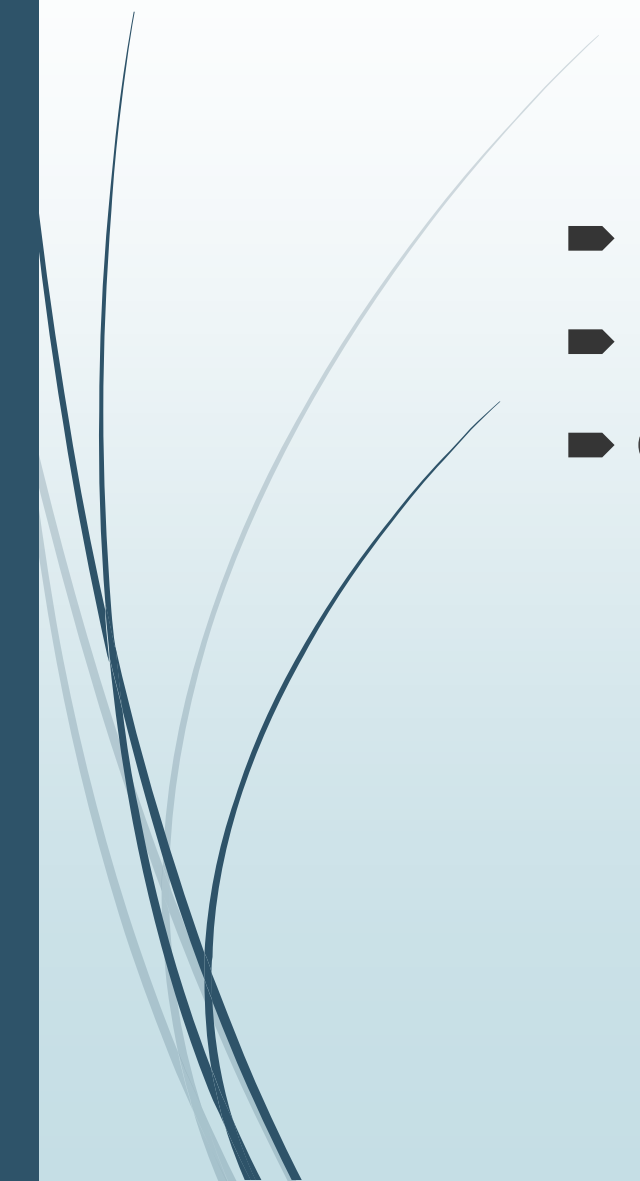


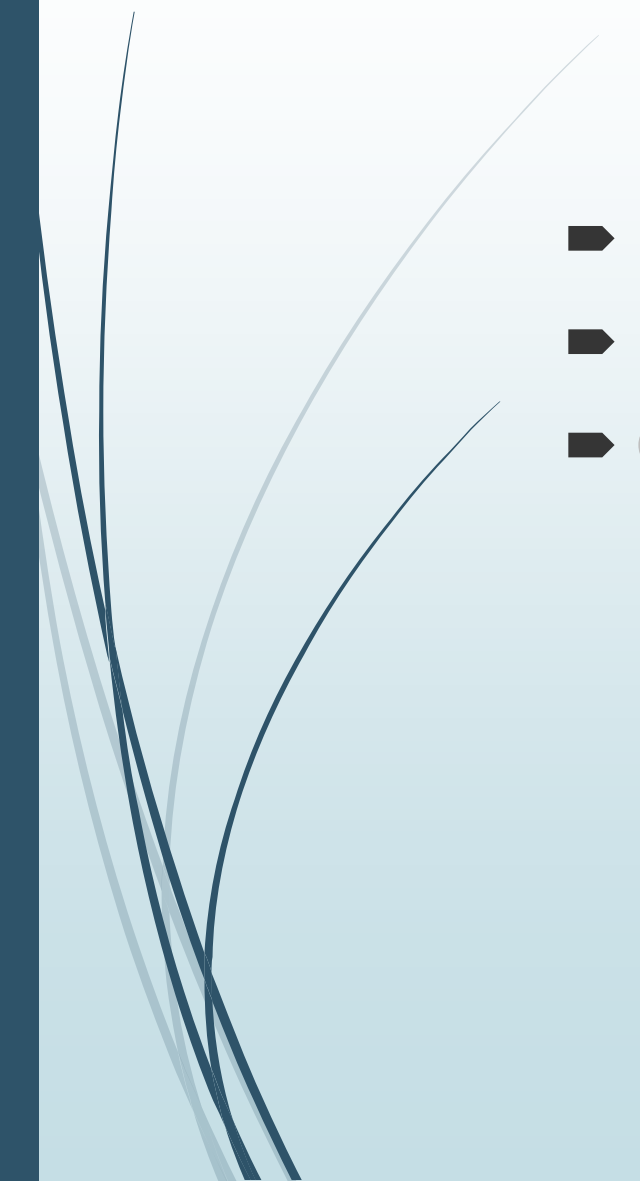


Session 5

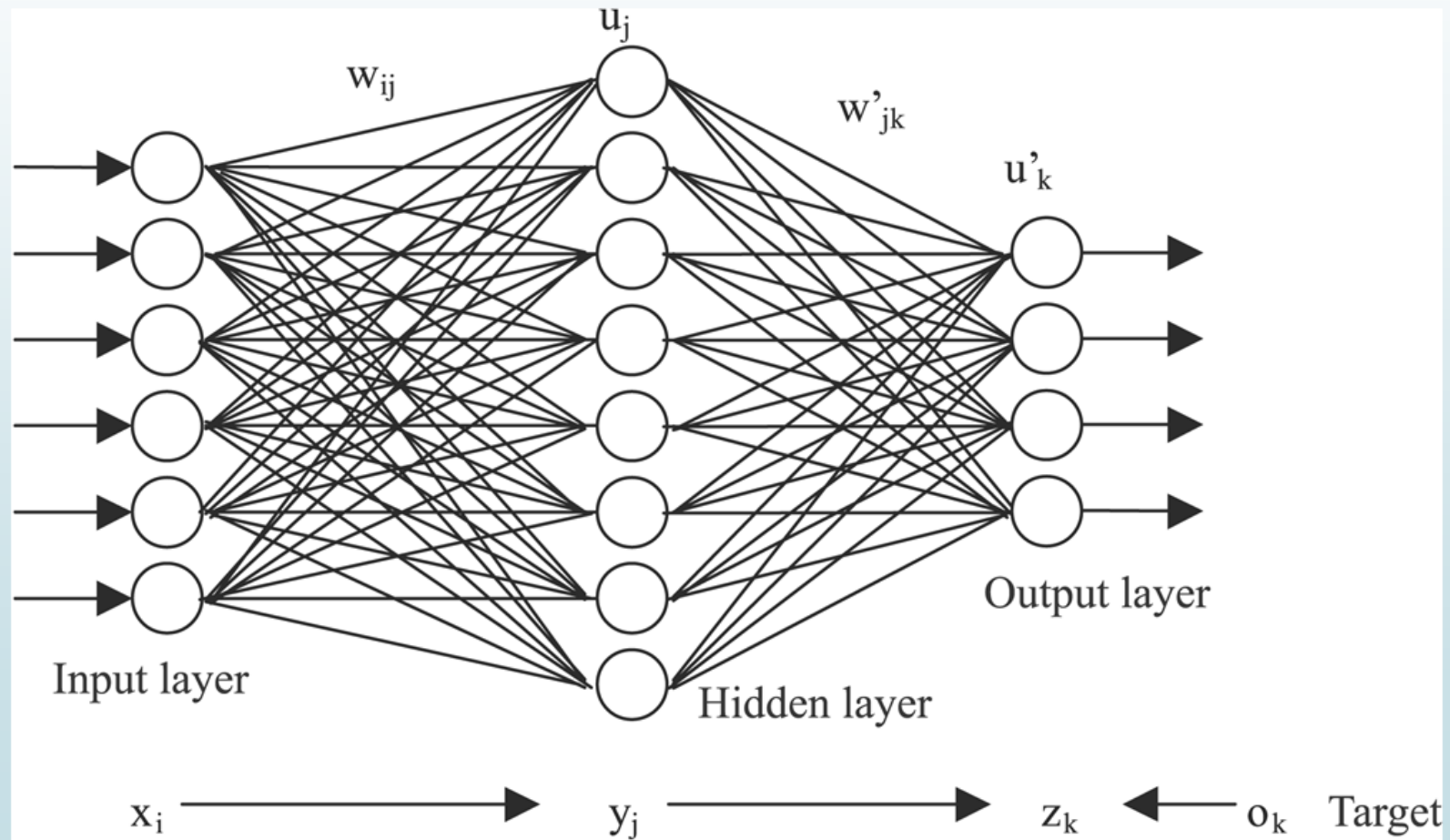
- 
- Recap of Session 4
 - Feedforward neural net
 - Convolutional Neural Net



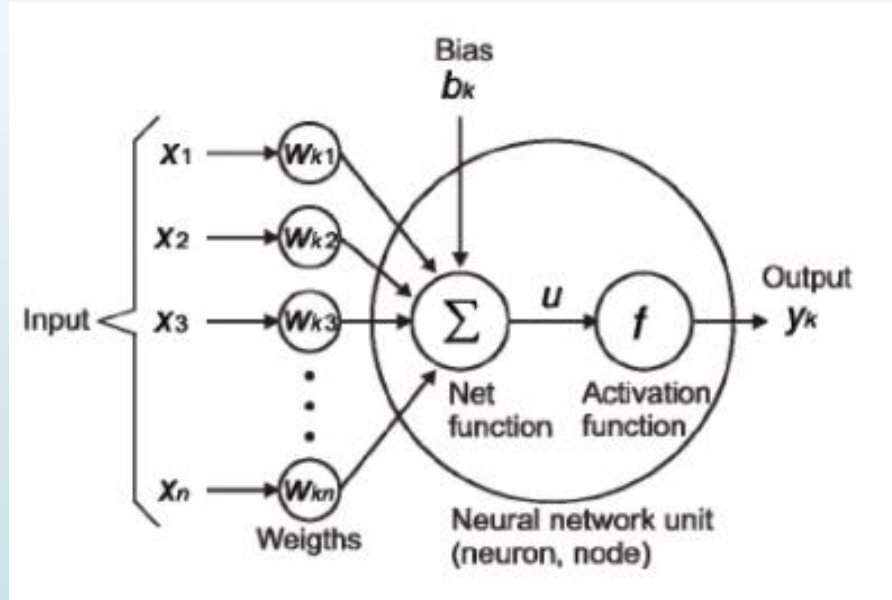
Session 5

- Recap of Session 4
 - Feedforward neural net
 - Convolutional Neural Net
- 

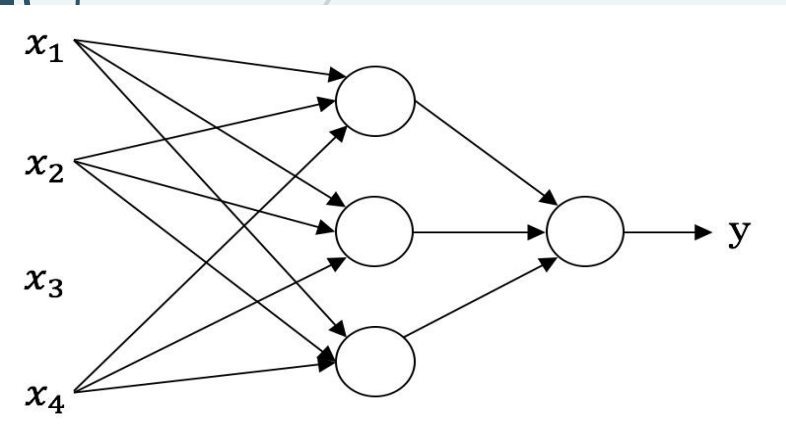
What is a neural network



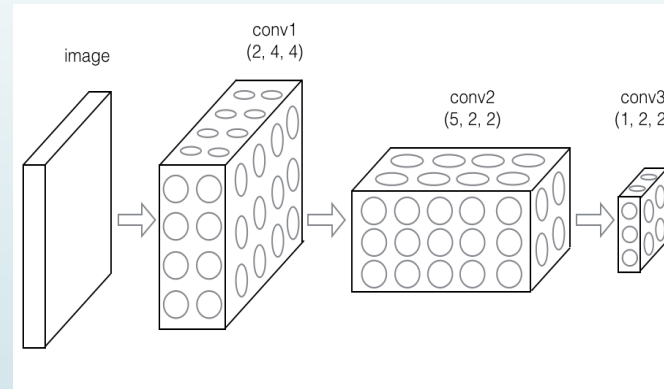
What is a neural network



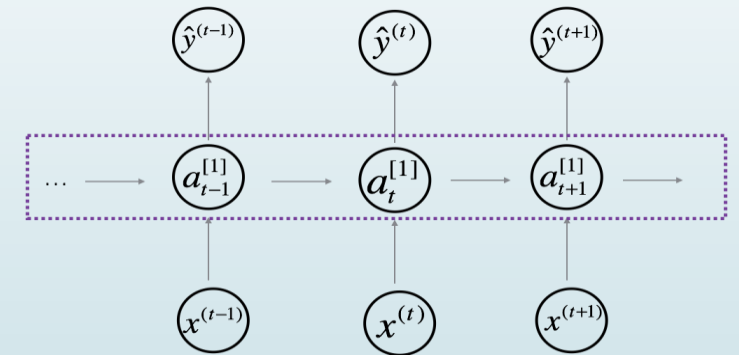
Types of Neural Networks



Standard
NN



Convolutional
NN



Recurrent
NN

Neural Networks

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Our Model of Neural Network

$$\hat{y} = \sigma(w^T x + b), \text{ where } \sigma(z) = \frac{1}{1+e^{-z}}$$

Given $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss (error) function:
$$L(w, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Learn(adjust w and b) : Gradient Descent using Back Propagation

Gradient Descent

We want to find w and b such that $L(w,b)$ is minimum

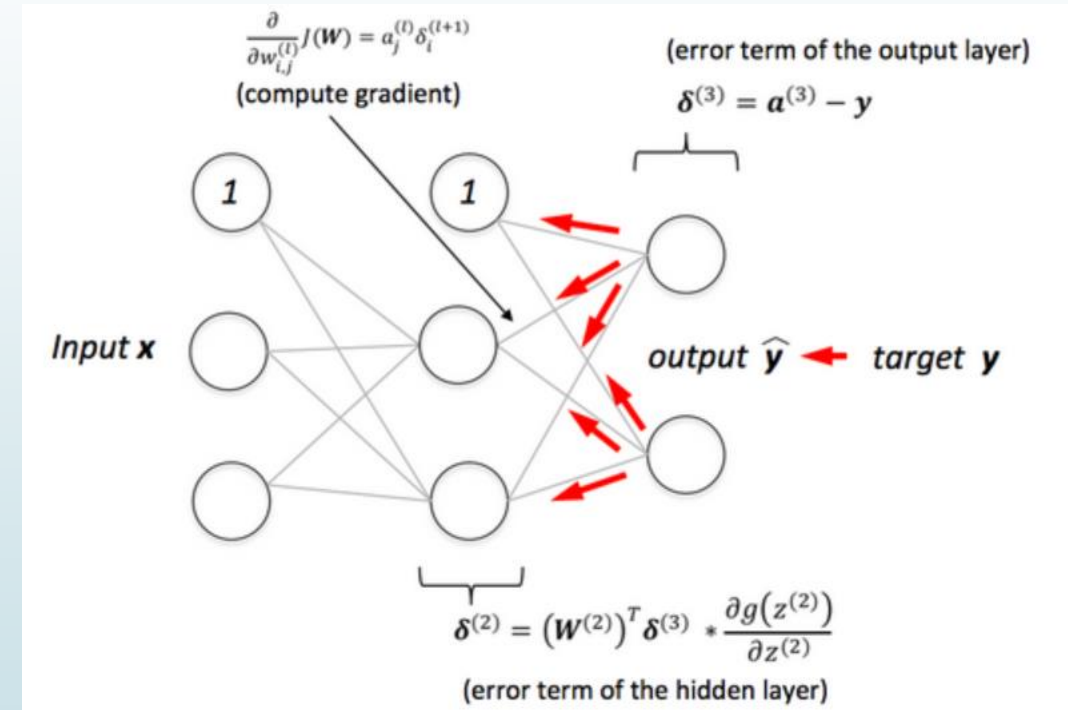
Use gradient Descent:

$$w = w - \alpha \cdot \frac{\partial L(w, b)}{\partial w}$$

$$b = b - \alpha \cdot \frac{\partial L(w, b)}{\partial b}$$

Back Propagation

- Apply chain rule of calculus and propagate error from output layer to input layer.
- It is like a flow graph you flow the error back and try to find the impact of each variable (weight, bias) on the total error
- The most popular algorithm to train neural networks
- “Tensorflow” from Google allows you to build the graph of neural network and apply chain rule with partial derivatives in a systematic and efficient manner.
- This field is also known as “Automatic differentiation”



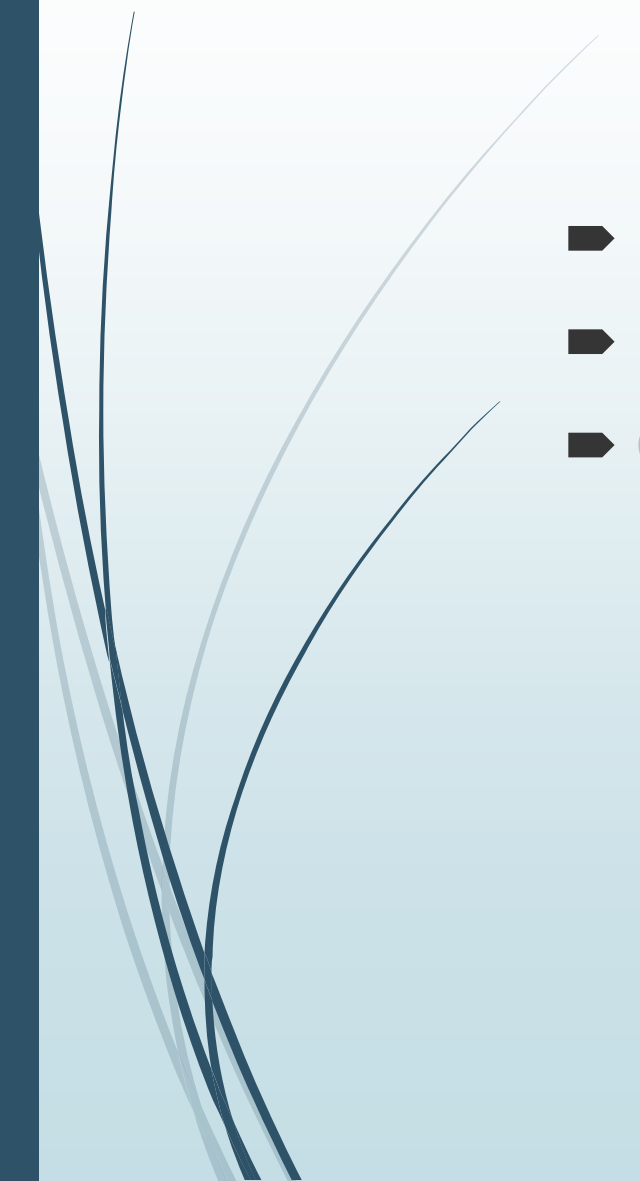


Tensorflow and Keras

- Tensorflow – open source framework from Google to build and train Neural network models.
- Caffe from Facebook is another very popular framework
- MSFT: Microsoft Cognitive Toolkit—previously known as CNTK
- And many more
 - They are try to do the same – build, train deep learning models but vary in the approach they take.
- Keras: is a higher level framework that can sit on top of Caffe, Tensorflow. It abstarcts the nuts and bolts of underlying framework. Same Keras code will work even if you swap the lower level framework.



Session 5

- ▶ Recap of Session 4
 - ▶ Feedforward neural net
 - ▶ Convolutional Neural Net
- 



Feedforward Neural Network

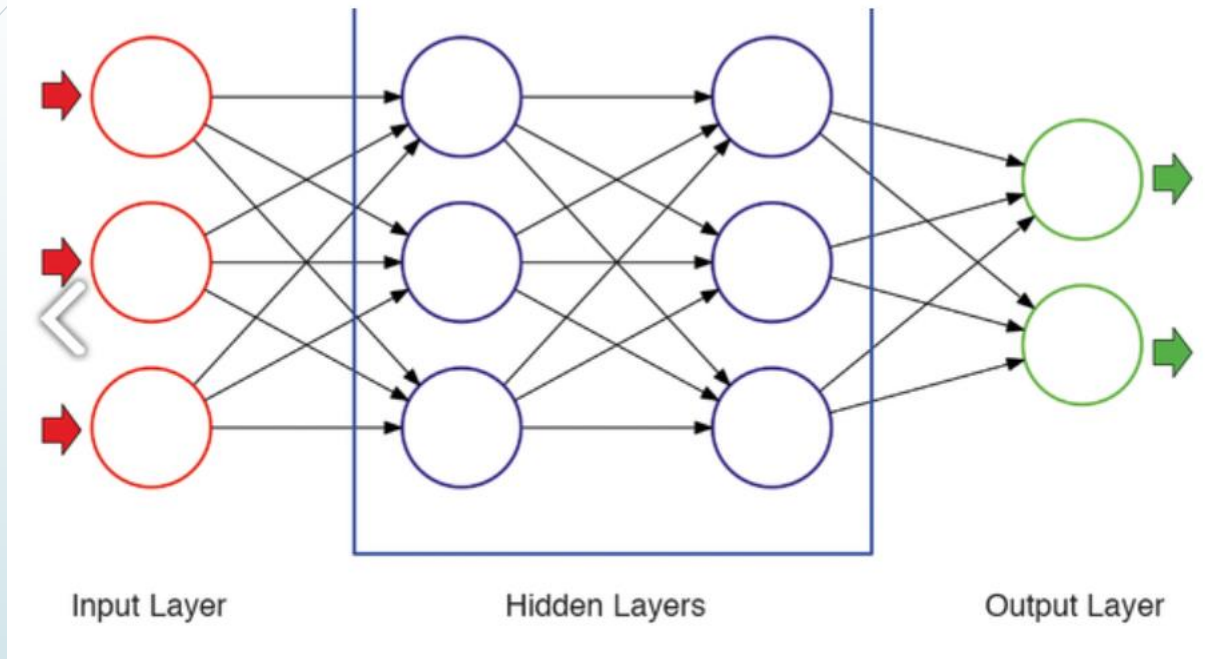
- We will take student admissions data for UCLA
 - Inputs:
 - GRE Scores,
 - GPA Scores,
 - Rank (prestige of undergrad school)
 - Output
 - Yes/No – did student get admission to UCLA grad school



EDA and pre processing

- We do some plots
- We also convert categorical data to One hot encoding
- We normalize the feature values
- We split data into train and test
- We also one hot encode output

Our Neural Network

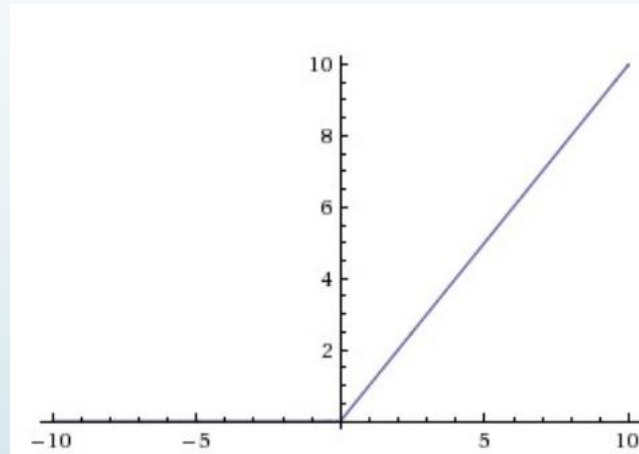


We have

- 6 inputs + one bias
- First hidden layer with 128 nodes
- 2nd Hidden layer has 64 nodes
- Final output layer has two nodes

Activations Non-linearities

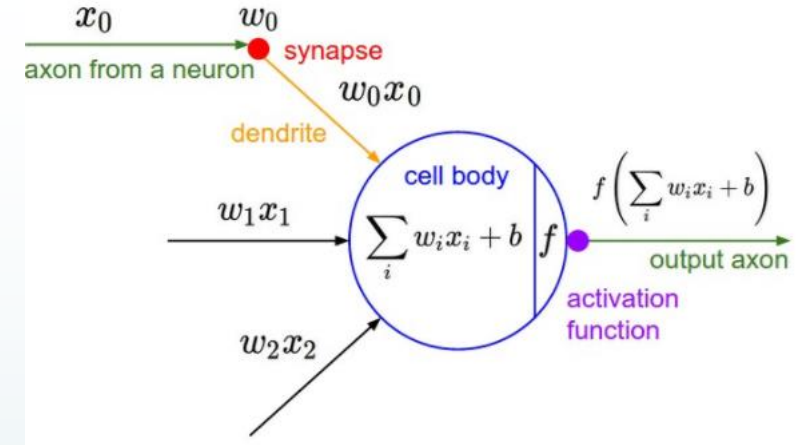
- Hidden Layers - RELU



- Final layer Softmax – converts output to probability score

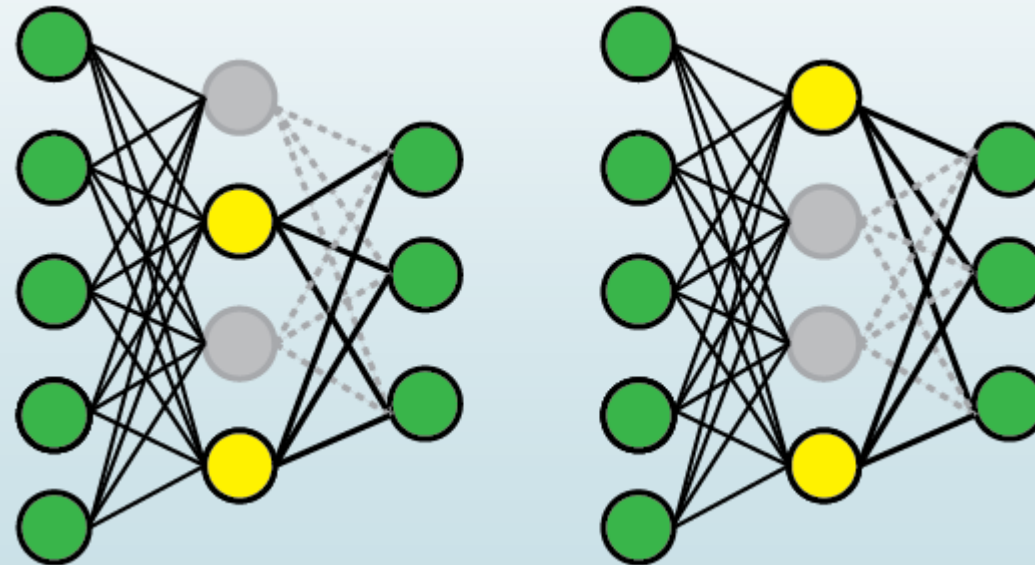
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \rightarrow \text{Softmax} \rightarrow \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$



Dropout Layer

- Neural nets tend to overfit esp with small data set
- Recently concept called dropout has become a fairly standard way to use dropout for **regularization**





Calculating Number of Params

- Input to Hidden 1
 - $(6+1)*128 = 896$
- Hidden 1 to Hidden 2
 - $(128+1)*64 = 8256$
- Hidden 2 to Output
 - $(64+1)*2 = 130$



Loss

- As we are doing a classification, we will use cross-entropy loss. Tuning the parameters (learning) will try to minimize this loss

$$L(w, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

Optimizer

- State of art way to train neural networks is SGD with back-propagation
- There are variants of SGD to dynamically adjust the step size of learning i.e. alpha

$$\blacksquare w = w - \alpha \cdot \frac{\partial L(w,b)}{\partial w}$$

- There are many approaches to this and new ones are coming
- Neural network optimization is not a convex problem and hence you finally need to explore the space of parameter values in an intelligent way to find “best” combination that minimizes loss.
- “Adam” is a fairly popular one
- Read this link if you want to dig deep - <http://ruder.io/optimizing-gradient-descent/>

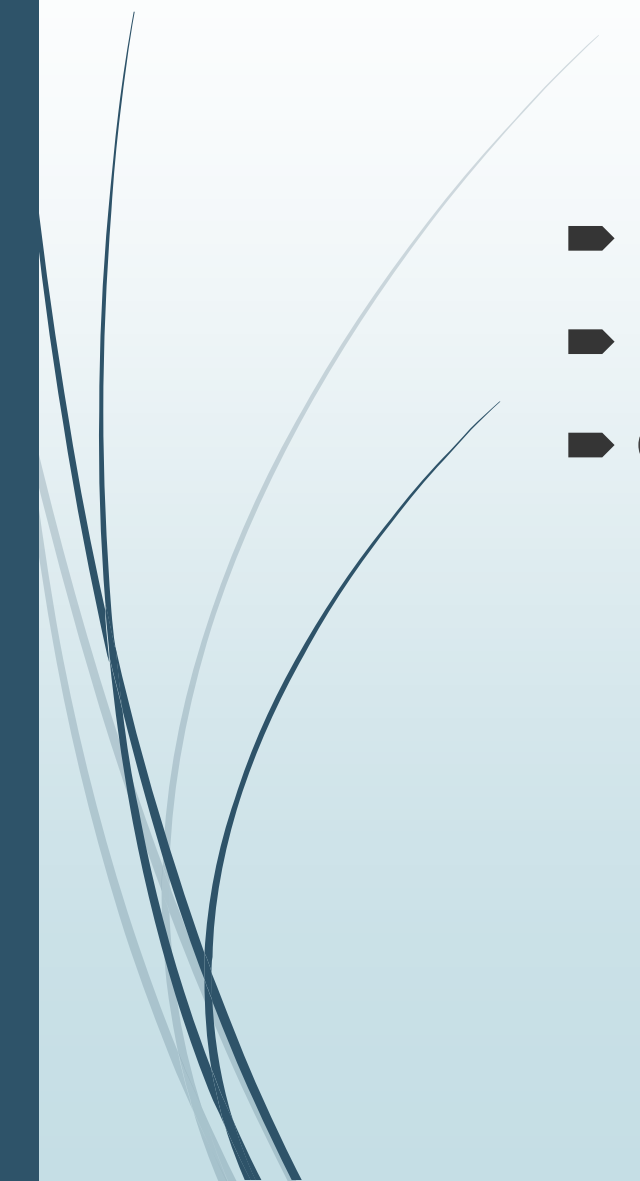


Keras

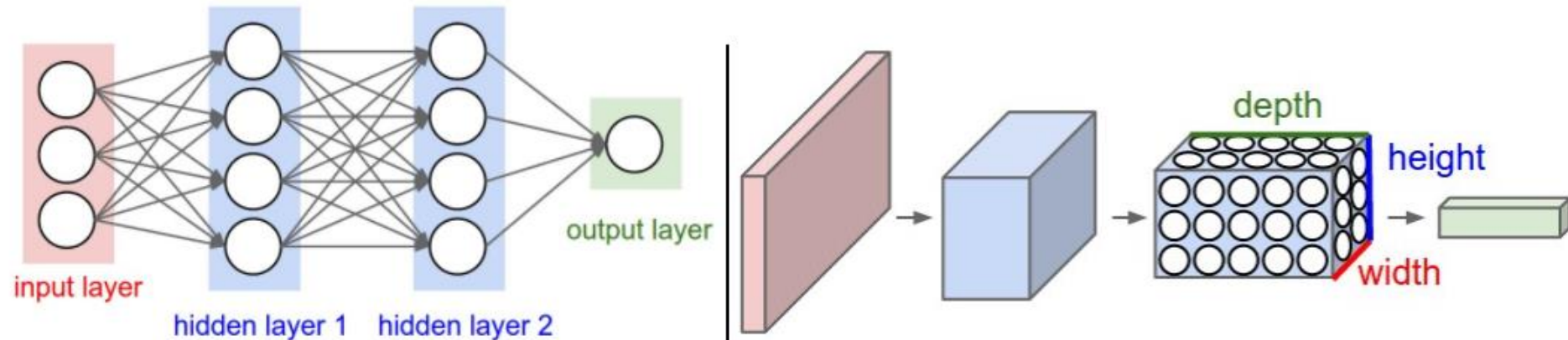
- Step1 – prepare your data
- Step2 – define your model
- Step3 – fit/train the model using training data
- Step4 – check the quality (generalization) using test data
- My setup uses tensorflow behind the scene with Keras sitting on top of it



Session 5

- Recap of Session 4
 - Feedforward neural net
 - Convolutional Neural Net
- 

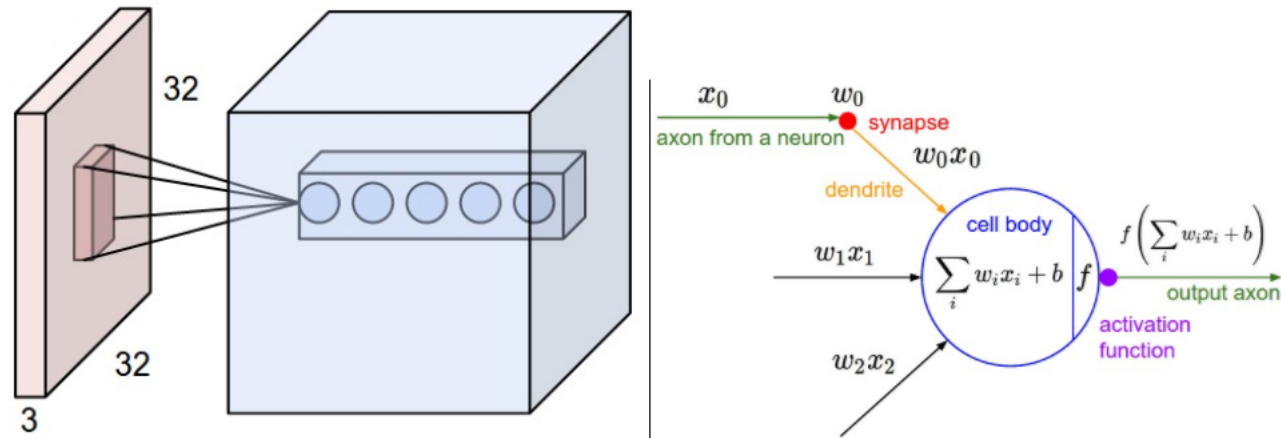
What is Convolutional NN



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Demo

➡ <http://cs231n.github.io/convolutional-networks/>



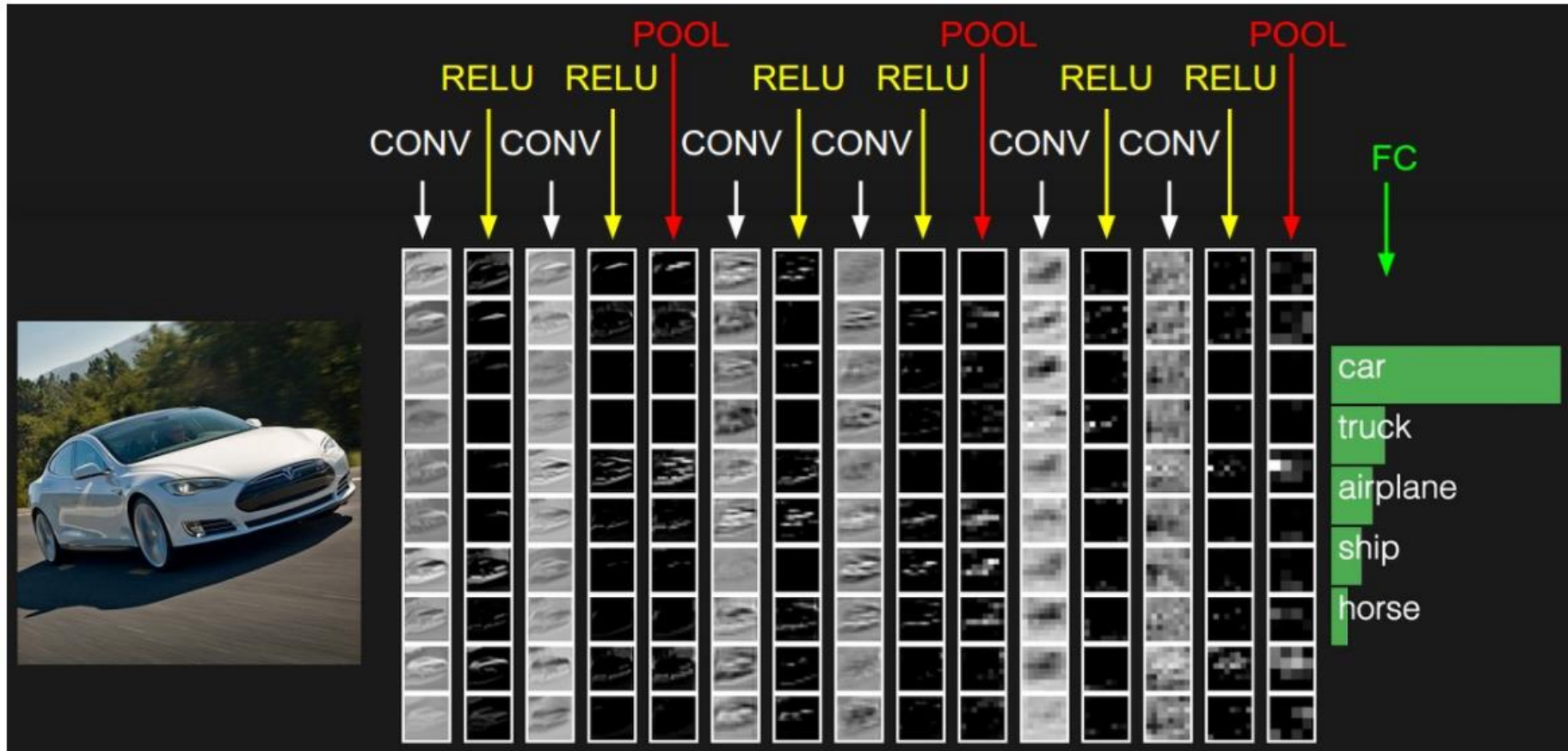
Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. **Right:** The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.



Demo

- <http://cs231n.github.io/convolutional-networks/>
 - Parameter sharing
 - Filters
 - Size
 - Depth
 - Strides
 - Number of filters
 - Strategy to handle edges (same/valid)

What is each layer doing?



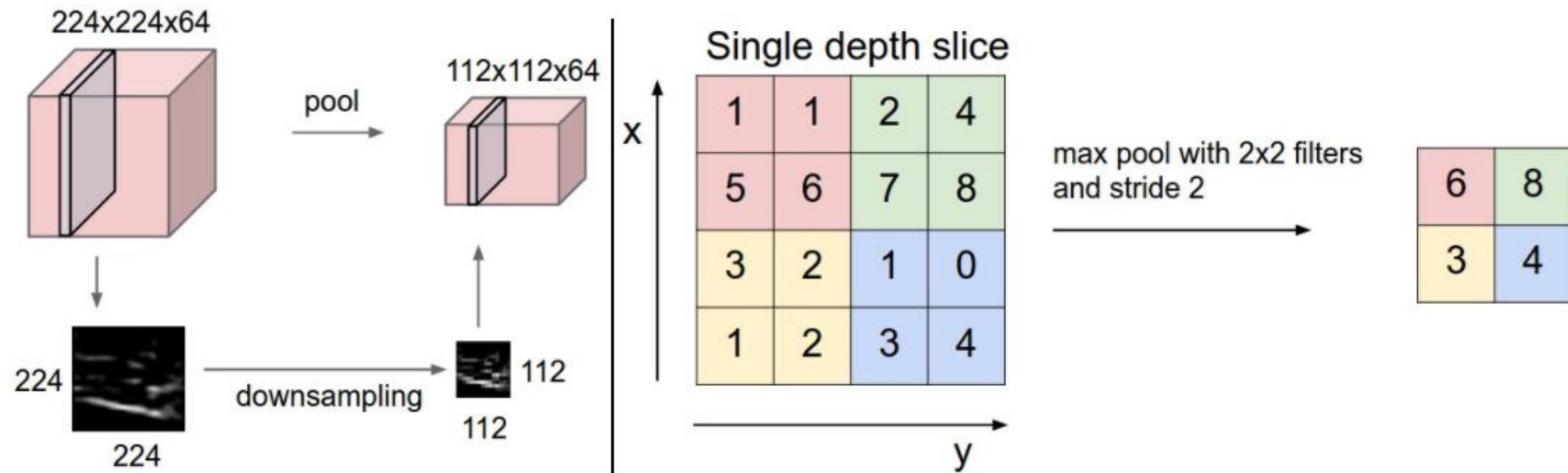
The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full [web-based demo](#) is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.



New types of layers

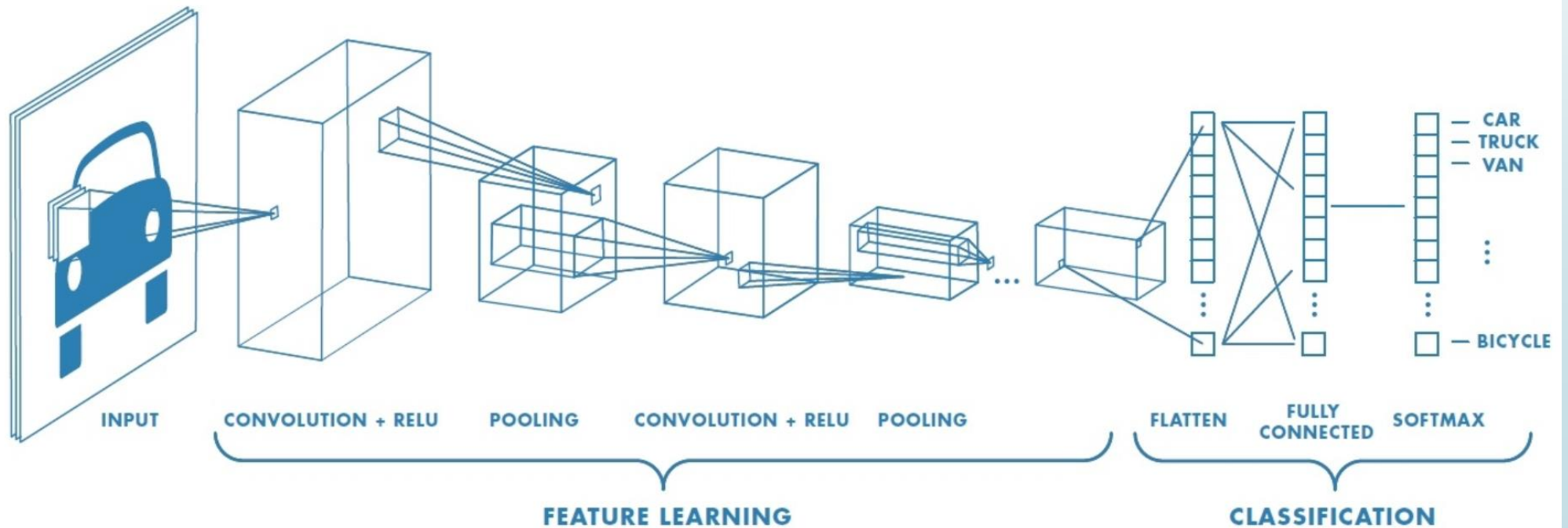
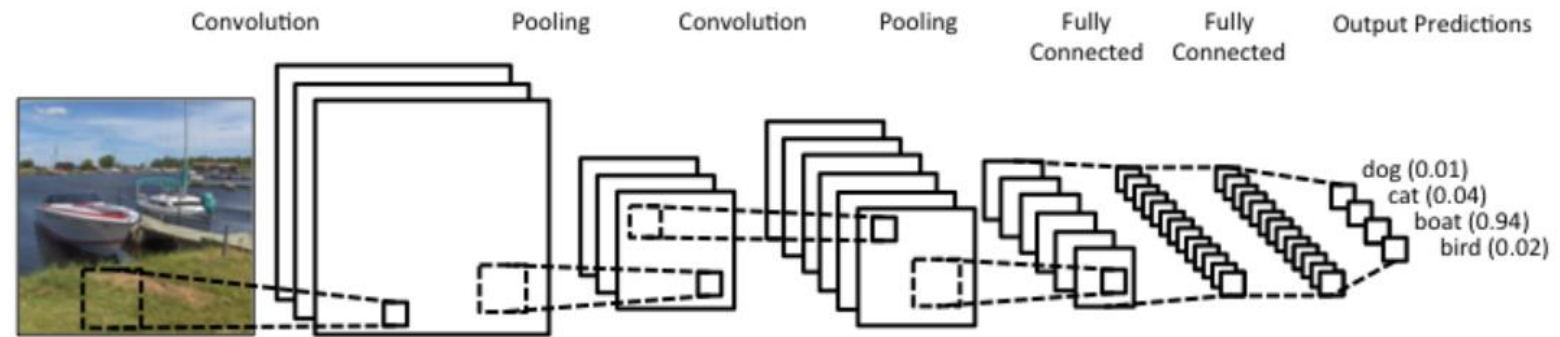
- Convolutional Layer – we just saw
- Dropout layers – we talked in previous section
- Activation functions/layers
 - RELU – most common for hidden layers
 - Softmax – as output activation - most of classification problems
- Pooling layers
- Flatten and then FC (fully connected layers)

Pooling layers



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size $[224 \times 224 \times 64]$ is pooled with filter size 2, stride 2 into output volume of size $[112 \times 112 \times 64]$. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2×2 square).

Flatten followed by FC





Why are NN so popular and powerful?

**They learn features on their own.
No need for extensive feature engineering
that was required earlier**