Introduction to Applied Machine Learning

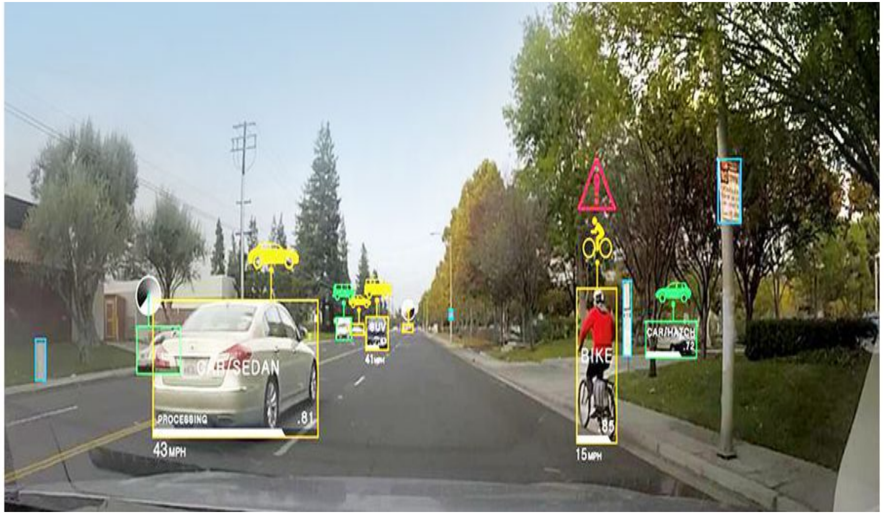
BUMIC + DSC React Series

BOSTON UNIVERSITY
MACHINE INTELLIGENCE
COMMUNITY

Darcy
03/09/2021

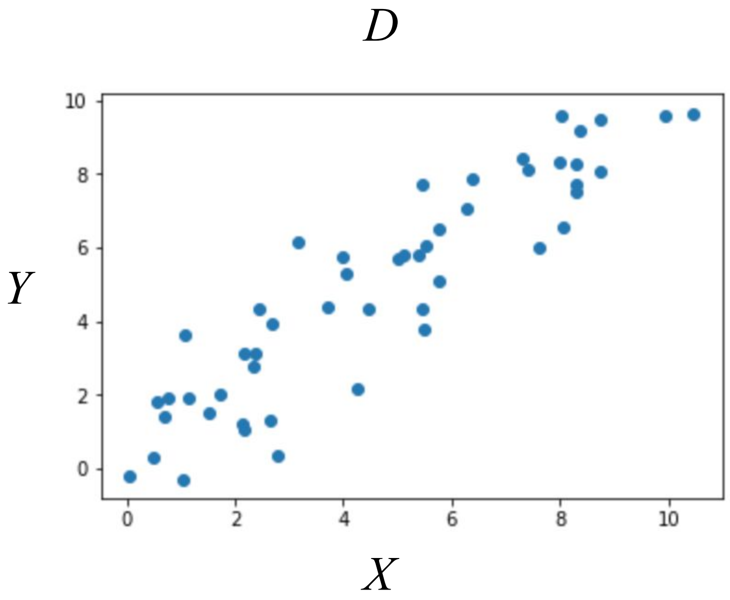
Applications of Deep Learning

1. Cool things using deep learning
    a. Computer Vision
        i. Tesla recognizing items on a street
    b. Text generation
        i. OpenAI GPT3 can solve almost any language task in a few examples
    c. Reinforcement Learning
        i. Can play Atari games, Board games, Real Time Strategy games
        ii. Robotic control
    d. Many more...

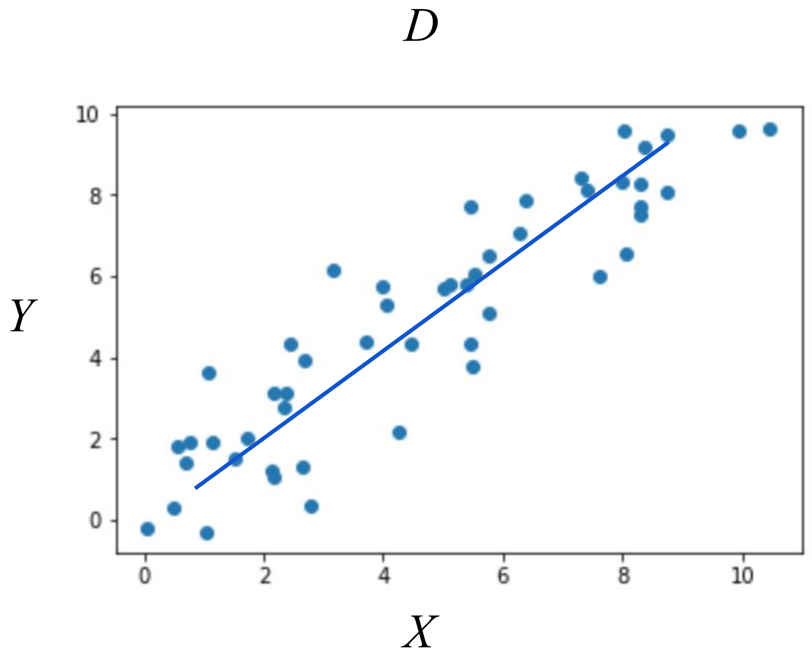


Learning from data

We have some data D



Make an assumption about D



$$y = b + mx$$

$$\downarrow$$

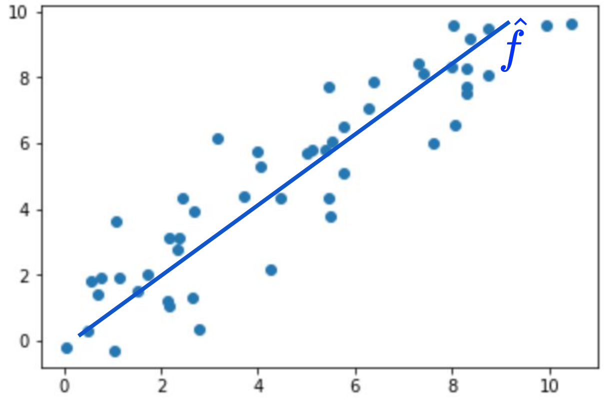
$$\hat{f} = \theta_0 + \theta_1 x$$

What is learning?

The approximation of some unknown function f based on some data D.

D

Y



X

$$f : X \to Y$$

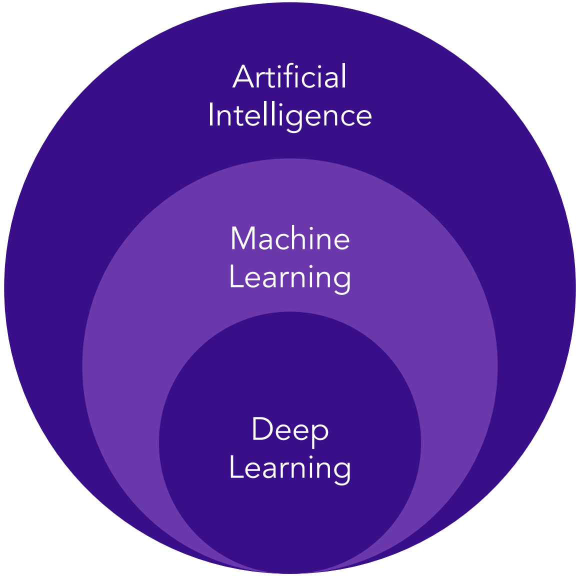
$$\hat{f} = \theta_0 + \theta_1 x$$

How do we set the parameters?

How do we know what assumptions to make?

Intro to Deep Learning

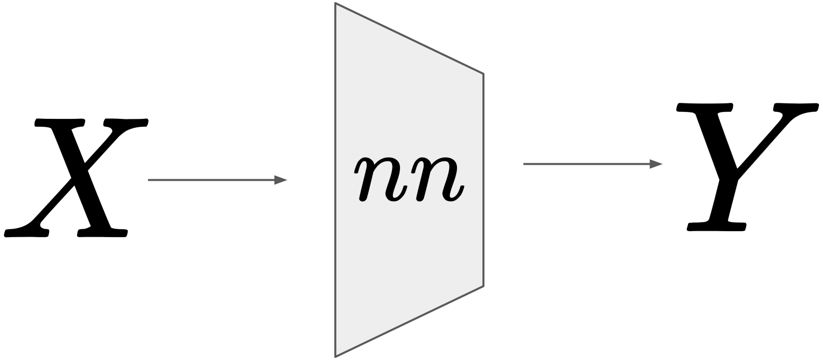
What is Deep Learning



Deep learning is a subset of machine learning

What is Deep Learning

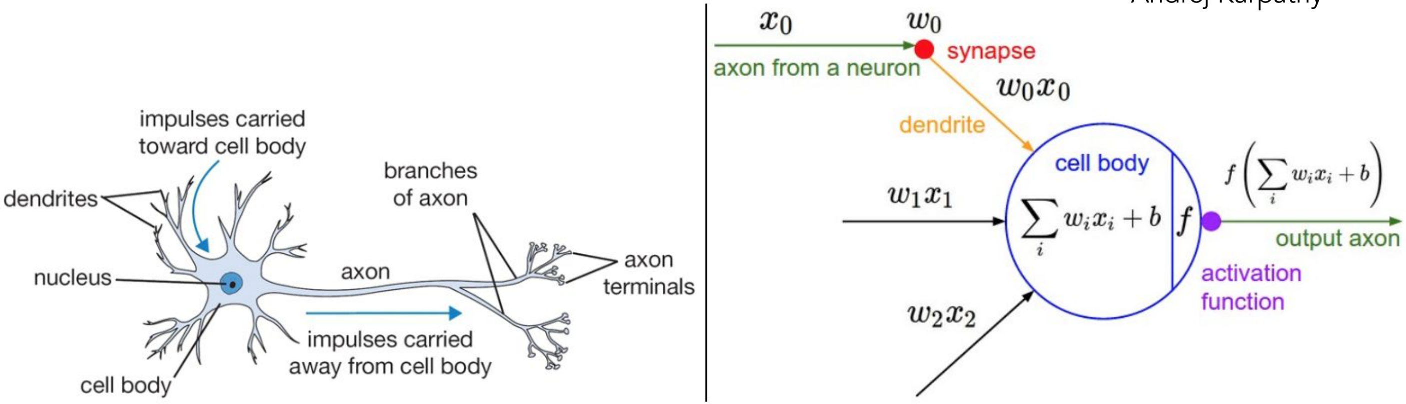
Deep learning learns from data using a class of functions known as Neural Networks



A neural network maps an input to an output

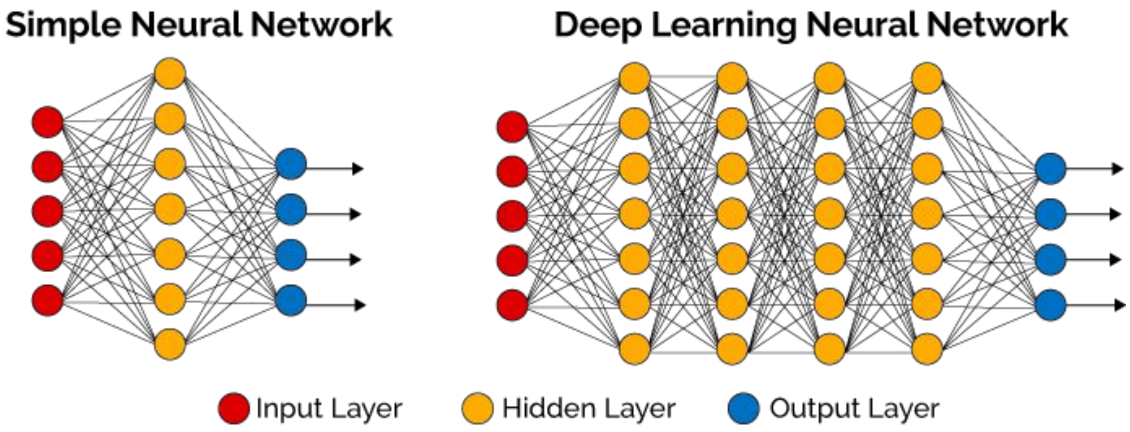
Biological Neuron vs. Artificial Neuron

Andrej Karpathy



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

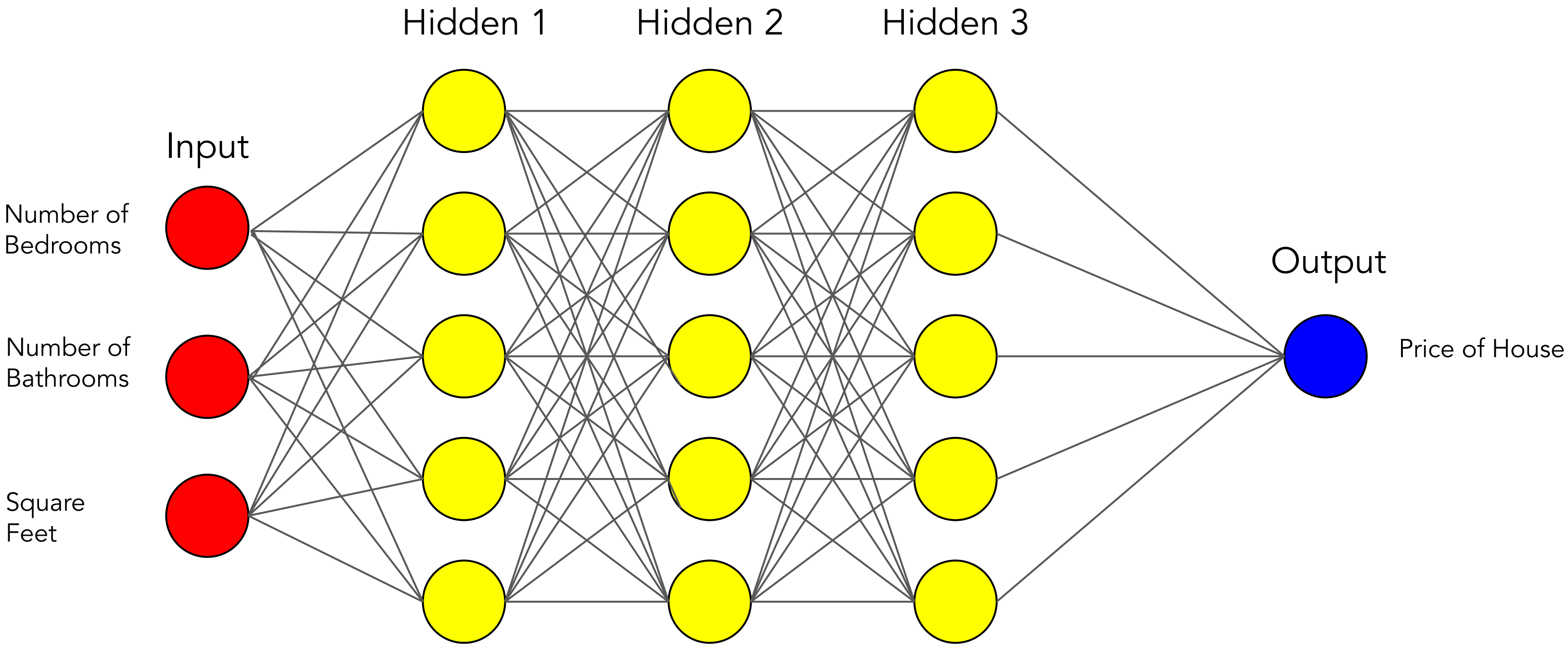
What is a Neural Network?



Steps to Train a NN

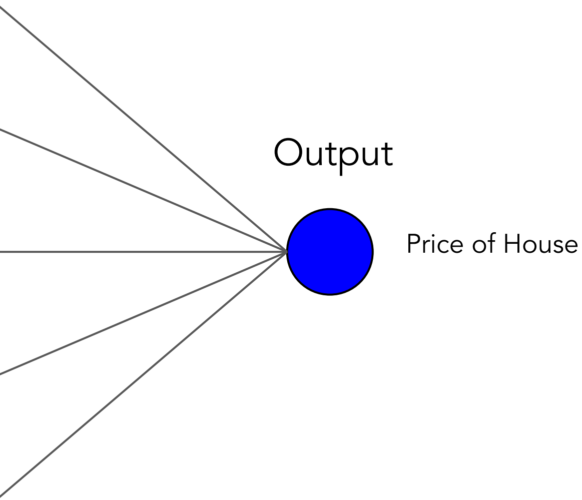
Forward propagation

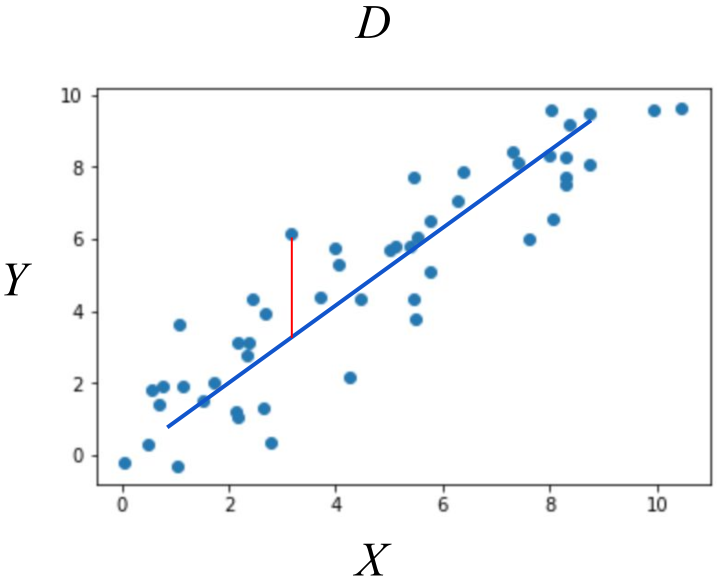
Push example through the network to get a predicted output



Compute the cost

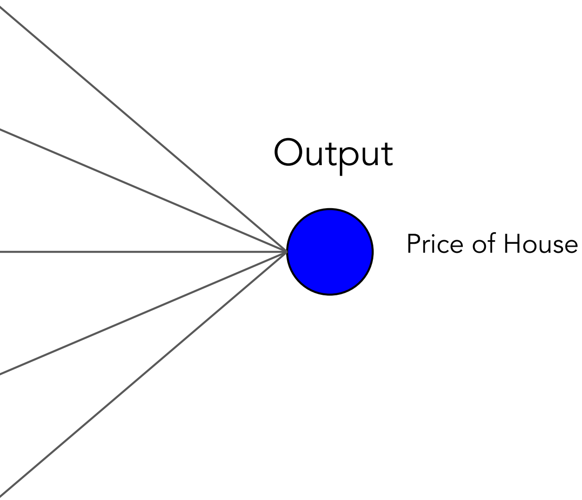
Calculate difference between predicted output and actual data





Compute the cost

Calculate difference between predicted output and actual data

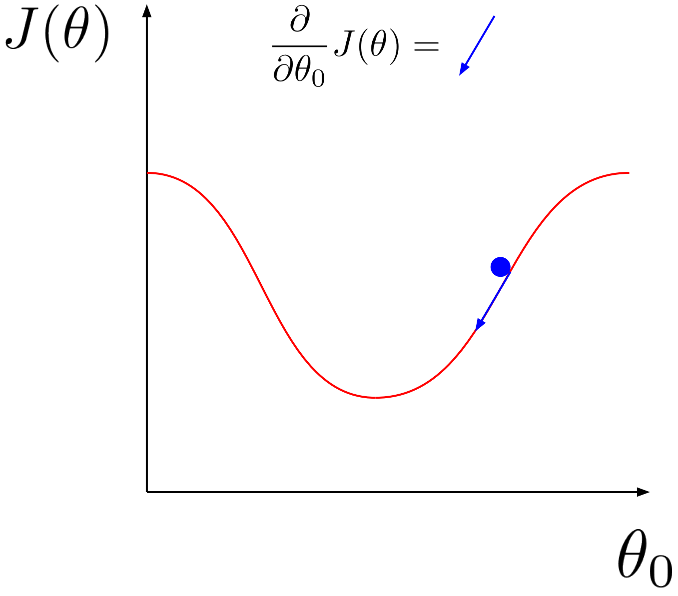


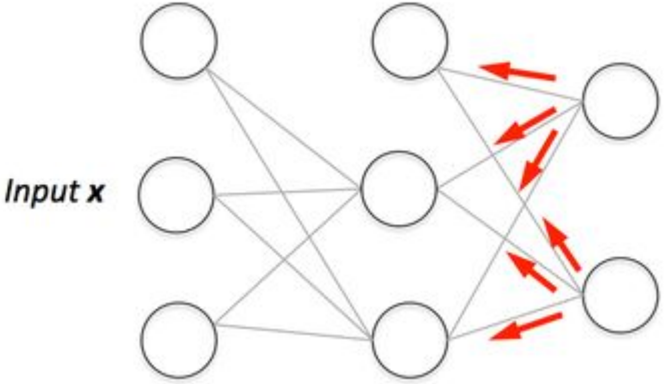
$$J(\theta) = \frac{1}{2m}\sum_{i}^{m}(y_i - \hat{y}_i)^2$$

Where i is the ith training example and m is the number of training examples

Backward propagation - “Update”

Push back the derivative of the error and apply to each weight, such that next time it will result in a lower error





https://hmkcode.github.io/ai/backpropagation-step-by-step/

Convolutional Neural Networks

Image Data

- Images are commonly represented in code as a 3D array of pixels. Here, we notice 3 represents RGB values
- In vanilla neural networks, we would simply flatten this 3D array into a 3072 length vector. However, by doing this, we lose spatial correlation between pixels close to other pixels

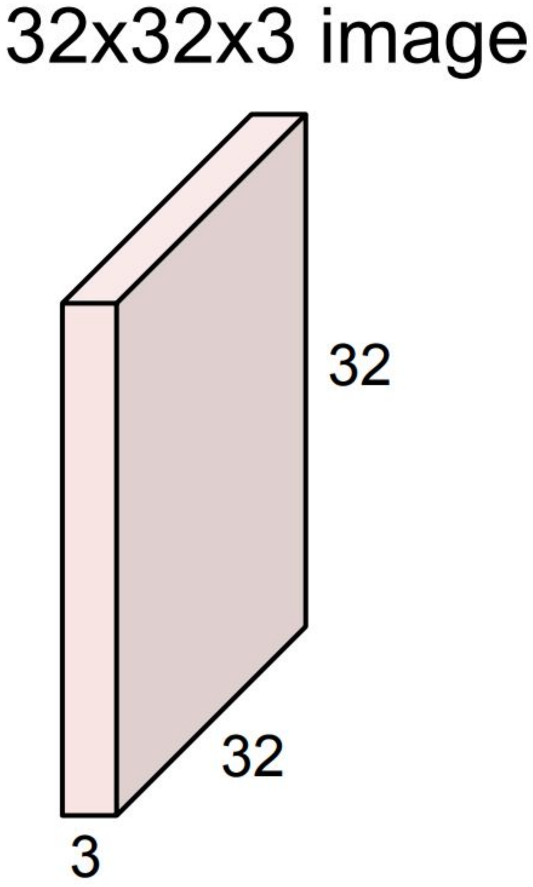
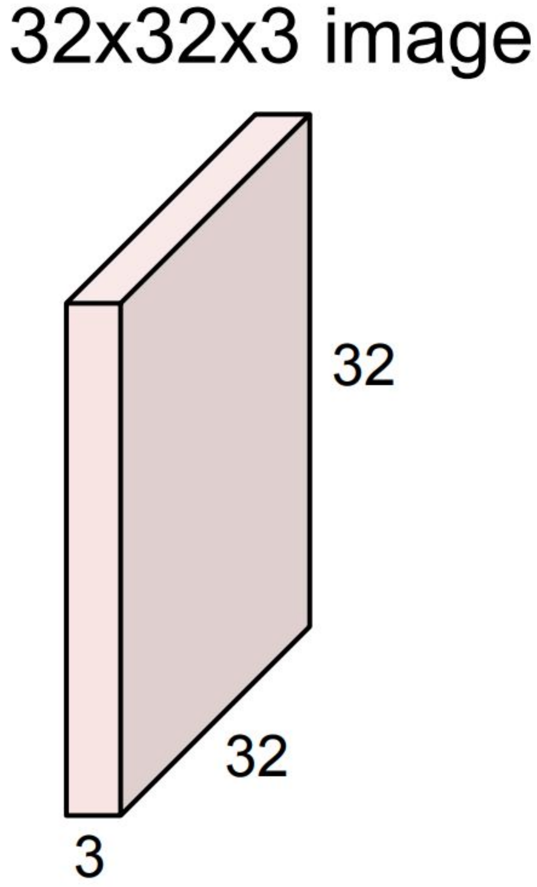


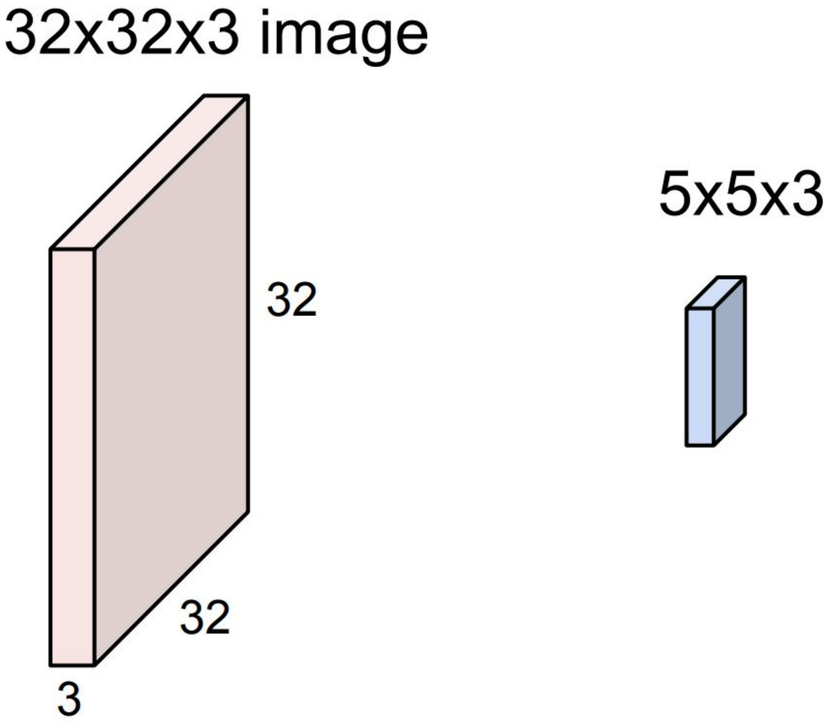
Image Data

- In 2012 a paper called AlexNet out competed state of the art image classification models through the usage of kernels (also called filters)



Kernel

- Kernel: a small matrix used for feature detection on an image
  - Also called a filter
- Usage
  - Superimpose the kernel over a section of an image
  - Do element-wise multiplication between the weights in the kernel and the values in the image
  - Record the sum of the multiplications

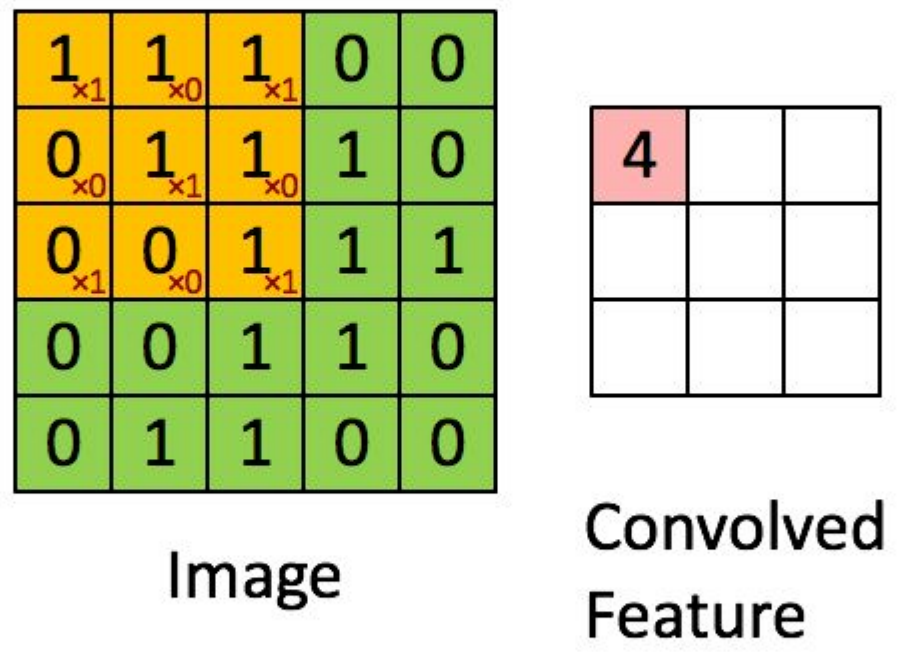


Example Convolution

Example: Multiply the 5x5 image by a 3x3 kernel with weights:

1 0 1
0 1 0
1 0 1

The output?
Sum of weight times part of image to a single number.



Kernel example

$$\begin{bmatrix} 6 & 3 & 2 \\ 4 & 3 & 1 \\ 3 & 5 & 5 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \text{sum}\left(\begin{bmatrix} 6*0 & 3*1 & 2*0 \\ 4*1 & 3*2 & 1*1 \\ 3*0 & 5*1 & 5*0 \end{bmatrix} \right) = \boxed{19}$$

Section of an image

Kernel

Kernel example (cont.)

$$\begin{array}{|c|c|c|}\hline 3 & 3 & 1 \\ \hline 4 & 6 & 5 \\ \hline 3 & 5 & 2 \\ \hline \end{array} \;*\; \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \;=\; \text{sum}\left(\begin{array}{|c|c|c|}\hline 3*0 & 3*1 & 1*0 \\ \hline 4*1 & 6*2 & 5*1 \\ \hline 3*0 & 5*1 & 2*0 \\ \hline \end{array}\right) \;=\; \boxed{23}$$

Section of
an image

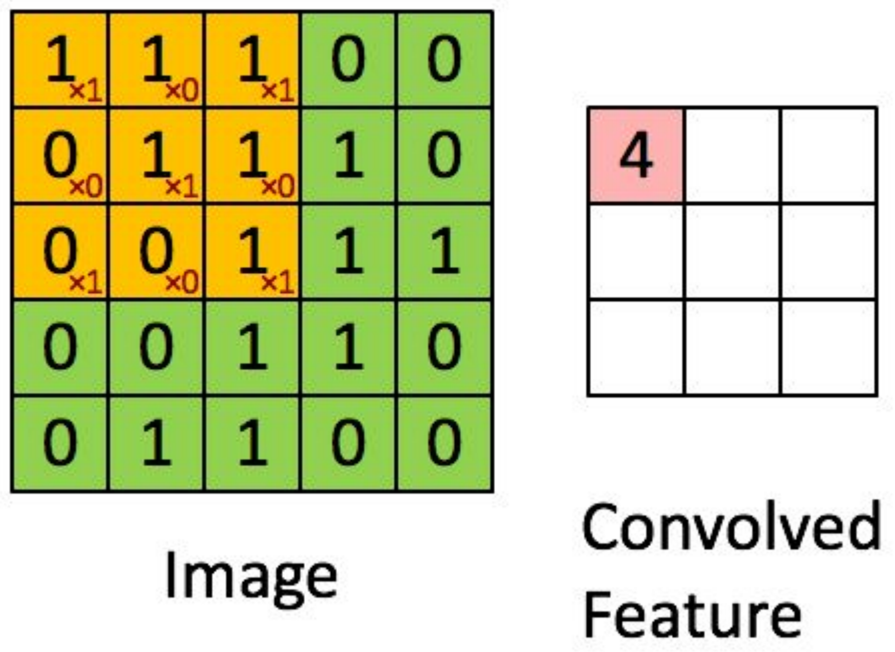
Kernel

This image section contains the same values as before, but they have been rearranged, resulting in a greater activation with this kernel

Example Convolution

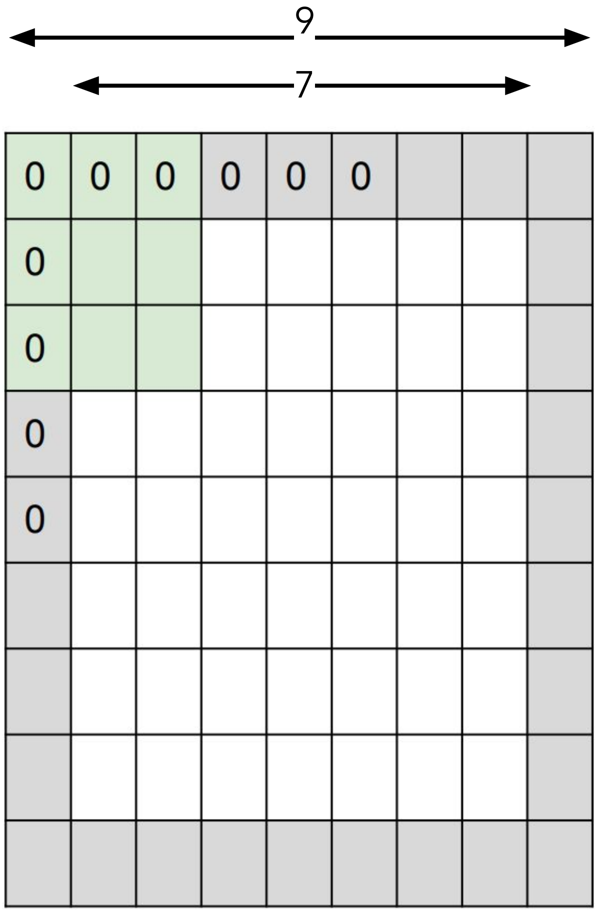
- Note that the output is smaller than the input

- This can be prevented by using padding around the edges of the image.



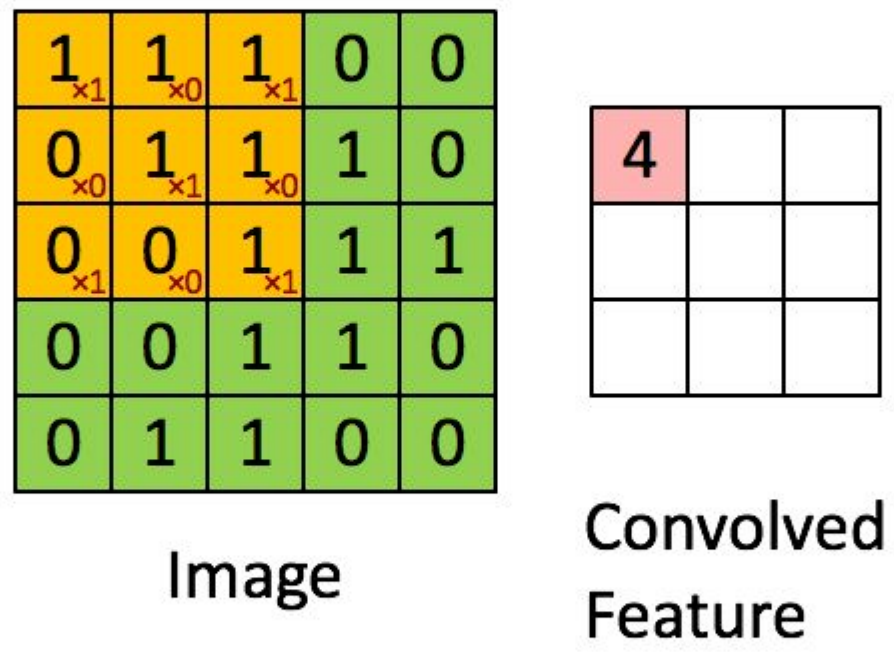
Padding

- Before padding:
  - 7x7 input, 3x3 filter creating a 5x5 sized output
- After padding:
  - 9x9 input, 3x3 filter creating a 7x7 sized output which maintains the same size as our input
- Edges and corners aren’t as accurate but in practice this works well enough



Stride

- Here, the kernel is moving one pixel at a time ("stride" = 1)
- The kernel can move by more than one pixel at a time
- Size = (N - F) / Stride + 1



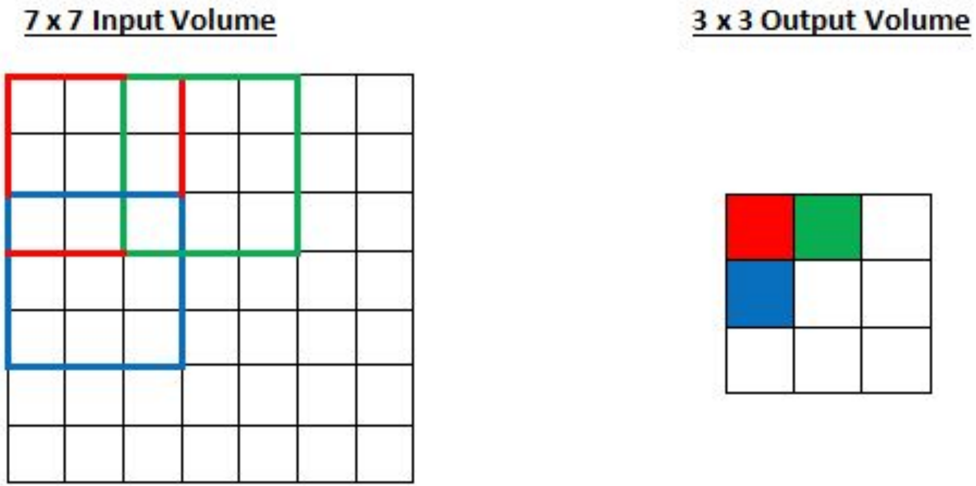
Stride

- Increasing stride decreases the size of the output

- Here, stride = 2

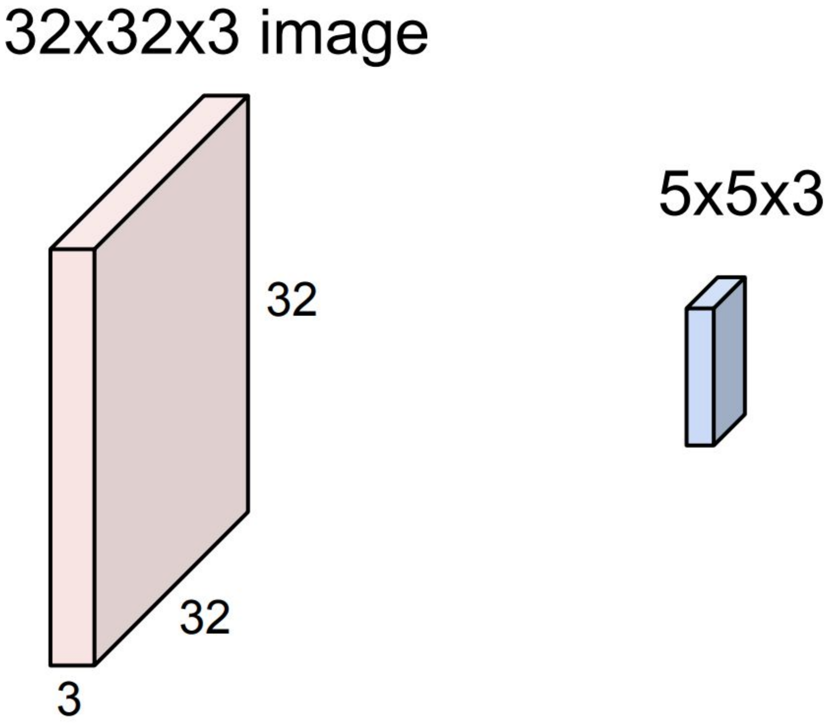
- (N - F) / Stride + 1

  (7 - 3) /   2   + 1 = 3



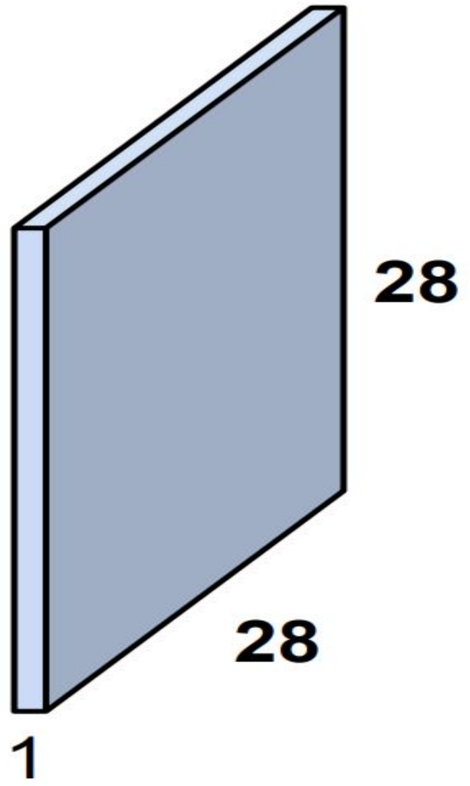
Dimensionality Practice

- What would be the output size of a 5x5x3 filter with a 32x32x3 image and a stride of 1?
- (N - F) / Stride + 1



Dimensionality Practice

- (32 - 5) / 1 + 1 = 28
- Now let's say we had a stride of 2,
  - (32 - 5) / 2 + 1 = 14.5
  - Fractional size means the filter hangs off the input
  - We wouldn't use this stride value consequently

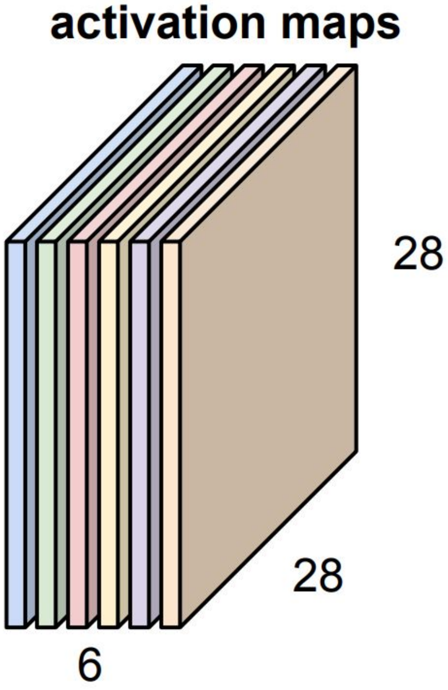


Intentionally Shrinking Output Size

- Now, let's say you want to shrink your outputs (which are inputs to the next layer) to reduce operations.
- You can do this by either increasing the stride
  - (N - F)/Stride + 1
- Alternatively, you can use a pooling layer

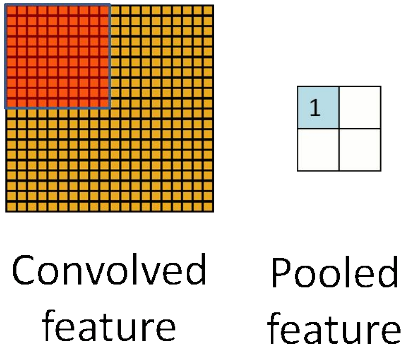
Conv Layer Output

- Use multiple kernels for multiple activation maps
- In this example, we have 6 activation maps each created through a different filter with its own set of weights and biases



Additional Layers

Pooling Layers



- Limitation of output of Convolutional Layers:
  - Record the precise position of features in the input
  - Small movements in the position of the feature in the input image will result in a different feature map
- Solution: Pooling Layers
  - Lower resolution version of input is created with large and important structure elements preserved
  - Reduces the computational cost by reducing the number of parameters to learn

Max Pooling

Extracts the sharpest features of an image, making it more general

Input (4 x 4)

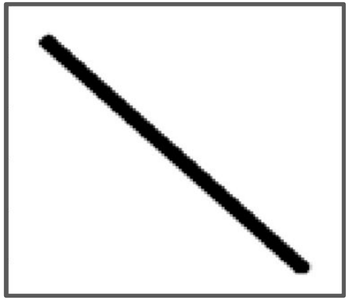
255	0	0	0
0	255	0	0
0	0	255	0
0	0	0	255

Output (2 x 2)

255	0
0	255

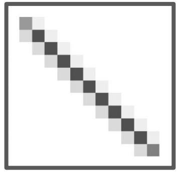
Average Pooling

Takes average feature of an image, minimize overfitting



Input (4 x 4)

0	255	255	255
255	0	255	255
255	255	0	255
255	255	255	0

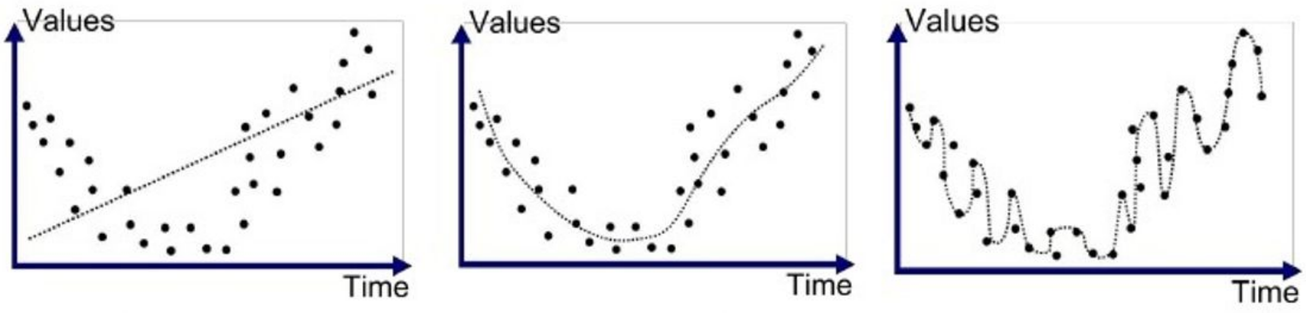


Output (2 x 2)

128	255
255	128

Dropout

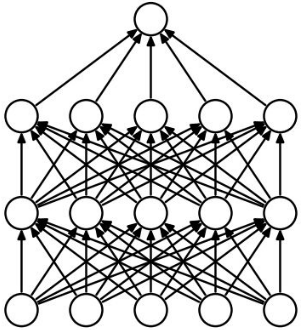
1. First, what is overfitting?
   a. Overfitting is when the neural network corresponds too closely to the dataset, and cannot be generalized. This tends to happen when a model is excessively complex relative to the data
   b. Conversely, underfitting is when the network cannot capture the underlying trend of the dataset which may happen if your network is not complicated enough.



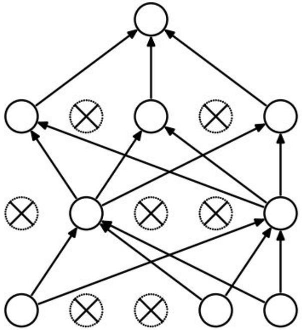
Underfitted | Good Fit/Robust | Overfitted

Dropout - How can we solve overfitting?

1. Training phase
    a. Each weight has a probability *p* that they will be multiplied by zero (dropped). This probability is often set to 0.5, which is considered to be close to optimal for a wide range of networks and tasks
    b. This has the effect of removing random connections between activations effectively creating a new network/outlook on the data per each train set



(a) Standard Neural Net



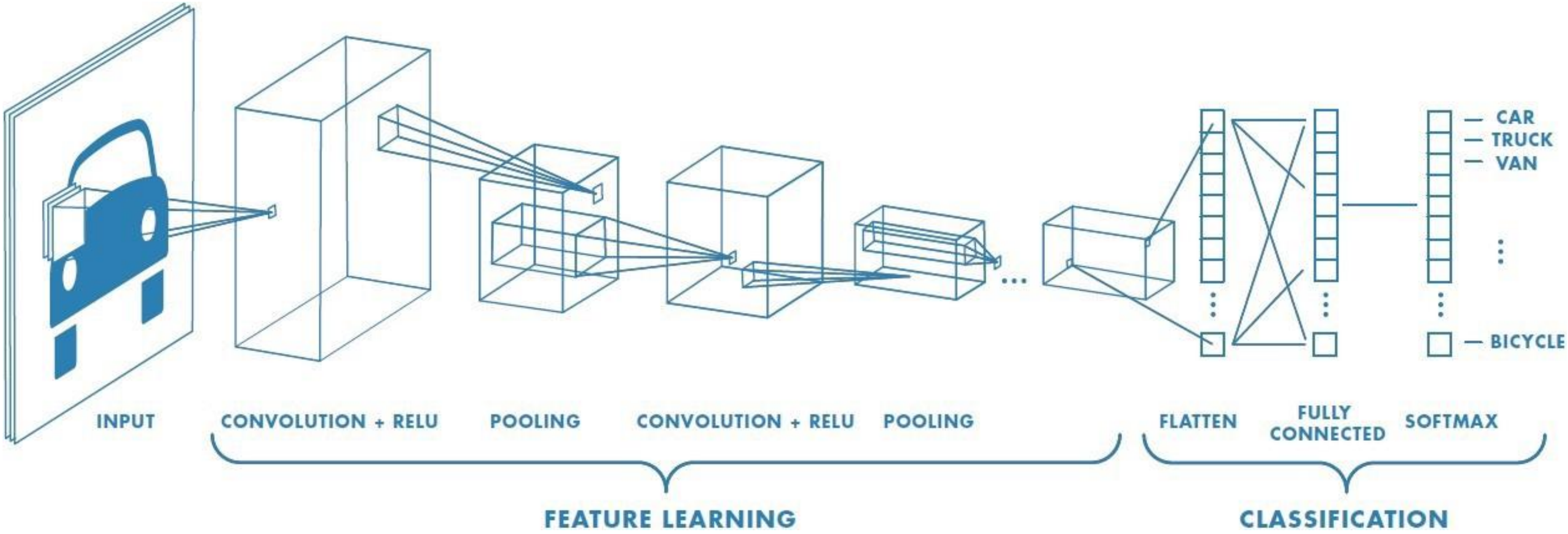
(b) After applying dropout.

Dropout - How can we solve overfitting?

2. Post Train

    a. After training weights will be abnormally high as they were adjusted assuming only (1-p) percent of the weights would be summed together and used.
    b. To fix this we normalize weights to lower the expectation of each weight. We do this by scaling each weight by 1/p
    c. “This makes sure that for each unit, the expected output from it under random dropout will be the same as the output during pretraining.” ~Dropout: A Simple Way to Prevent Neural Networks from Overfitting
        i. http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Convolutional Neural Network



So what does our network look like?