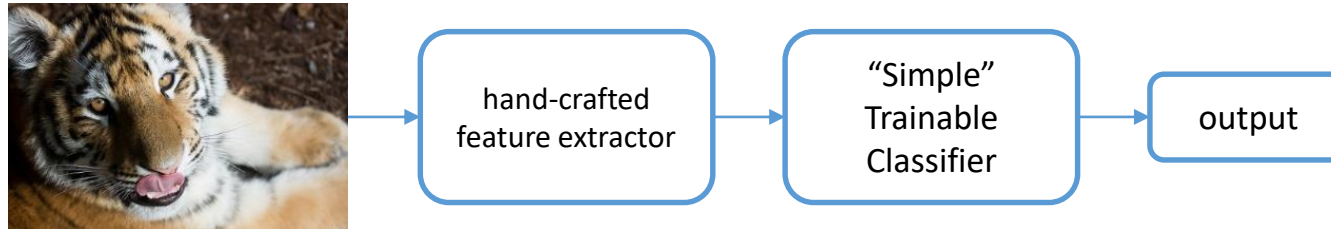


NEED FOR DEEP LEARNING

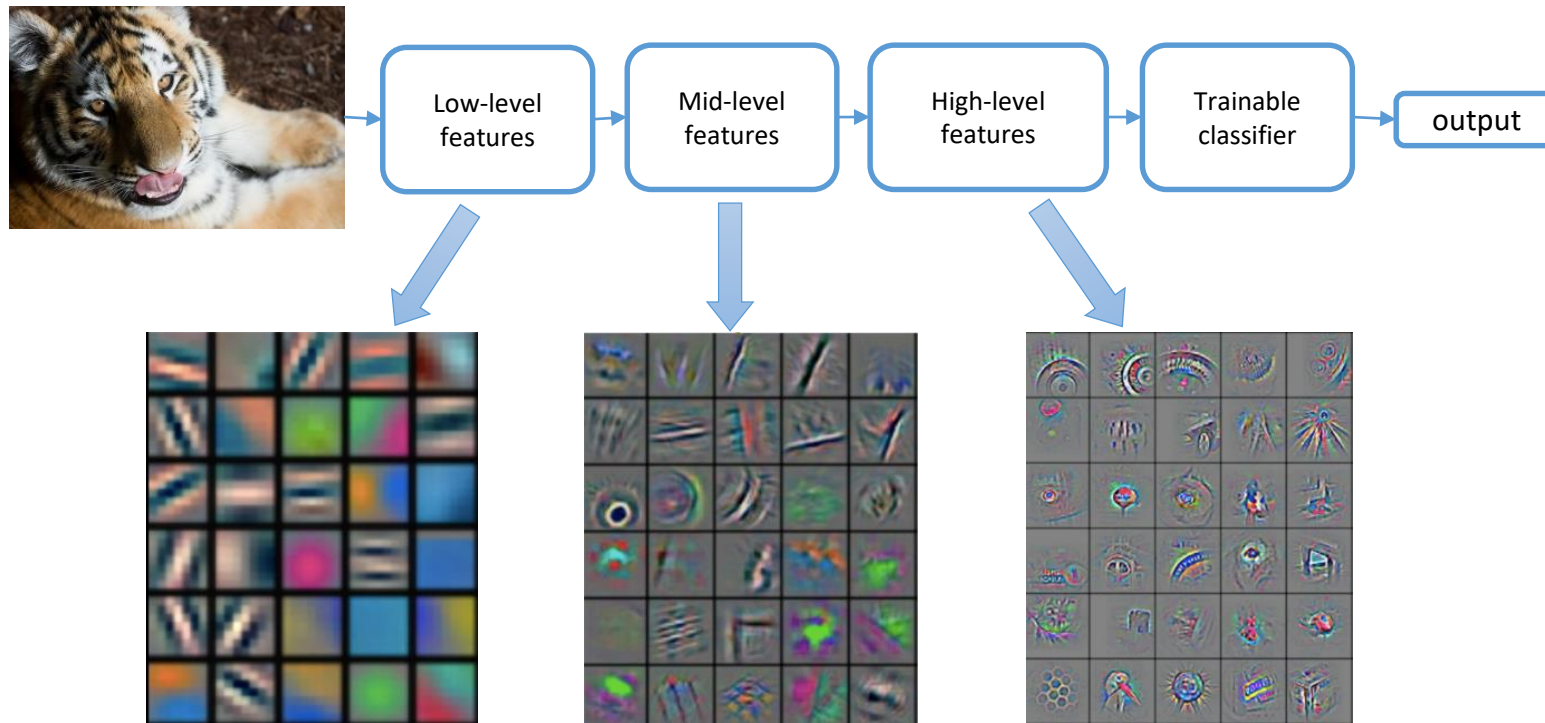
- Traditional pattern recognition models use hand-crafted features and relatively simple trainable classifier.



- This approach has the following limitations:
 - It is very tedious and costly to develop hand-crafted features
 - The hand-crafted features are usually highly dependent on one application, and cannot be transferred easily to other applications

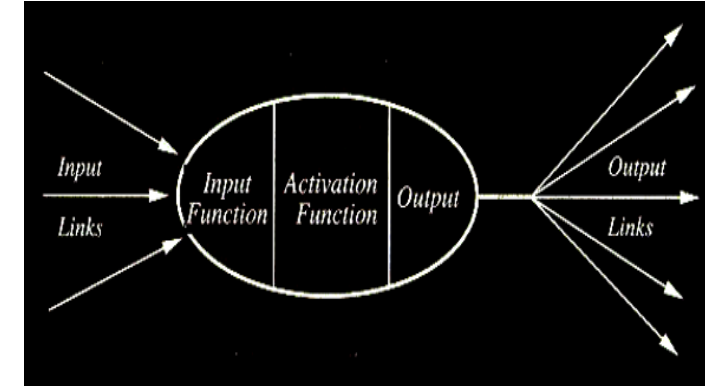
DEEP LEARNING

- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.

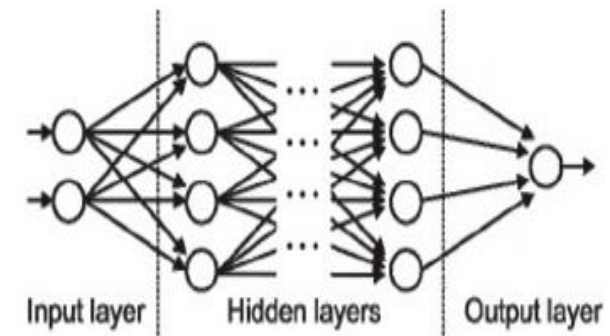


Key Concepts

➤ **Neurons:** Neurons are the fundamental building blocks of a neural network. Neurons are used for receiving input, processing input and generating output. Artificial neural networks are made up of neurons. Depending on the layer in which neurons are present they perform different functionalities.



➤ **Layers(Input/hidden/output):** The neurons are organized in the form of layers in an ANN. All layers in a neural network can be segregated as either a input layer, hidden layer or an output layer. The input layer receives the input and is the first layer of the neural network. The hidden layers are the processing layers that learn the features or patterns for mapping the input to the output layer. The output layer is the final layer that generates the output.

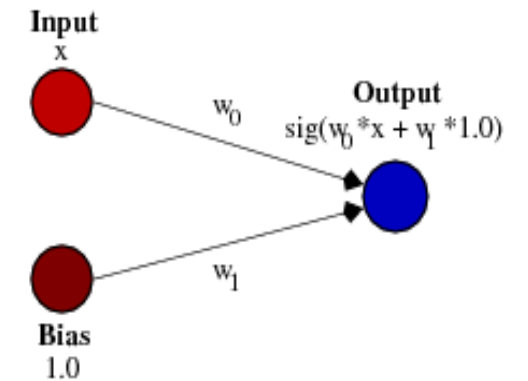


Key Concepts

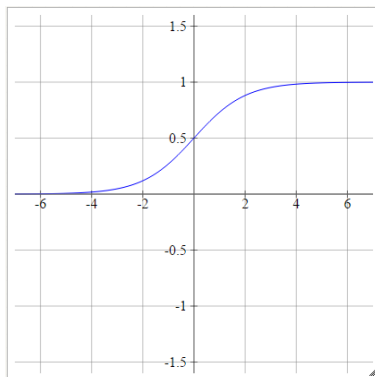
➤ **Weights and biases:** The neurons in a neural network are connected and each connection has an associated weight assigned to it. We initialize the weights randomly and these weights are updated during the model training process.

Biases are analogous to the intercept in a regression model.

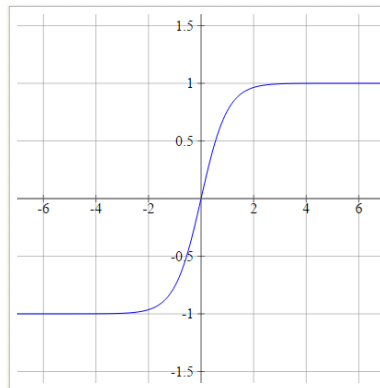
➤ **Activation functions:** Activation functions introduce non-linearity into neural networks. Without these activation functions, neural networks will be very similar to that of a linear model. Commonly used activation functions include sigmoid, tanh, ReLu and softmax.



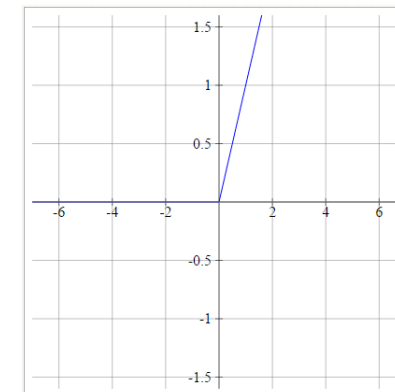
$$\text{Sigmoid} = 1/(1+e^{(-x)})$$



$$\text{Tanh} = (e^x - e^{-x}) / (e^x + e^{-x})$$

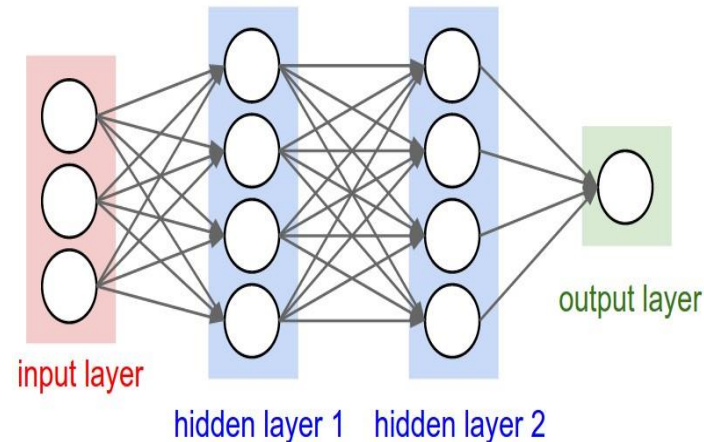


$$\text{ReLu} = \max(0, x)$$



Key Concepts

➤ **Multi-Layer Perceptron's (MLP):** A single neuron would not be able to perform highly complex tasks. Therefore, we use stacks of neurons to generate the desired outputs. In the simplest network we would have an input layer, a hidden layer and an output layer. Each layer has multiple neurons and all the neurons in each layer are connected to all the neurons in the next layer.



➤ **Training the neural network(Forward Pass/ Backward Pass):** After configuring the neural network, the training samples are passed to the neural network with the input and target(i.e Forward pass) and the optimal weights and biases are learnt by the network(i.e. Backward pass).

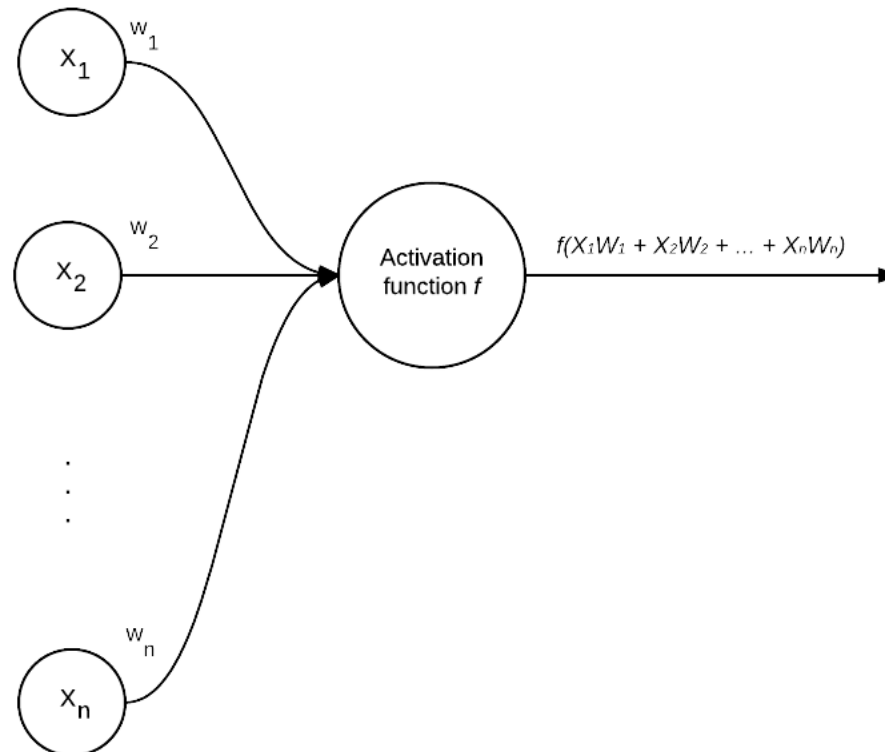
Key Concepts

- **Cost function:** Cost function measures how close the model's prediction is to the actual value. A higher value indicates that the model's predictions are not accurate. The objective of the neural network is to minimize this cost function, thereby reducing the error.
- **Backpropagation:** Backpropagation is a technique that is used for training a neural network. Initially, random weights and biases are assigned to the neural network. Once the training process starts, the weights and biases are updated based in the direction of the gradient.
- **Learning rate:** This parameter determines how fast or slow we will move towards the optimal weights. If the learning rate is very large we will skip the optimal solution. If it is too small we will need too many iterations to converge to the best values.
- **Optimizers:** Optimizers are the algorithms that are used for calculating the gradients and the parameter updates with the aim of reducing the cost function. Commonly used optimizers are stochastic gradient descent, adam, adagrad and rmsprop.
- **Regularizers:** Sometimes the neurons in the neural network learn the exact representation of the training data. When this happens the network will not perform well on unseen data. To avoid such an overfitting, regularizers are used. Popular regularizers include dropouts and batch normalization.
- **Epochs:** One forward and backward pass of all training examples.

NEURAL NETWORKS

- Neural networks are the building blocks of deep learning systems.
- Each of us contains a real-life biological neural network that is connected to our nervous system — this network is made up of a large number of interconnected neurons (nerve cells).
- The word “neural” is the adjective form of “neuron”, and “network” denotes a graph-like structure; therefore, an Artificial Neural Network is a computation system that attempts to mimic the neural connections in our nervous system.

- Each node performs a simple computation. Each connection then carries a “signal” (i.e., the output of the computation) from one node to another, labeled by a “weight” indicating the extent to which the signal is amplified or diminished.
- Some connections have large, positive weights that amplify the signal, indicating that the signal is very important in making a classification. Others have negative weights, diminishing the strength of the signal, thus specifying that the output of the node is less important in the final classification.



pixel image




imread



					Blue
				Green	255 134 93 22
			Red	255 134 202 22	2
255	231	42	22	4	30
123	94	83	2	92	124
34	44	187	92	64	142
34	76	232	124	04	
67	83	194	202		



reshaped image vector

$$\begin{pmatrix} 255 \\ 231 \\ 42 \\ 22 \\ 123 \\ 94 \\ \vdots \\ \vdots \\ 92 \\ 142 \end{pmatrix}$$


- The values X_1, X_2, \dots, X_n are the inputs to our NN. These inputs could be raw pixel intensities or the entries in a feature vector.
- Each X is connected to a neuron via a weight vector, W , consisting of w_1, w_2, \dots, w_n . This means that for each input x , we also have an associated weight w .

$$f(w_1x_1 + w_2x_2 + \dots + w_nx_n)$$

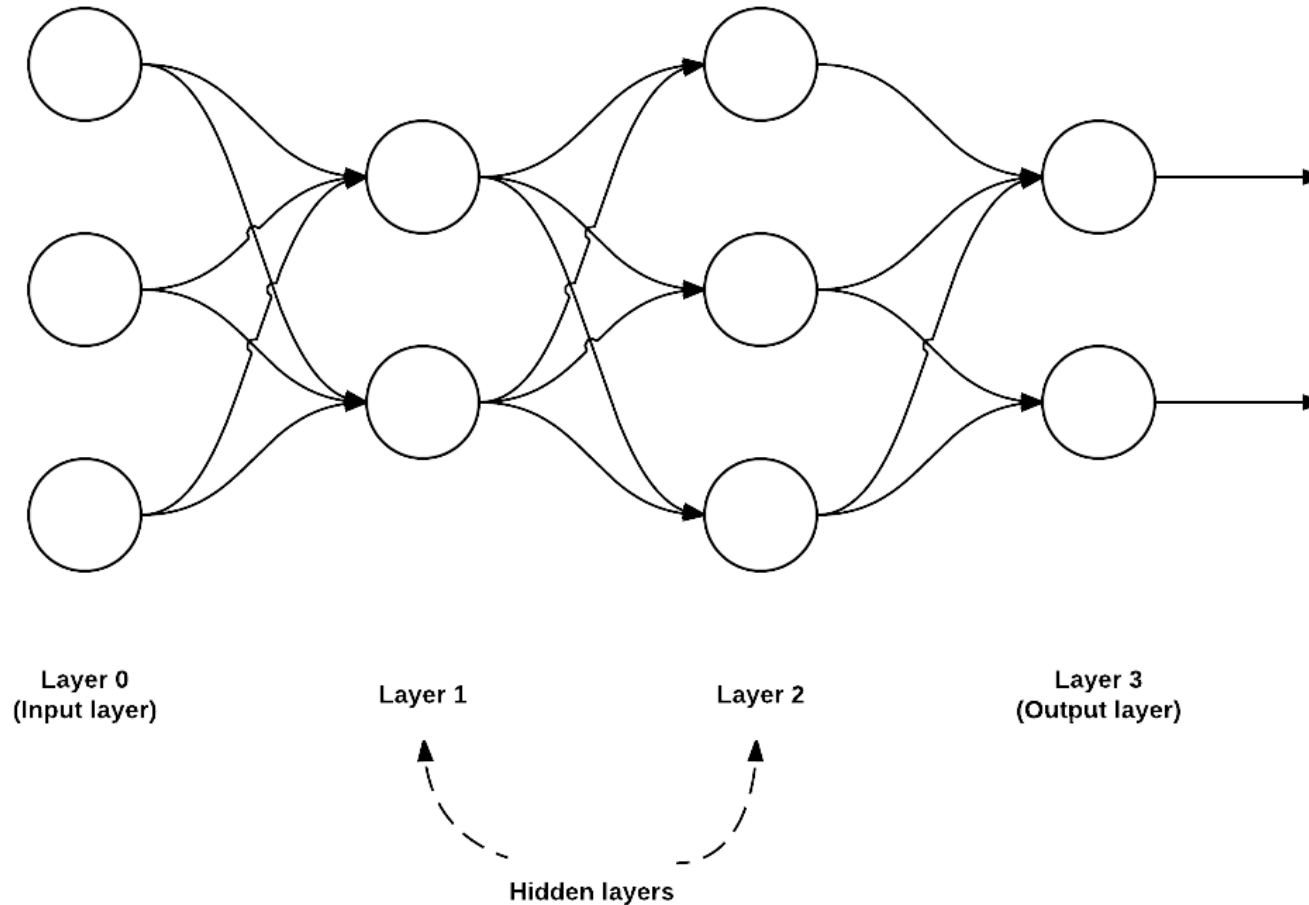
$$net = \sum_{i=1}^n w_i x_i$$

$$f\left(\sum_{i=1}^n w_i x_i\right)$$

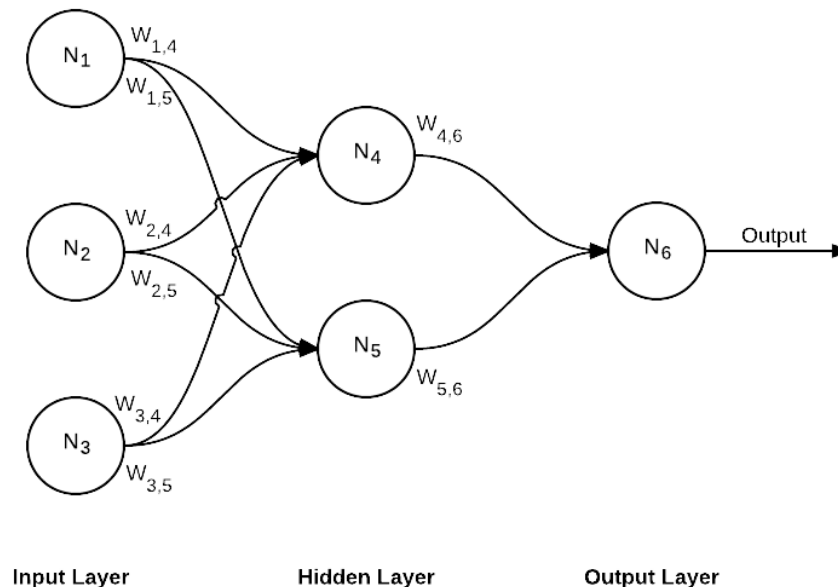
$$f(net) = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

NETWORK ARCHITECTURE

- While there are many, many different NN architectures, the most common architecture is the feedforward network.



- In this type of architecture, a connection between nodes is only allowed from nodes in layer i to nodes in layer $i + 1$ (hence the term feedforward; there are no backwards or inter-layer connections allowed).
- **Layer 0** contains 3 inputs, our values. These could be pixel intensities or entries from a feature vector.
- **Layers 1 and 2** are **hidden layers** containing 2 and 3 nodes, respectively.
- **Layer 3** is the **output layer** or the **visible layer**



The computation hidden layer neurons are N4 and N5.

$$N_4 = f(w_{1,4} \times N_1 + w_{2,4} \times N_2 + w_{3,4} \times N_3)$$

$$N_5 = f(w_{1,5} \times N_1 + w_{2,5} \times N_2 + w_{3,5} \times N_3)$$

computation output layer neuron is O.

$$o = N_6 = f(w_{4,6} \times N_4 + w_{5,6} \times N_5)$$

Loss function computation.

$$err = y - o$$



image2vector
standardize

x_0

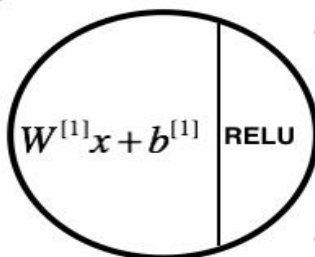
x_1

...

x_{12286}

x_{12287}

Linear Relu



...

$a_0^{[1]}$

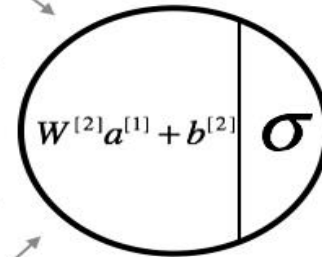
$a_1^{[1]}$

...

$a_{n^{[1]}-2}^{[1]}$

$a_{n^{[1]}-1}^{[1]}$

Linear Sigmoid



→ 0.73

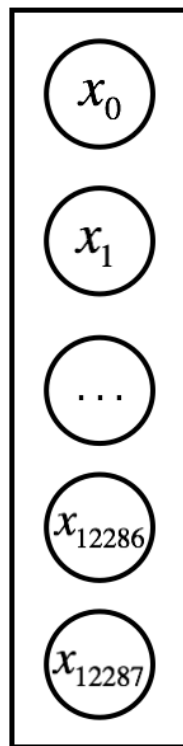
0.73 > 0.5
probability cat
more than
probability non-cat

“it’s a cat”

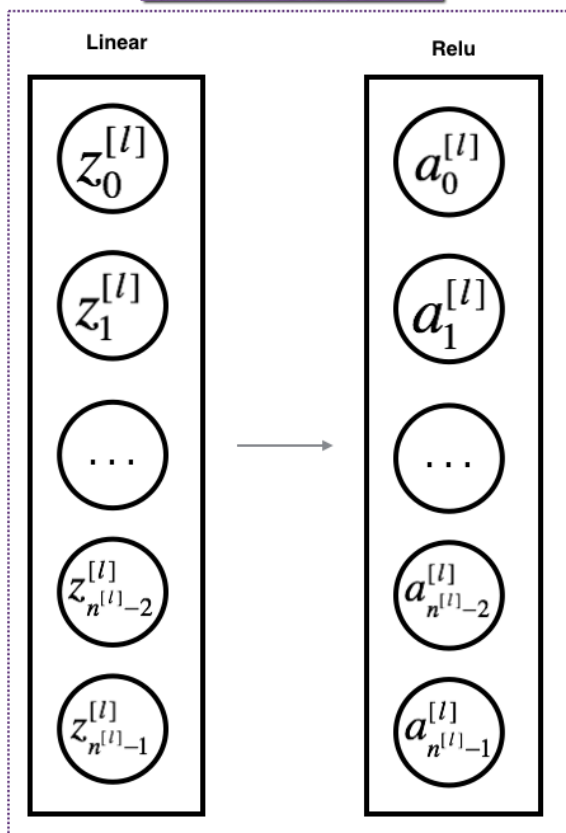


image2vector
standardize

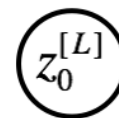
x



repeat for l from 1 to $L-1$



Linear



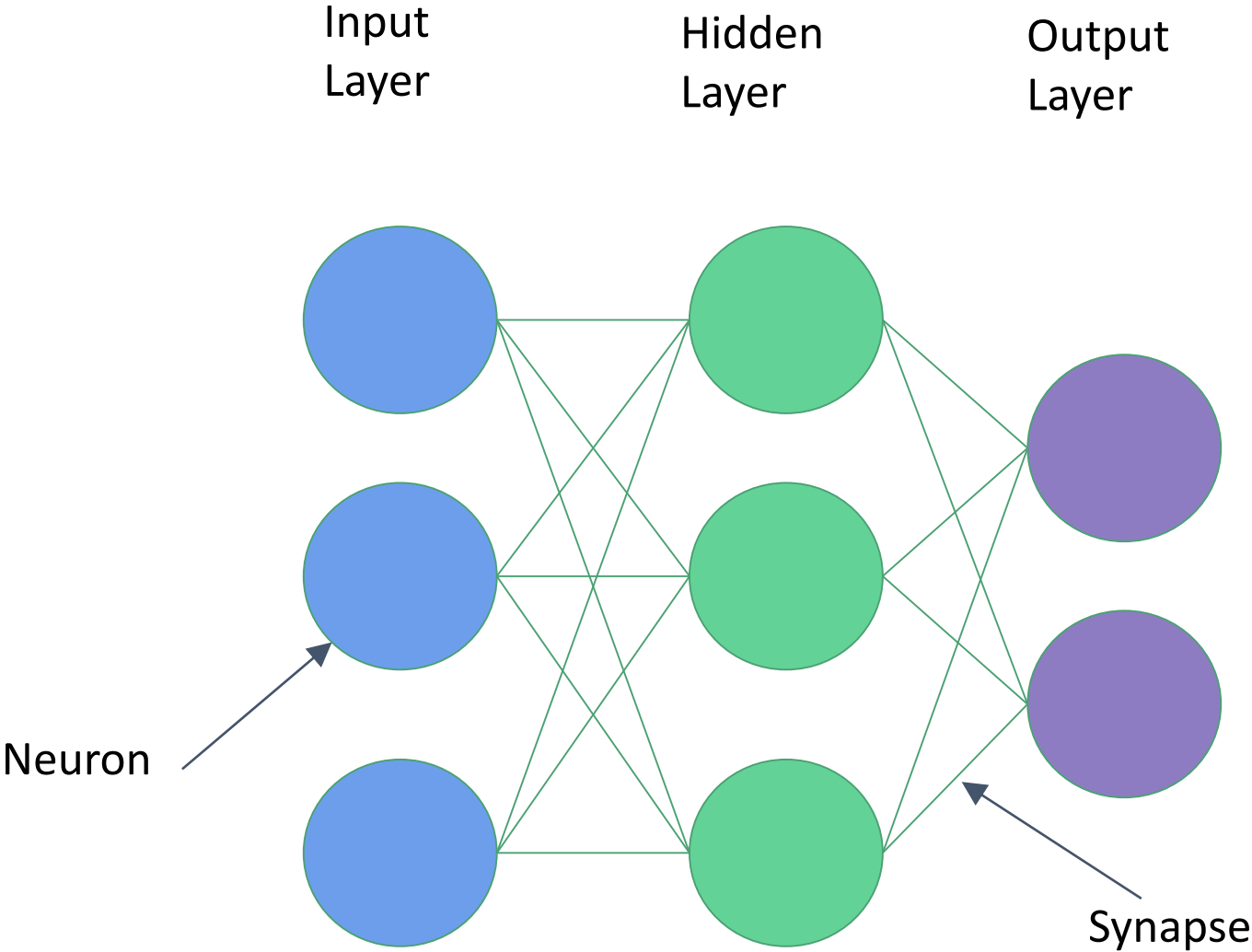
"it's a cat"

$0.73 > 0.5$
probability cat
more than
probability non-cat

0.73

Sigmoid

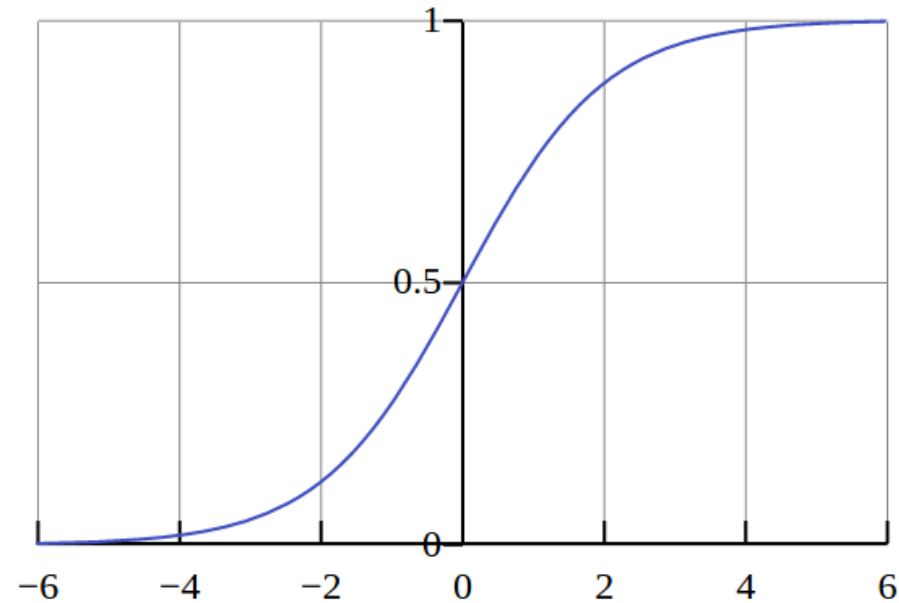
Neural Network Architecture



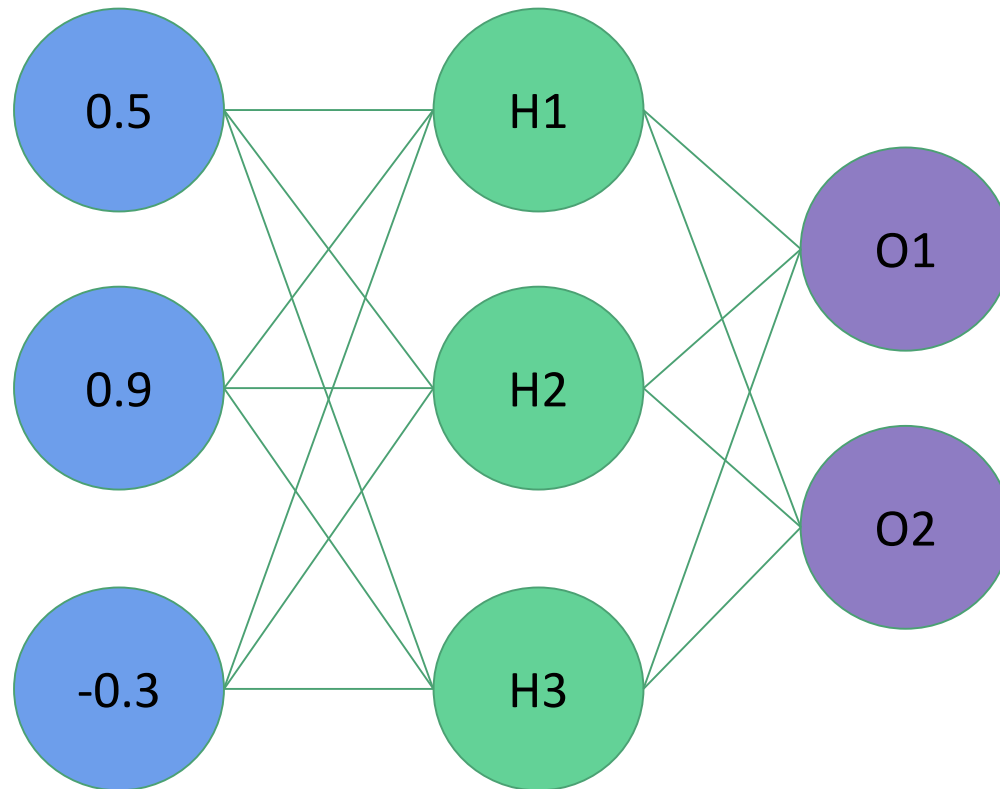
Activation Functions

- Activation Functions are applied to the inputs at each neuron
 - A common activation function is the Sigmoid

$$S(t) = \frac{1}{1 + e^{-t}}$$



Inference



H1 Weights = (1.0, -2.0, 2.0)

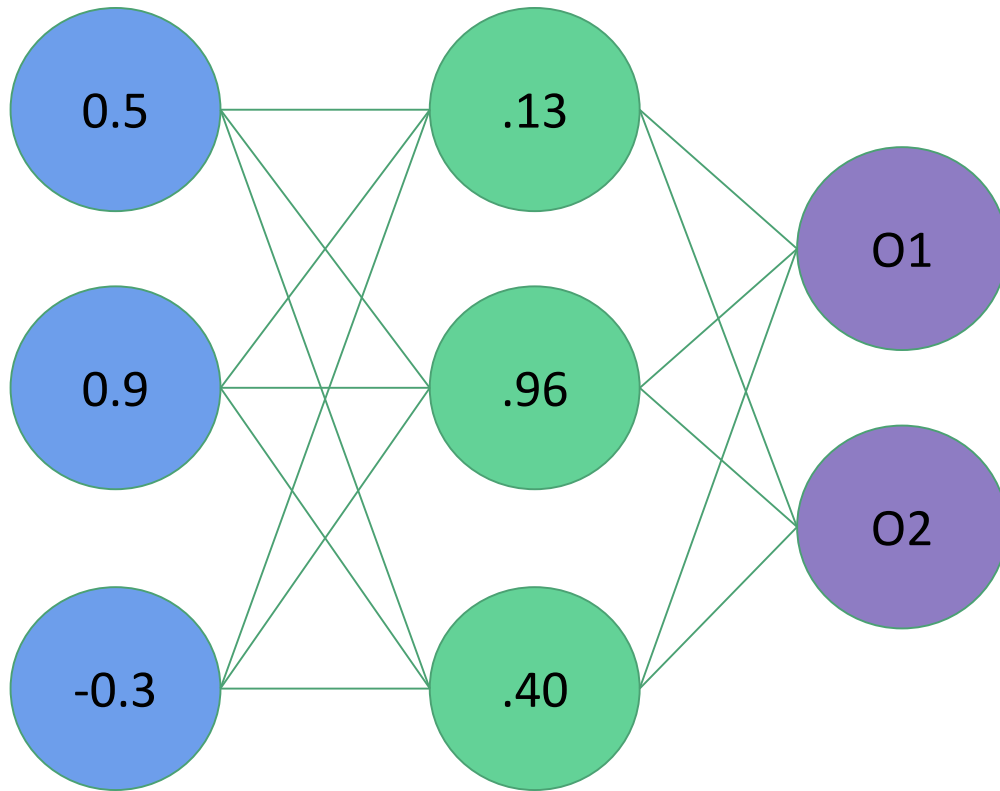
H2 Weights = (2.0, 1.0, -4.0)

H3 Weights = (1.0, -1.0, 0.0)

O1 Weights = (-3.0, 1.0, -3.0)

O2 Weights = (0.0, 1.0, 2.0)

Inference



H1 Weights = (1.0, -2.0, 2.0)

H2 Weights = (2.0, 1.0, -4.0)

H3 Weights = (1.0, -1.0, 0.0)

O1 Weights = (-3.0, 1.0, -3.0)

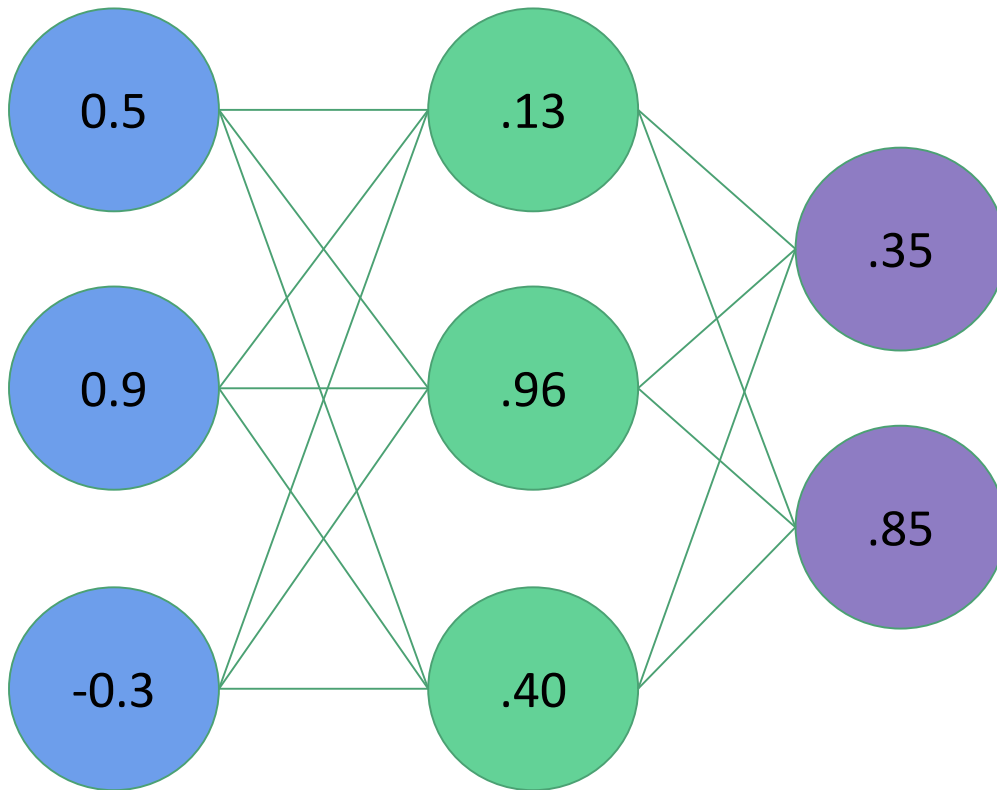
O2 Weights = (0.0, 1.0, 2.0)

$$H1 = S(0.5 * 1.0 + 0.9 * -2.0 + -0.3 * 2.0) = S(-1.9) = .13$$

$$H2 = S(0.5 * 2.0 + 0.9 * 1.0 + -0.3 * -4.0) = S(3.1) = .96$$

$$H3 = S(0.5 * 1.0 + 0.9 * -1.0 + -0.3 * 0.0) = S(-0.4) = .40$$

Inference



H1 Weights = (1.0, -2.0, 2.0)

H2 Weights = (2.0, 1.0, -4.0)

H3 Weights = (1.0, -1.0, 0.0)

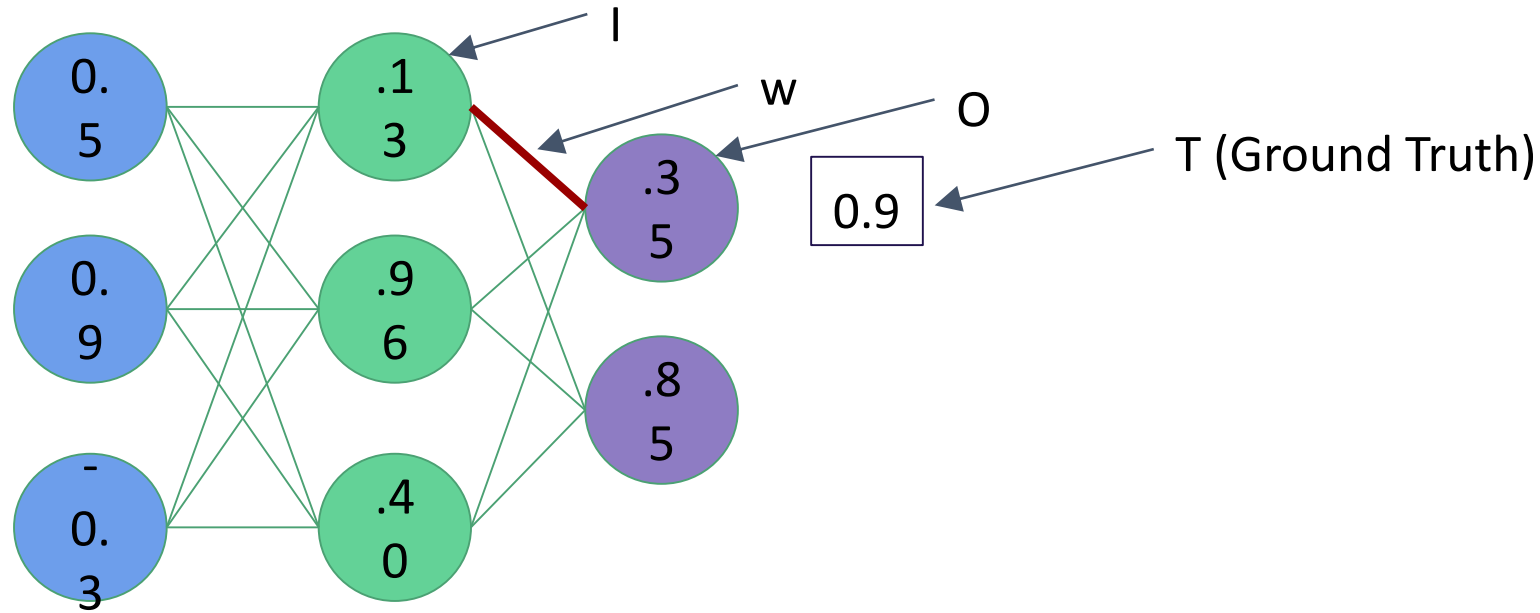
O1 Weights = (-3.0, 1.0, -3.0)

O2 Weights = (0.0, 1.0, 2.0)

$$O1 = S(.13 * -3.0 + .96 * 1.0 + .40 * -3.0) = S(-.63) = .35$$

$$O2 = S(.13 * 0.0 + .96 * 1.0 + .40 * 2.0) = S(1.76) = .85$$

Backpropagation Example



$$\frac{\partial E}{\partial w} = I \cdot (O - T) \cdot O \cdot (1 - O)$$

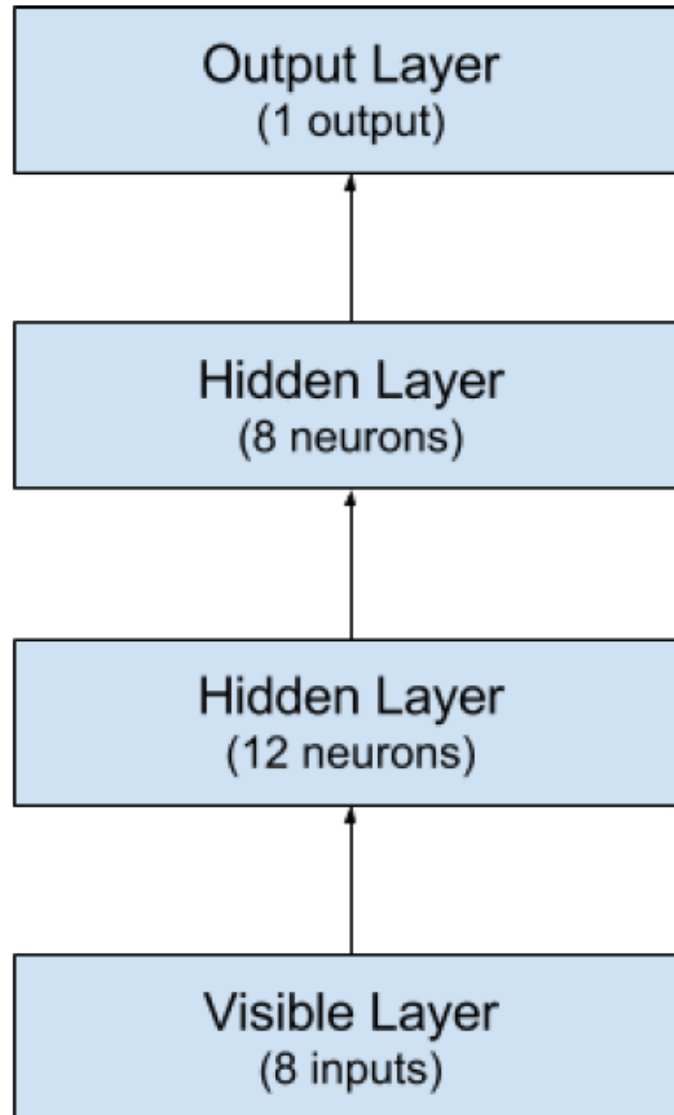
$$\frac{\partial E}{\partial w} = .13 \cdot (.35 - .9) \cdot .35 \cdot (1 - .35)$$

Pima Indians Onset of Diabetes Dataset

1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skin fold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index.
7. Diabetes pedigree function.
8. Age (years).
9. Class, onset of diabetes within five years.

6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
model.add(Dense(8, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, nb_epoch=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

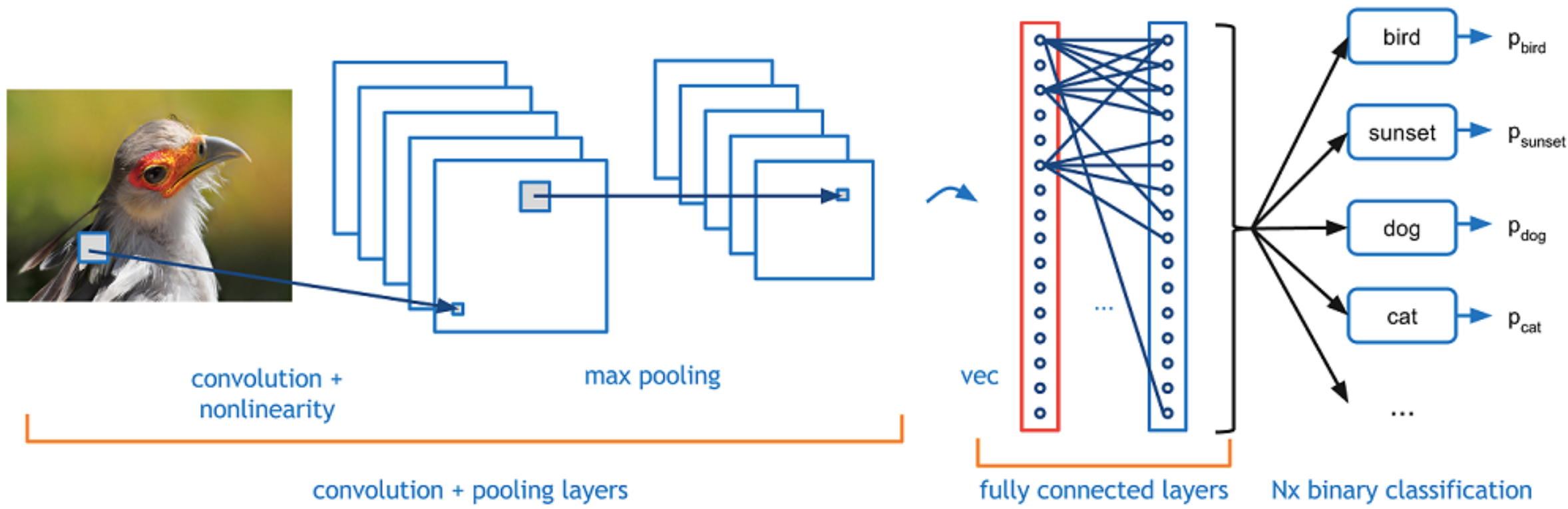


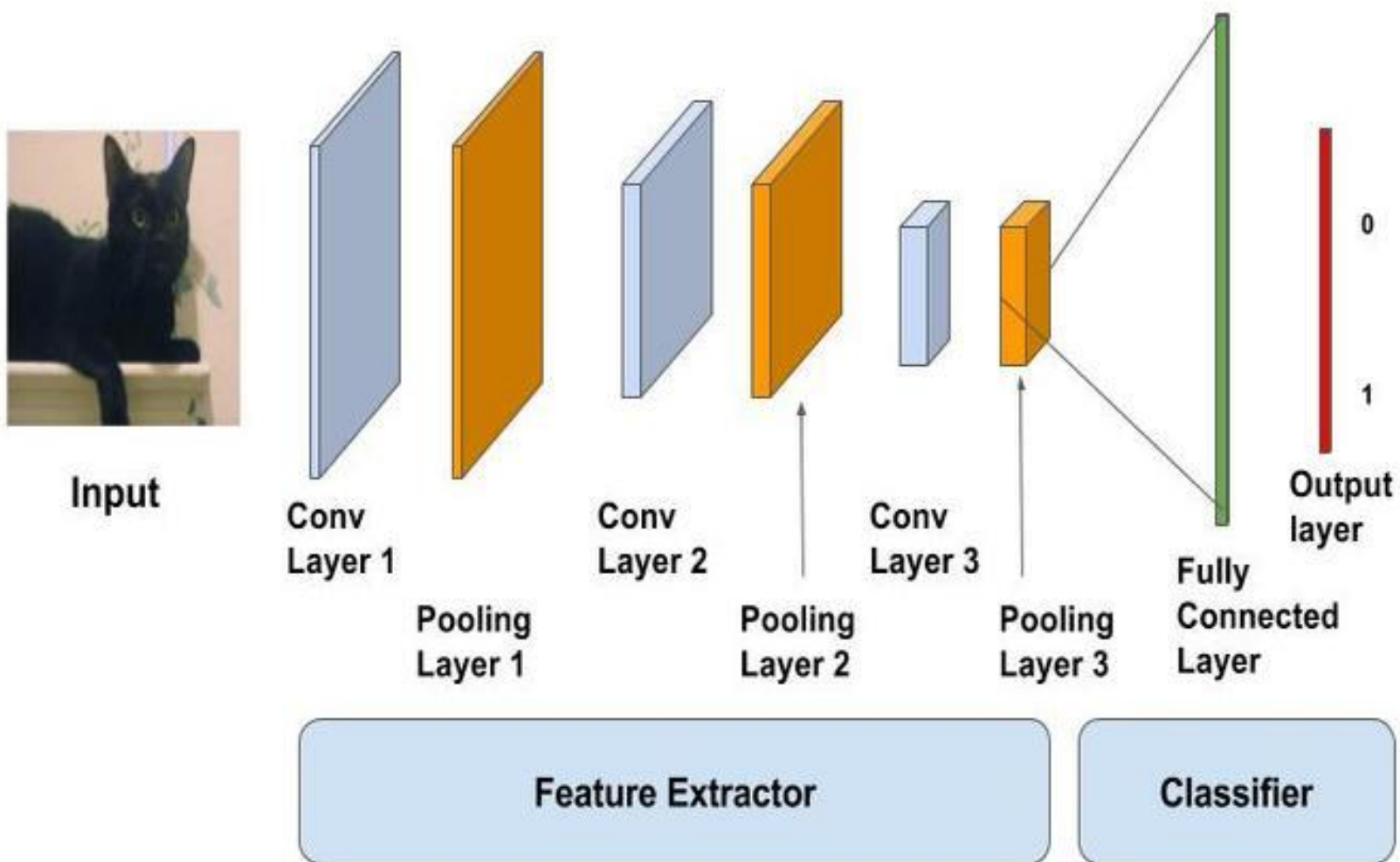
Convolutional Neural Networks

- Each layer in a CNN applies a different set of filters, typically hundreds or thousands of them, and combines the results, feeding the output into the next layer in the network.
- During training, a CNN automatically learns the values for these filters.
- In the context of image classification, our CNN may learn to:
 - Detect edges from raw pixel data in the first layer.
 - Use these edges to detect shapes (i.e., “blobs”) in the second layer.
 - Use these shapes to detect higher-level features such as facial structures, parts of a car, etc. in the highest layers of the network.
- The last layer in a CNN uses these higher-level features to make predictions regarding the contents of the image.

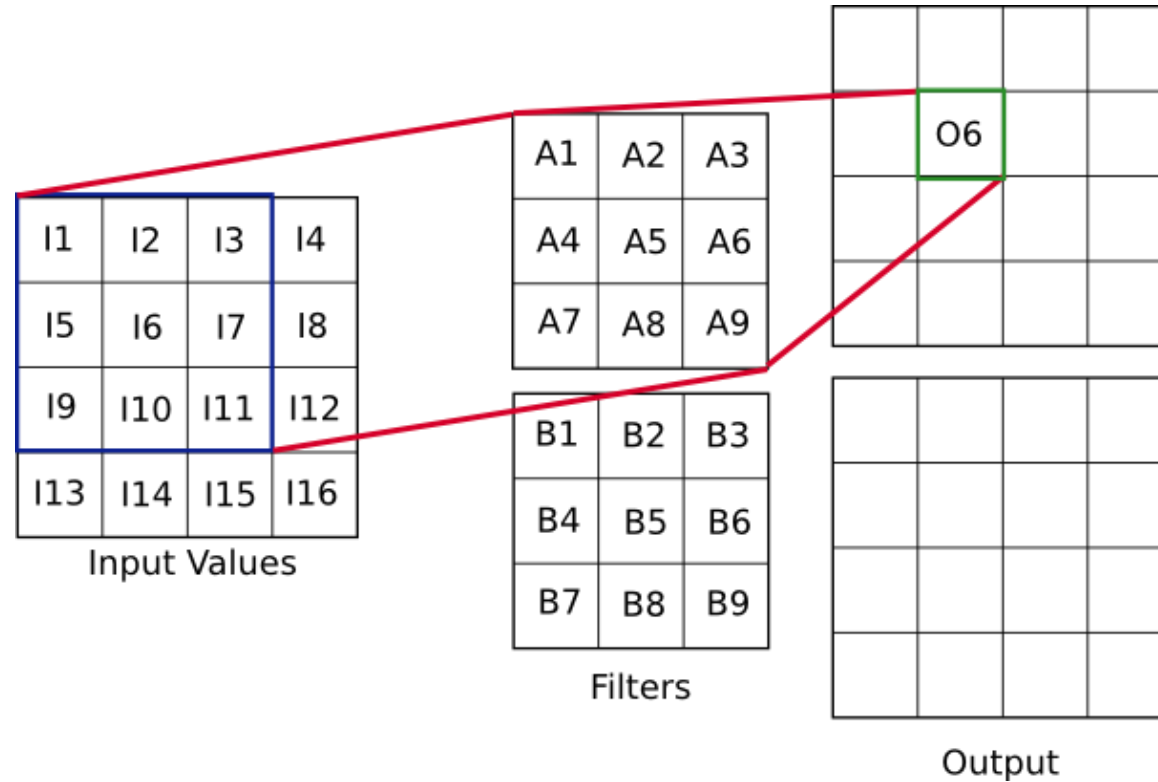
Convolutional Neural Networks

- CNN consist of three layers such as Convolution Layer, Pooling Layer, and Fully Connected Layer. ConvNet is architecture is formed stacking these layers.
- The output can be a softmax layer indicating whether there is a cat or something else. You can also have a sigmoid layer to give you a probability of the image being a cat.



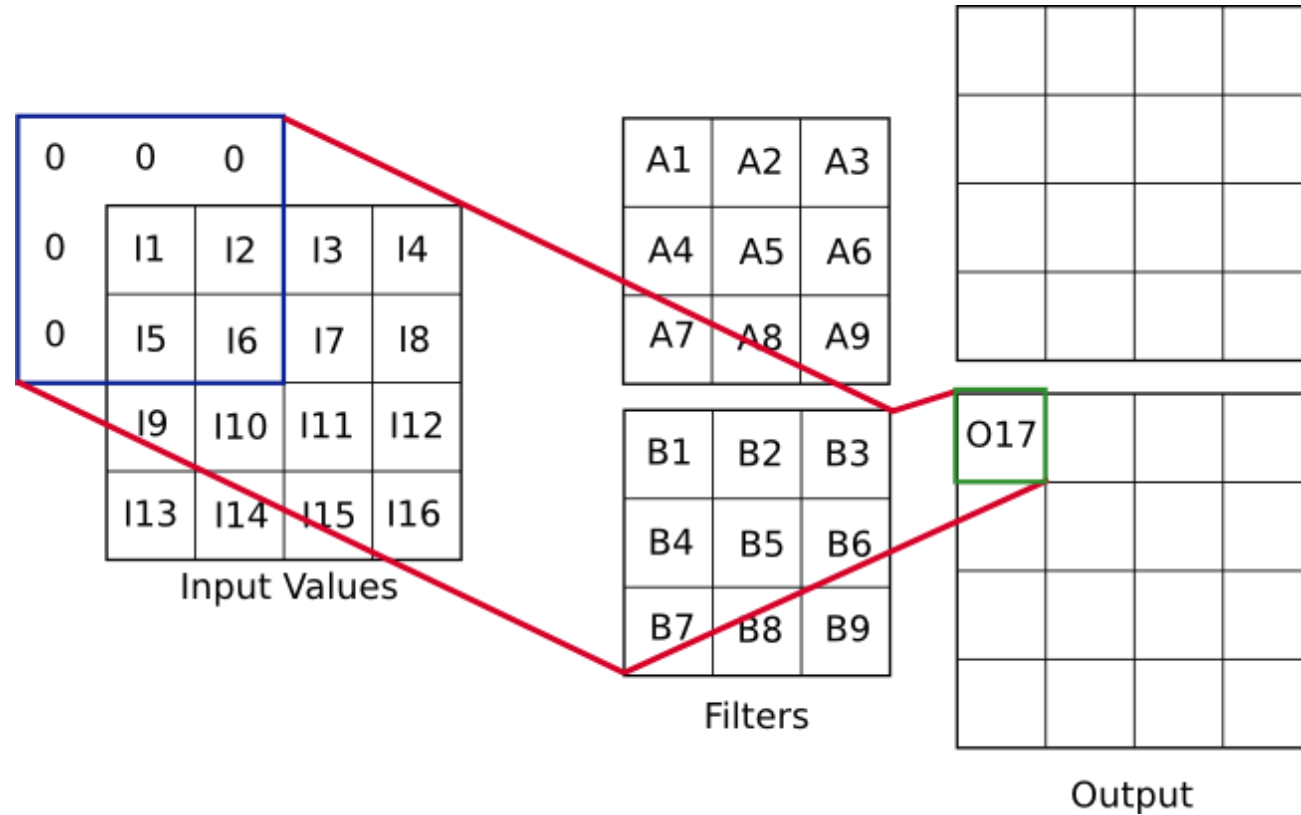


Convolutional Neural Networks



$$\begin{aligned} O_6 = & A_1 \cdot I_1 + A_2 \cdot I_2 + A_3 \cdot I_3 \\ & + A_4 \cdot I_5 + A_5 \cdot I_6 + A_6 \cdot I_7 \\ & + A_7 \cdot I_9 + A_8 \cdot I_{10} + A_9 \cdot I_{11} \end{aligned}$$

Convolutional Neural Networks



$$O_{17} = B_5 \cdot I_1 + B_6 \cdot I_2 + B_8 \cdot I_5 + B_9 \cdot I_6$$

Convolutional Layer

- The convolutional layer can be thought of as the eyes of the CNN. The neuron

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

6		

$7 \times 1 + 4 \times 1 + 3 \times 1 +$
 $2 \times 0 + 5 \times 0 + 3 \times 0 +$
 $3 \times -1 + 3 \times -1 + 2 \times -1$
 $= 6$

- Convolution can be thought of as a weighted sum between two signals.
- In image processing, to calculate convolution at a particular location (x,y), we extract k*k sized chunk from the image centered at location (x,y). We then multiply the values in this chunk element-by-element with the convolution filter and then add them all to obtain a single output.

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

 $*$

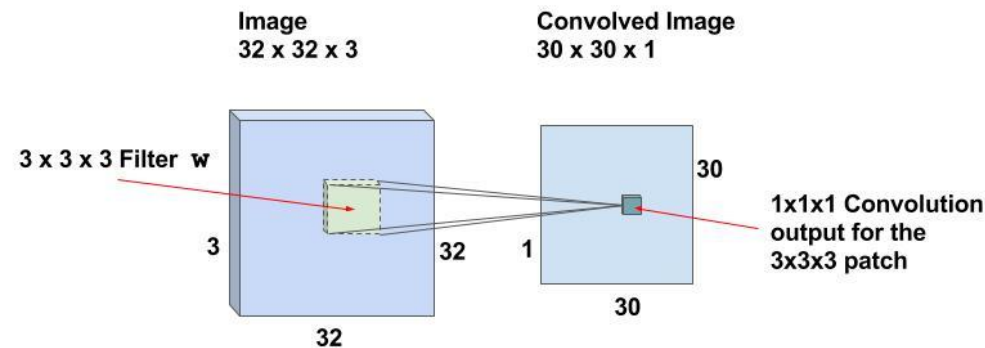
1	0	-1
1	0	-1
1	0	-1

 $=$

6	-9	-8
-3	-2	-3
-3	0	-2

The convolution kernel is slide over the entire matrix to obtain an activation map.

The input image is of size $32 \times 32 \times 3$. This is nothing but a 3D array of depth 3. Any convolution filter we define at this layer must have a depth equal to the depth of the input. So we can choose convolution filters of depth 3 (e.g. $3 \times 3 \times 3$ or $5 \times 5 \times 3$ or $7 \times 7 \times 3$ etc.). Let's pick a convolution filter of size $3 \times 3 \times 3$.



- Perform the convolution operation by sliding the $3 \times 3 \times 3$ filter over the entire $32 \times 32 \times 3$ sized image.
- We will obtain an output image of size $30 \times 30 \times 1$. This is because the convolution operation is not defined for a strip 2 pixels wide around the image. We have to ensure the filter is always inside the image. So 1 pixel is stripped away from left, right, top and bottom of the image.

Activation Maps

- $32 \times 32 \times 3$ input image and filter size of $3 \times 3 \times 3$, we have $30 \times 30 \times 1$ locations and there is a neuron corresponding to each location.
- $30 \times 30 \times 1$ outputs or activations of all neurons are called the activation maps. The activation map of one layer serves as the input to the next layer.

Shared weights and biases

- In our example, there are $30 \times 30 = 900$ neurons because there are that many locations where the $3 \times 3 \times 3$ filter can be applied.
- Traditional neural nets where weights and biases of neurons are independent of each other, in case of CNNs the neurons corresponding to one filter in a layer share the same weights and biases.

Stride

- We slide the window by 1 pixel at a time. We can also slide the window by more than 1 pixel. This number is called the stride.

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

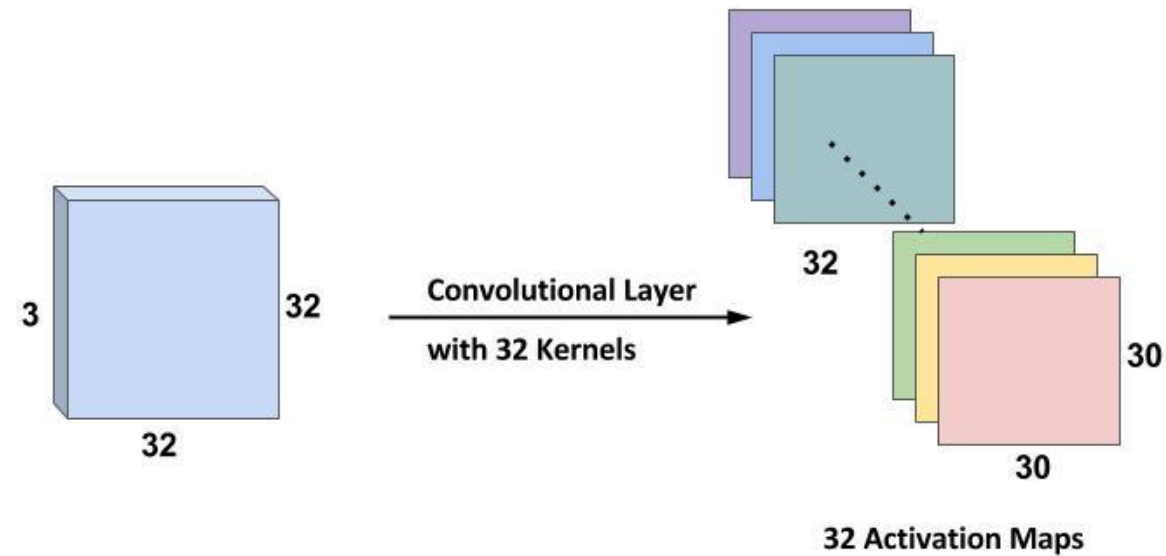
4		

Convolved
Feature

Multiple Filters

- We use more than 1 filter in one convolution layer. If we use 32 filters we will have an activation map of size 30x30x32.
- Note that all neurons associated with the same filter share the same weights and biases. So the number of weights while using 32 filters is simply $3 \times 3 \times 3 \times 32 = 288$ and the number of biases is 32.

- The 32 Activation maps obtained from applying the convolutional Kernels is shown below.

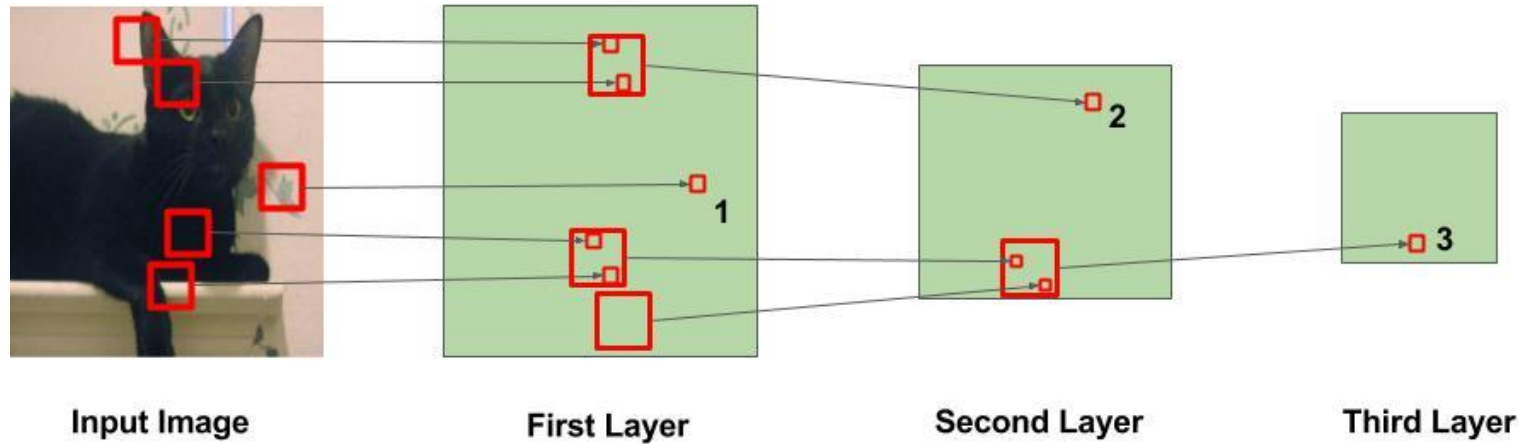


Zero padding

- After each convolution, the output reduces in size (as in this case we are going from 32×32 to 30×30).
- A standard practice to pad zeros to the boundary of the input layer such that the output is the same size as input layer.
- In this example, if we add a padding of size 1 on both sides of the input layer, the size of the output layer will be 32x32x32 which makes implementation simpler as well.
- Let's say you have an input of size $N \times N$, a filter of size F and you are using stride S and zero padding of size P is added to the input image. Then output will be of size $M \times M$.

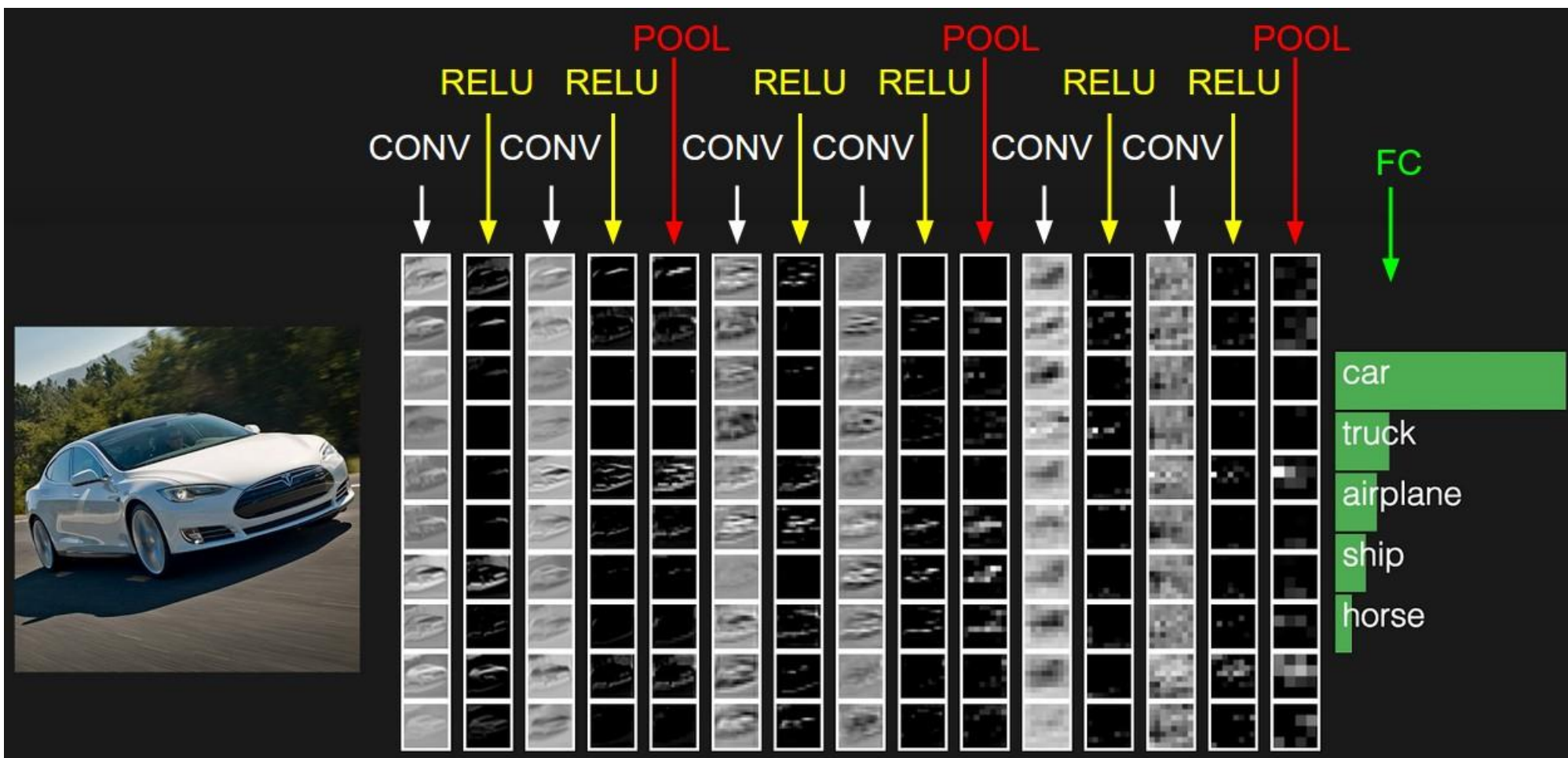
$$M = \frac{N - F + 2P}{S} + 1$$

CNNs learn Hierarchical features



In the above figure, the big squares indicate the region over which the convolution operation is performed and the small squares indicate the output of the operation which is just a number.

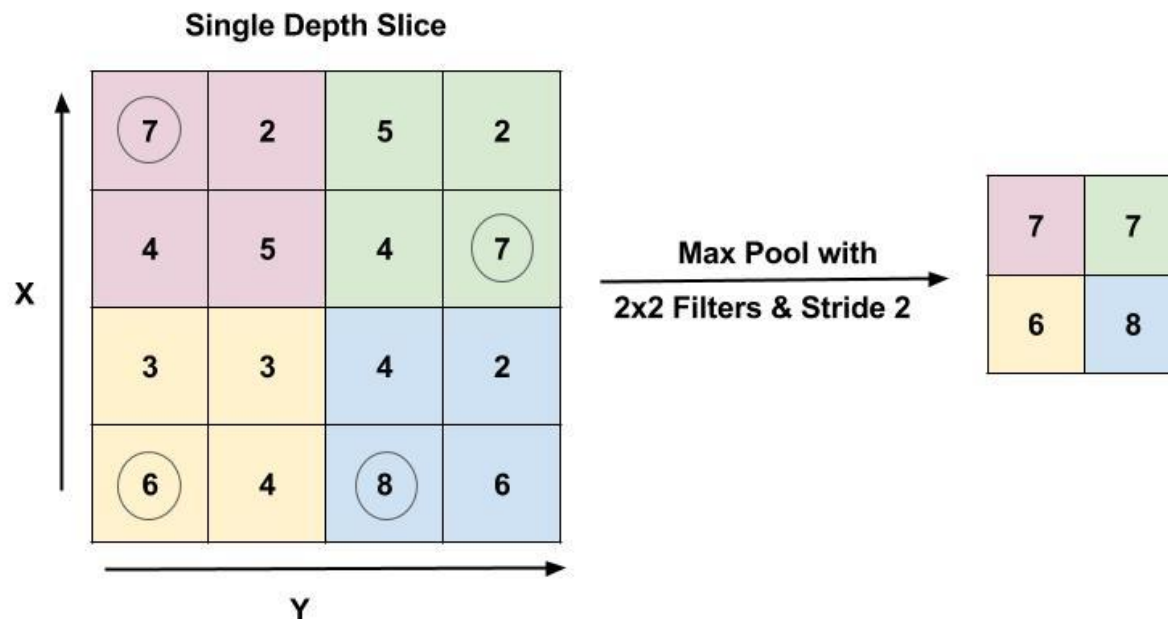
- In the first layer, the square marked 1 is obtained from the area in the image where the leaves are painted.
- In the second layer, the square marked 2 is obtained from the bigger square in Layer 1. The numbers in this square are obtained from multiple regions from the input image. Specifically, the whole area around the left ear of the cat is responsible for the value at the square marked 2.
- Similarly, in the third layer, this cascading effect results in the square marked 3 being obtained from a large region around the leg area.
- We can say from the above that the initial layers are looking at smaller regions of the image and thus can only learn simple features like edges / corners etc. As we go deeper into the network, the neurons get information from larger parts of the image and from various other neurons. Thus, the neurons at the later layers can learn more complicated features like eyes / legs.



Max Pooling Layer

- Pooling layer is mostly used immediately after the convolutional layer to reduce the spatial size (only width and height, not depth).
- The most common form of pooling is Max pooling where we take a filter of size P and apply the maximum operation over the sized part of the image.

- Max pool layer with filter size 2×2 and stride 2 is shown in below figure. The output is the max value in a 2×2 region shown using encircled digits.
- The most common pooling operation is done with the filter of size 2×2 with a stride of 2, which reduces the spatial dimensions by half.



Max Pool

2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5

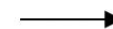


7	9
8	5

Max-Pool with a
2 by 2 filter and
stride 2.

Average Pool

2	3	1	9
4	7	3	5
8	2	2	2
1	3	4	5



4	4.5
3.25	3.25

Average Pool with
a 2 by 2 filter and
stride 2.

Image Input Data

- Each image has the same size of 32 pixels wide and 32 pixels high, and pixel values are between 0 and 255, e.g. a matrix of $32 \times 32 \times 1$ or 1,024 pixel values.

Convolutional Layer

- We define a convolutional layer with 10 filters and a receptive field 5 pixels wide and 5 pixels high and a stride length of 1. Because each filter can only get input from (i.e. see) 5×5 (25) pixels at a time, we can calculate that each will require $25 + 1$ input weights (plus 1 for the bias input).
- Dragging the 5×5 receptive field across the input image data with a stride width of 1 will result in a feature map of 28×28 output values or 784 distinct activations per image.
- We have 10 filters, so that is 10 different 28×28 feature maps or 7,840 outputs that will be created for one image.

Pool Layer

- We define a pooling layer with a receptive field with a width of 2 inputs and a height of 2 inputs. We also use a stride of 2 to ensure that there is no overlap.
- This results in feature maps that are one half the size of the input feature maps. From 10 different 28×28 feature maps as input to 10 different 14×14 feature maps as output.

Fully Connected Layer

- Finally, we can flatten out the square feature maps into a traditional fully connected layer.
- We can define the fully connected layer with 200 hidden neurons, each with $10 \times 14 \times 14$ input connections, or $1,960 + 1$ weights per neuron.
- That is a total of 392,000 connections and weights to learn in this layer.

Convolutional Neural Network for MNIST

