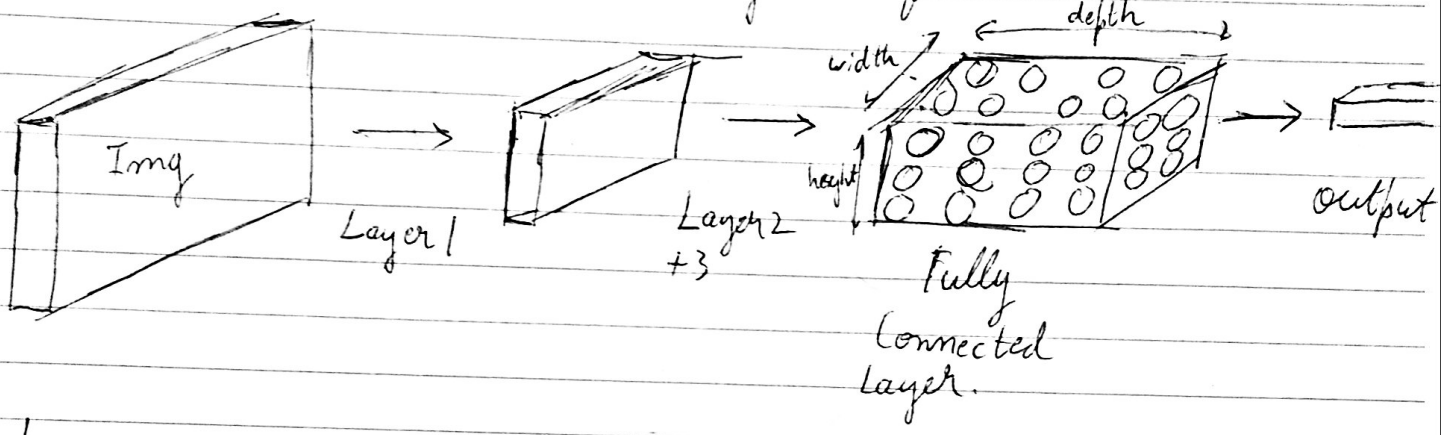
☆ Convolutional Neural Networks (a.k.a. ConvNets)

- Works just like Neural Net, except it has 3 additional layers and each data point is 2D or 3D image.
- ConvNets need to be thought of in 3D.



☆ Layers:

1. Convolution Layer
2. Pooling Layer
3. ReLU / Nonlinearity Layer

Note: For better visualizations of these, go to:
a) Stanford CS231 website
b) http://ujjwalkarn.me/2016/08/11/intuitive-explonation-convnets

Note: Some clarification on terminology:

i) Feature map: Output of Conv. Layer.
ii) Activation map: Output of ReLU layer.
iii) "3D matrix" / "Volume" = 2D matrices stacked one on top of the other. Each of them is a "layer" or "channel"

(Note: "3D matrix" is ~~more~~ formally called a "Tensor", which is 2D matrices stacked along z-axis. We loosely use "3D Matrix" here.)

1. Convolution Layer:

- Convolution is a mathematical integral function; Given two functions f and g, it measures the amount of overlap of g over f, as g is moved over f.
It "blends" them, in a way
- Formula: $(f * g)(t) = \int_0^t f(\tau)\, g(t-\tau)\, d\tau$
$= \int_0^t f(t-\tau)\, g(t)\, d\tau$

- In ConvNets, we take a much simpler and intuitive view of this. We have two matrices A and B:

$$A: \begin{bmatrix} 0 & 9 & 0 & 10 \\ 0 & 11 & 1 & 4 & 7 \\ 7 & 0 & 2 & 6 & 0 \\ 1 & 1 & 3 & 0 & 1 \\ 9 & 0 & 1 & 5 & 3 \end{bmatrix} \qquad B: \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

We are going to "slide" B over A, placing B on every 4 sub-matrices of A, and taking their dot product, like: (as indicated by dotted lines above)

1. $\begin{bmatrix} 0 & 9 \\ 0 & 11 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 11 & 0 \end{bmatrix} \rightarrow$ Sum up = 20

2. $\begin{bmatrix} 9 & 0 \\ 11 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 9 \\ 1 & 11 \end{bmatrix} \rightarrow$ Sum up = 21

1 on on ...

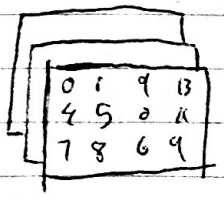
- Now replace A with our input images. This is our convolution layer. B is what we call "filter." The Filter creates a "Feature Map" as the output of this convolution operation.
- Just as we learned weights in the Fully connected Neural Net, the filter is also learnt using backpropagation.
- ~~The~~ After we get the dot product and sums as shown previously, we place these sums in a matrix, resulting in the Feature Map.

$$\begin{bmatrix} 20 & 21 & 12 \\ 9 & \dots & \dots \\ \vdots & & \end{bmatrix}$$

(We can also add a bias term to this.)

Notice how this is just $\text{Sum}(W \cdot x_1 + W \cdot x_2 +$

- Note that images can have "channels," or layers of matrices stacked on on top of the other, where each layer represents a color. So a typical image would have 3 channels - Red, blue and green. In this case, we will have a "3D matrix":



(This is why ConvNets are represented in 3D)
(M x N x 3 array in Python)

- The filter will also be 3D. (So an M x N x 3 array in Python.)
- The result will be M x N x 3, which we will sum up like before

- We can have multiple ~~feat~~ filters, and each will give us a new 2D Feature Map.
- The output of the convolution layer is thus a set or stack of feature maps. (can be represented as an $p \times q \times k$ matrix, where k = no. of filters.

★ Convolution layer hyperparameters:

a) Zero-Padding P: We can pad the input images with 0's all around the border. This does not affect the convolution output, and can be convenient sometimes.

b) Stride S: Stride is the no. of pixels or steps we slide the filter by over the ~~height or~~ image. It is commonly 1 or 2, never 3. We will lose out capturing features (rarely) if S is too high.

c) Filter Dimensions: F (Receptive field size): The dimensions of filter. FxFxk, where k = no. of channels in input images. k=1 (bitmap / black & white images); k = 3 (color images).

d) Input Image Dimension W.

- Using these, we can calculate the volume / dimensions of output (the stack of ~~filte~~ feature maps) using formula: $(W - F + 2P)/S + 1$
For a 5x5 input with a 3x3 filter and 1 zero-padding,
output Volume = $(5 - 3 + 2)/1 + 1 = 5 = 5 \times 5$.

- If we use n filters, it would be 5 x 5 x n. (n feature maps, each of 5x5 size)

2. Non-Linearity Layer (ReLU) – (Activation for Conv.)

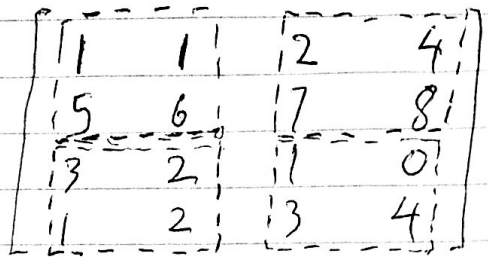
- After we get this feature map from previous layer, we apply a non-linearity on it.
- Most common is ReLU → max(0 x)
  This fairly simple, just apply ReLU on each pixel/element in the Feature Map. So any negative values will become 0.
- Same reason we use in Fully Connected Neural Net. Approximates a function well; easy to compute gradient on because it's derivative is either 1 or 0; solves vanishing gradient problem.

3. Pooling Layer:

- Make the feature map/representation smaller (resize it) by basically systematically picking out only certain values and ignoring the others!
- An example is Max-Pooling, which is the most commonly used. We use an N x N "sliding window" (like the filter) over the input volume and choose the max value in that window. Example: Max Pooling with 2x2 pool filter and stride 2

Input:

2x2 stride 2



4x4 = 16

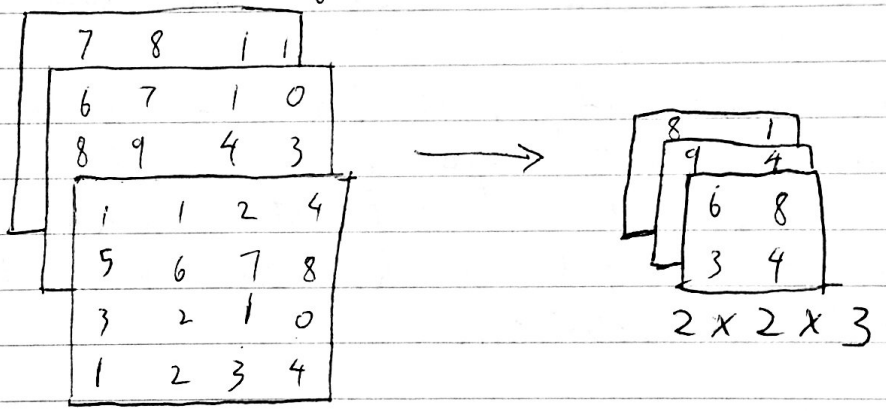
output:

$$\begin{bmatrix} 6 & 8 \\ 3 & 4 \end{bmatrix}$$

2x2 = 4

75% reduction

- This works same way on an input with k channels (3D matrix), where we do the same on each channel / layer of the input volume



4x4x3

2x2x3

- This gives a "summary" for each region of the activation map. We consider only the maximum activations. Preserve prominent features.
- Increases efficiency and speed of computation.

☆ Putting it all together:

The simplest architecture would be:

ConvLayer → ReLU → Pooling → F.C → output
(Fully-connected layer, a.k.a "regular" Neural Net.)

- This would be implemented just like a regular Neural Net. Assume (Conv + ReLU) and Pooling to just be additional layers in the beginning.
- Perform forward pass in the sequential order and then backpropagation, just like we did in the Neural Net. First backpropagate from last layer of F.C backwards into Pooling, ReLU and then Convolution.
- Note that in Conv layer we are learning weights of each filter.
- Naturally, we need to know derivative of conv. operation for this. Below is a mathematical background on it:

1. Forward Convolution Operation:

a) $$y_{ij}^{l} = \left(\sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{ab}\, y_{(i+a)(j+b)}^{l-1} \right)$$ ~~[θ = Non-linearity, can be ReLU, Sigmoid etc]~~

y^l = Output of the conv layer (which is layer l)
y^{l-1} = Input / output of previous layer (layer $l-1$)
y_m = Filter size / dimensions (F)
i, j = row, col of value we are calculating in output feature map y^l.

b) $z^l_{ij} = \theta(y^l_{ij})$ $\quad \theta$ = Non-linearity
= ReLU / Sigmoid / Tanh
↳ Most of the time.

W = filter matrix / tensor

2. Derivative / Backpropagation: Conv.

i) For Weight / Filter

$$\text{Output} = \frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial y^l_{ij}} y^{l-1}_{(i+a)(j+b)}$$

↓

Partial derivative of Error w.r.t w

ii) For the Conv layer output

$$dconv = \frac{\partial E}{\partial y^l_{ij}} = \frac{\partial E}{\partial z^l_{ij}} \theta'(y^l_{ij})$$

θ' = Derivative of Non-linearity. (ReLU, again) which would be $\mathbb{I}(y^l_{ij} > 0)$ ↓ Indicator function.

Note: [This actually just forward pass with spatially flipped filters]

☆ Architecture

- Architecture refers to the arrangement of different layers in the network. It follows the pattern:

Input → [[Conv → ReLU]*N → Pool?]*M → [FC → RELU]*K → FC → Output

- Pooling is optional; K >= 0
- N > 0, N <= 3, usually.
- M >= 0, If M = 0, it is just a Neural Net.

☆ Notes on implementation.

1. im2col

To make training faster, we can use the im2col method, where we reshape all matrices, into 1 column vectors. Do everything as before, (and tensors) all operations remain same.

Example: $K \times K \times C$ dimension images.
$M \times M \times C$ dimension filters.

Resize to: ~~K****C~~
$(W * H * C) \times 1$ images
$(M * M * C) \times 1$ filter

We can pass on entire dataset or batches of images as input to the conv net.

Say we have N images and R filters. Resize to:

$\underline{I} = (W * H * C) \times N$ image matrix
$F = (M * M * C) \times R$ filters matrix.

This makes all calculations, matrix multiplications easier and faster.

Convolution layer does: $F^T \cdot \underline{I} = O$
O is feature map of dimensions $R \times N$.

Note: We must ensure $W * H = M * M$

Note that in order to do this, we must have
$W * H * C = M * M * C$.
$\Rightarrow W * H = M * M$.

So, we could also resize images from $W * H * C$ to $(M * M * C) \times N$

- We then reshape this to the original output volume we would have got $[D = (W - F + 2P)/S + 1]$.

So we resize to $(D \times D \times R)$.

☆ Resources / References:

1. Stanford CS 231 Website: cs231n.github.io/convolutional-networks
2. Leonardo Sontos Github book on Deep Learning / Conv Nets.
leonardoaraujosontos.gitbooks.io/artificial-intelligence
3. Ujjwal Karn Blog: ujjwalkarn.me
4. Andrew Gibiansky Blog: andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks.