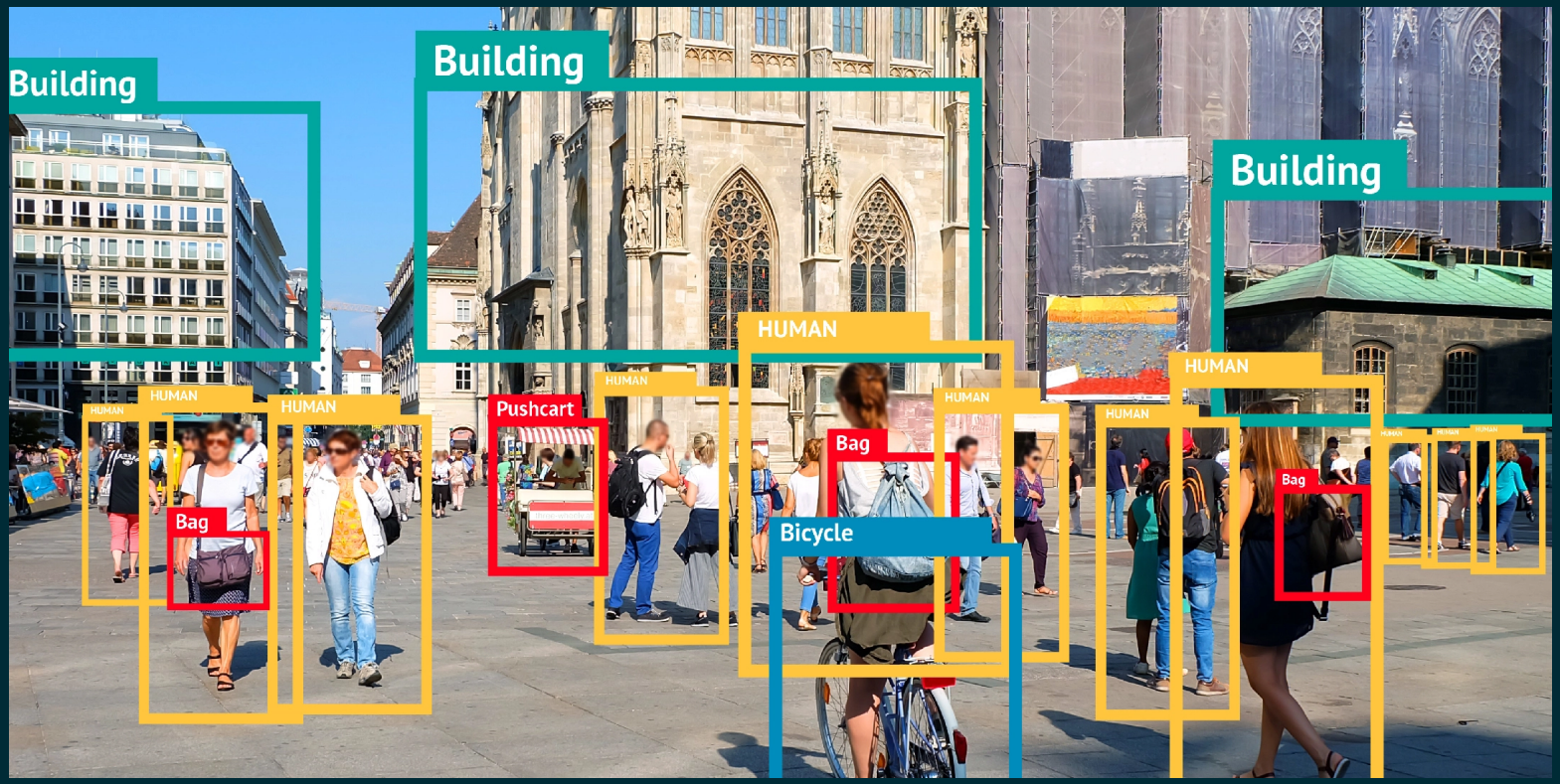
INTRODUCTION TO OBJECT DETECTION



OT ENGSOC ARTIFICIAL INTELLIGENCE WORKSHOP 01

AUGUST 23, 2020

BY: CHIRAG KARIA

IMAGE SOURCE

WHO AM I?

- UOIT SOFE Graduate (April 2019)
- Pursuing an MSc in CS
- Researching 3D perception with computer vision
- Worked at two ML based finance startups

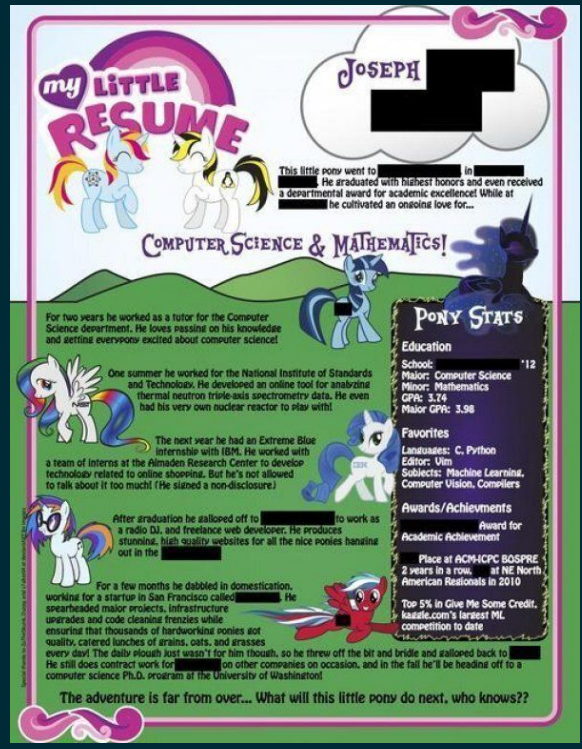
WHY CARE ABOUT OBJECT DETECTION?

WHY CARE ABOUT OBJECT DETECTION?

- Simplest way to facilitate scene understanding
- Good introduction to more complex DL models
- First step in many vision pipelines
- Any software company will hire you if you beat SOTA

Joseph Redmon

- PhD student at University of Washington
- Created the YOLO algorithm
- Has worked with research groups at IBM, Google, and Facebook



This is his resume.

OVERVIEW

1. Summary of Neural Networks
    - 1.1 - What is a Neuron?
    - 1.2 - Feed Forward Neural Networks
    - 1.3 - Convolutional Neural Networks
2. Object Detection
    - 2.1 - Types of Object Detectors
    - 2.2 - Intersection over Union
    - 2.3 - Anchor Boxes
    - 2.4 - Non-Maximum Suppression
    - 2.5 - Faster R-CNN Pipeline
3. Demo

SUMMARY OF NEURAL NETWORKS

WHAT IS A NEURON?

WHAT IS A NEURON?

ML algorithm:

- Maps a vector of inputs $\vec{x}$ to a single output $\hat{y}$
- Classification tasks (discrete output)
- Regression tasks (continuous output)
- Inspired by biological neurons

WHAT IS A NEURON?

BIOLOGICAL VS. ARTIFICIAL NEURON (PERCEPTRON)

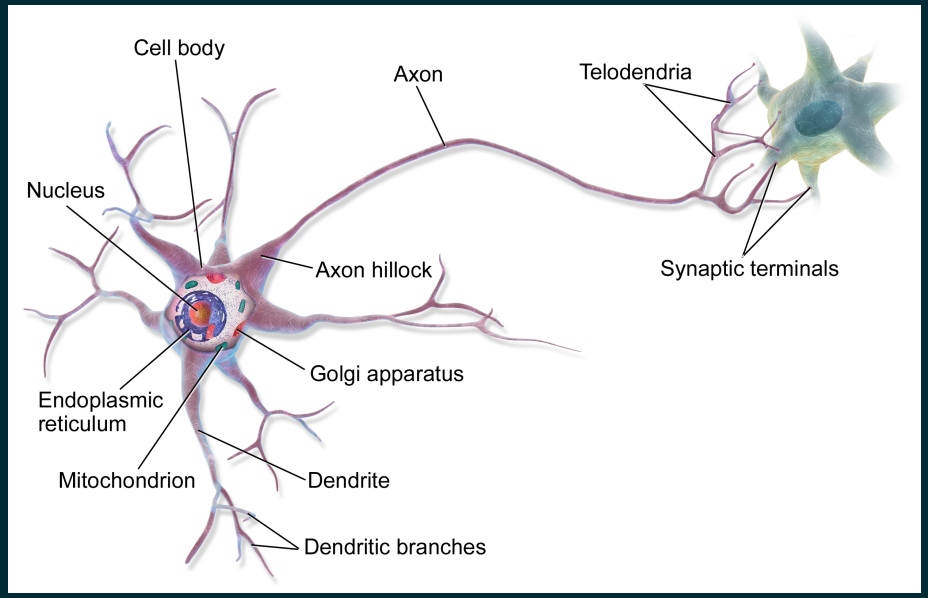


Image Source

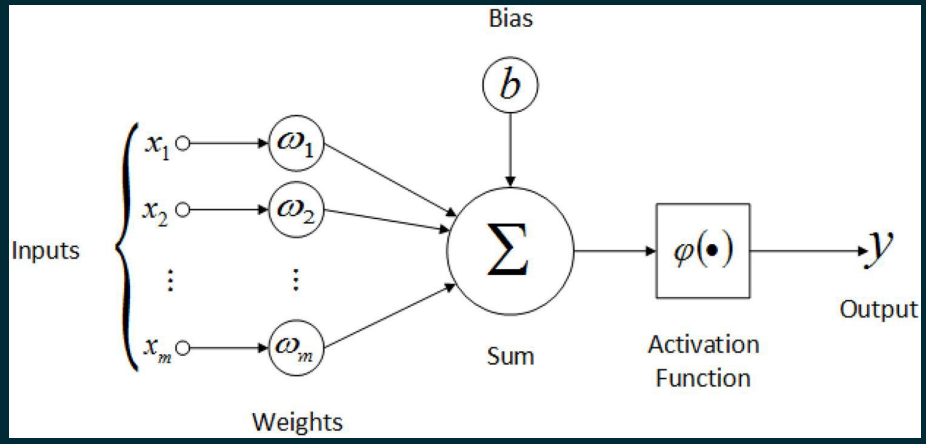


Image Source

WHAT IS A NEURON?

THE PERCEPTRON

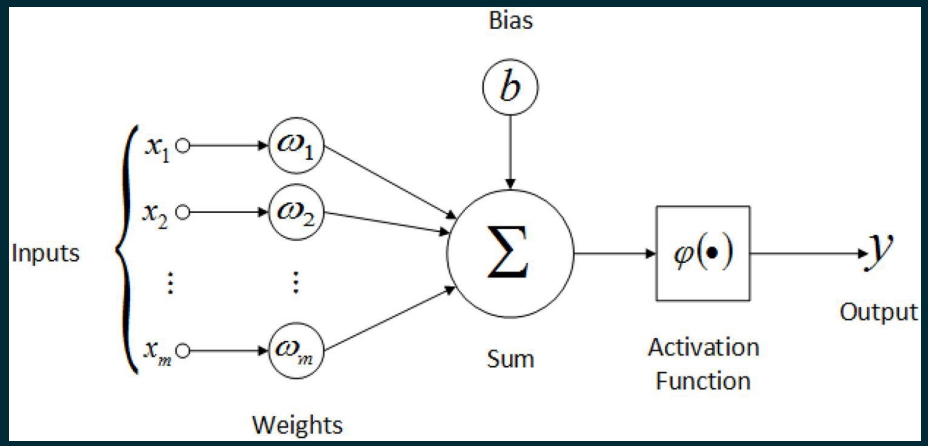


Image Source

$$\hat{y} = \varphi\left(\left(\sum_{n=1}^{m} x_n w_n\right) + b\right)$$

if we create row vectors of inputs and weights:

$$\vec{x} = [x_1, x_2, \ldots, x_m, 1]$$
$$\vec{w} = [w_1, w_2, \ldots, w_m, b]$$

then:

$$\hat{y} = \varphi(\vec{x} \cdot \vec{w})$$

A neuron is simply a dot product of inputs $\vec{x}$ and weights $\vec{w}$ passed through an activation function φ

WHAT IS A NEURON?

ACTIVATION FUNCTIONS

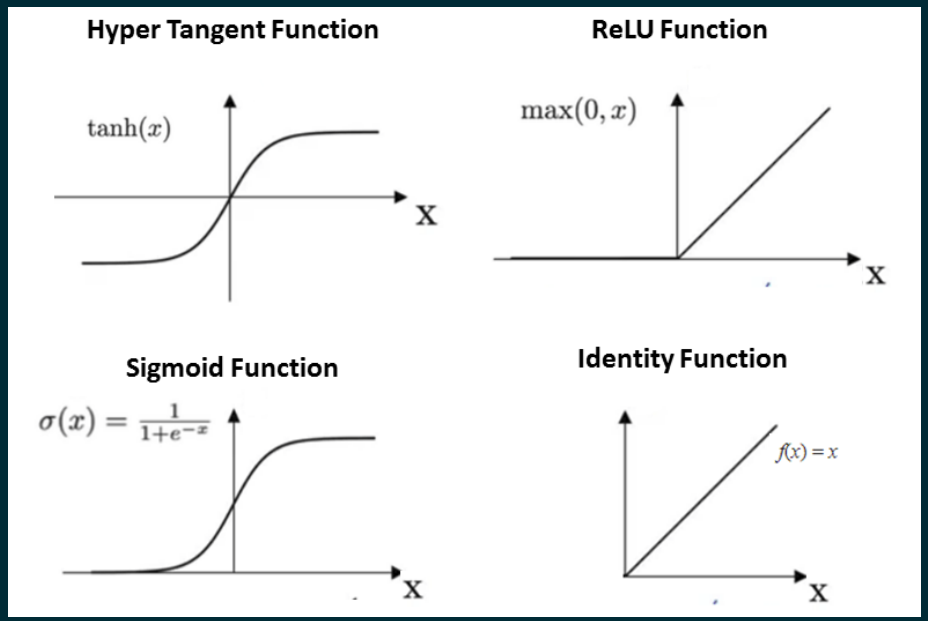


Image Source

- Without an activation function, a neuron is simply performing linear regression.
- Activation functions are also referred to as non-linearities; they serve to help neurons model non-linear domains

WHAT IS A NEURON?

ACTIVATION FUNCTIONS

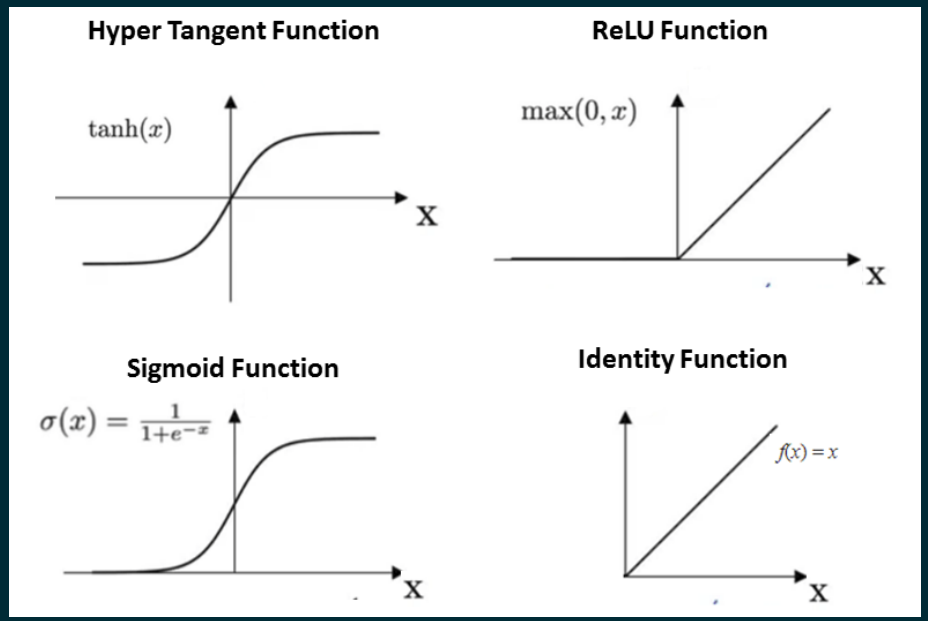


Image Source

- Without an activation function, a neuron is simply performing linear regression.
- Activation functions are also referred to as non-linearities; they serve to help neurons model non-linear domains

Bounds of activation functions

Hyper Tangent Function: $-1 < x < +1$

ReLu Function: $0 \leq x < +\infty$

Sigmoid Function: $0 < x < +1$

*Identity Function: $-\infty < x < +\infty$

*Using an Identity Function is equivalent to not having an activation function at all

WHAT IS A NEURON?

HOW DO WE TRAIN A NEURON?

- Take a neuron with m inputs in $\vec{x}$ and a single output $\hat{y}$
- Across our dataset we want to minimize the difference between the predicted value $\hat{y}$ and the actual value y.
- Imagine a function $J(\vec{w})$ that takes the current set of weights $\vec{w}$ and calculates the error of our model across a subset of our dataset. The lower this value, the better our weights are.
- Follow the gradient to the bottom.

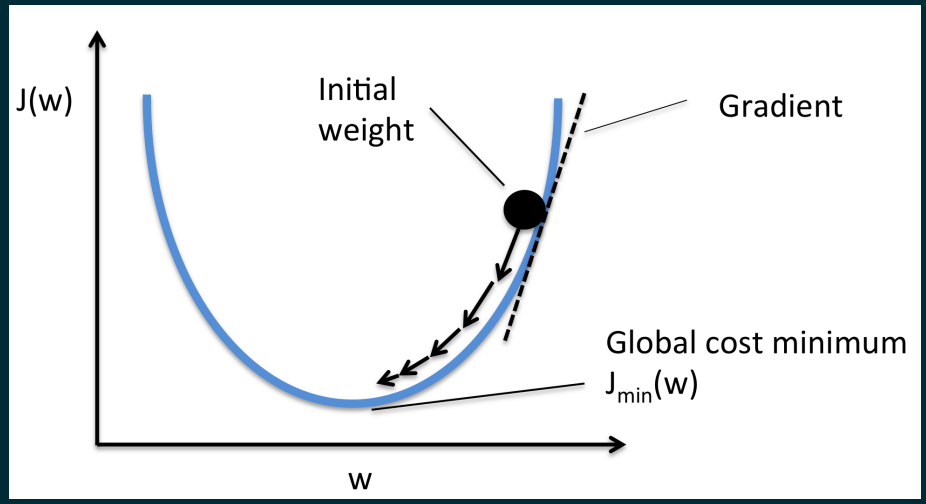


Image Source

WHAT IS A NEURON?

HOW DO WE TRAIN A NEURON?

let our loss function (J) be Mean Squared Error:

$$J(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

where N is the size of the subset of data. The partial derivatives of $J(\vec{w})$ with respect to w_i is:

$$\frac{\partial J(\vec{w})}{\partial w_i} = \frac{2}{N}\sum_{n=1}^{N} x_{(n,i)}\,(\hat{y}_n - y_n)$$

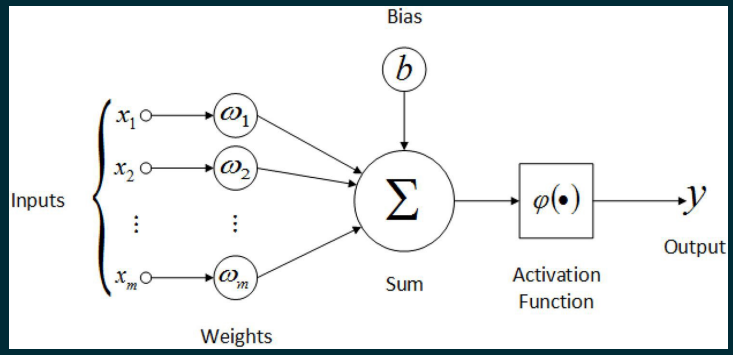


Image Source

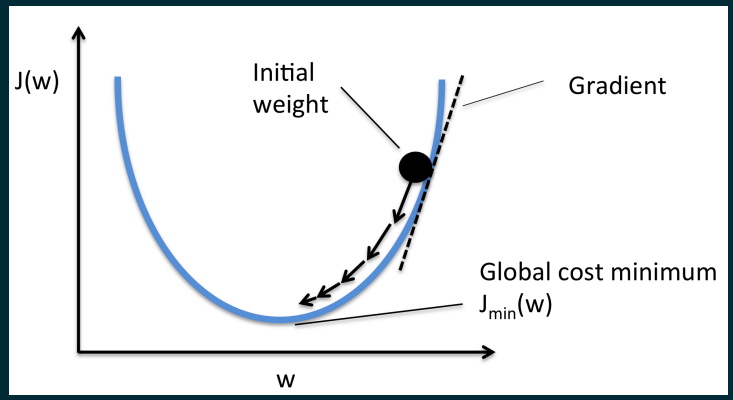


Image Source

WHAT IS A NEURON?

HOW DO WE TRAIN A NEURON?

where N is the size of the subset of data. The partial derivatives of $J(\vec{w})$ with respect to w_i is:

$$\frac{\partial J(\vec{w})}{\partial w_i} = \frac{2}{N}\sum_{n=1}^{N} x_{(n,i)}(\hat{y}_n - y_n)$$

the gradient of $J(\vec{w})$ with respect to $\vec{w}$ is:

$$\nabla_{\vec{w}} J(\vec{w}) = \begin{bmatrix} \frac{\partial J(\vec{w})}{\partial w_1} \\ \dots \\ \frac{\partial J(\vec{w})}{\partial w_m} \end{bmatrix} = \frac{2}{N}\boldsymbol{X}^T \cdot (\boldsymbol{X} \cdot \vec{w}^T - \boldsymbol{Y})$$

where:

$\boldsymbol{X} \in \mathbb{R}^{(N \times m)}$ $\quad$ $\vec{w}^T \in \mathbb{R}^{m \times 1}$

$\boldsymbol{Y} \in \mathbb{R}^{N \times 1}$ $\quad$ $\nabla_{\vec{w}} J(\vec{w}) \in \mathbb{R}^{m \times 1}$

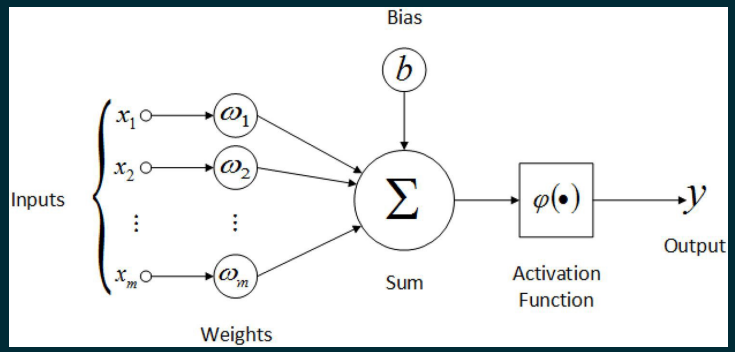


Image Source

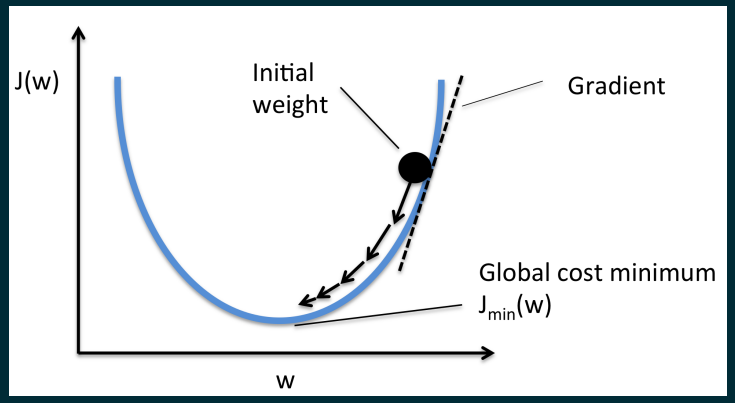


Image Source

WHAT IS A NEURON?

HOW DO WE TRAIN A NEURON?

the gradient of $J(\vec{w})$ with respect to $\vec{w}$ is:

$$\nabla_{\vec{w}} J(\vec{w}) = \begin{bmatrix} \frac{\partial J(\vec{w})}{\partial w_1} \\ \dots \\ \frac{\partial J(\vec{w})}{\partial w_m} \end{bmatrix} = \frac{2}{N} \boldsymbol{X}^T \cdot (\boldsymbol{X} \cdot \vec{w}^T - \boldsymbol{Y})$$

where:

$$\boldsymbol{X} \in \mathbb{R}^{(N \times m)} \qquad \vec{w}^T \in \mathbb{R}^{m \times 1}$$

$$\boldsymbol{Y} \in \mathbb{R}^{N \times 1} \qquad \nabla_{\vec{w}} J(\vec{w}) \in \mathbb{R}^{m \times 1}$$

We simply update our weights for the next iteration by:

$$\vec{w}_{next} = \vec{w}^T - \eta \nabla_{\vec{w}} J(\vec{w})$$

where η is a small constant.

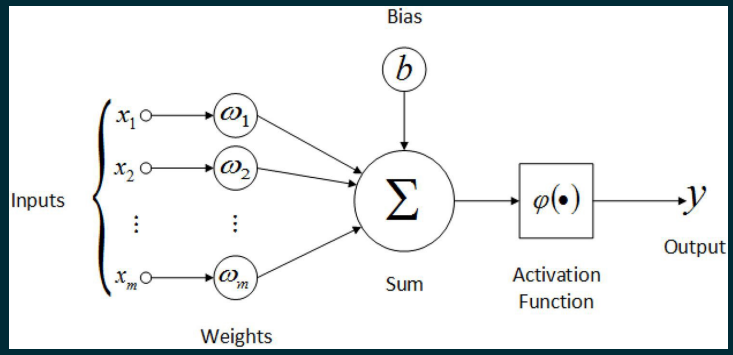


Image Source

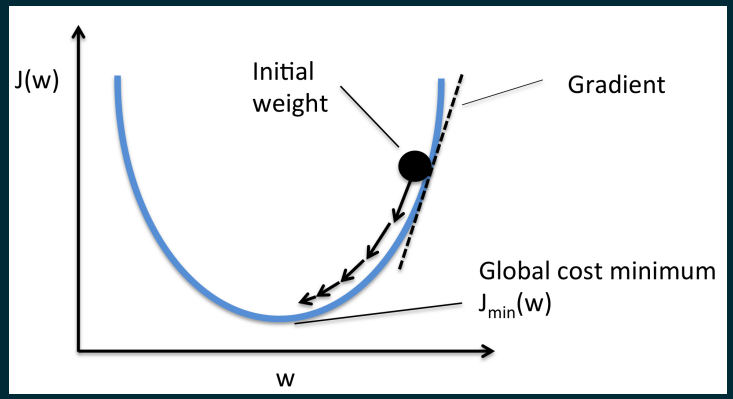


Image Source

FEED FORWARD NEURAL NETWORKS

FEED FORWARD NEURAL NETWORKS

A SINGLE NEURON IS NOT ENOUGH.

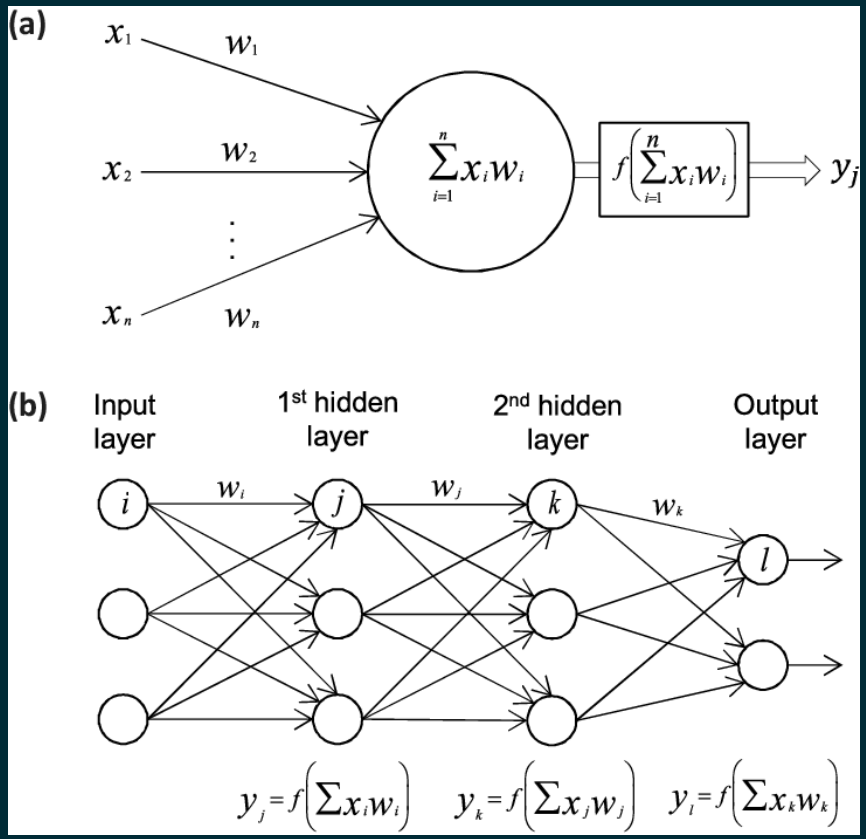


Image Source

- It can only provide a single output value.
- It solely looks at the different weighted combinations of inputs. Does not create new representations .
- Feed forward neural networks, or fully connected networks consist of multiple layers of multiple neurons.
- Each layer acts as the input for the subsequent layer. Each layer can be considered a representation of the prior layer.
- Gradients are calculated after the output of the last layer. They are then propagated back through the network (Back-Propagation) to update parameters/weights.

FEED FORWARD NEURAL NETWORKS

A SINGLE NEURON IS NOT ENOUGH.

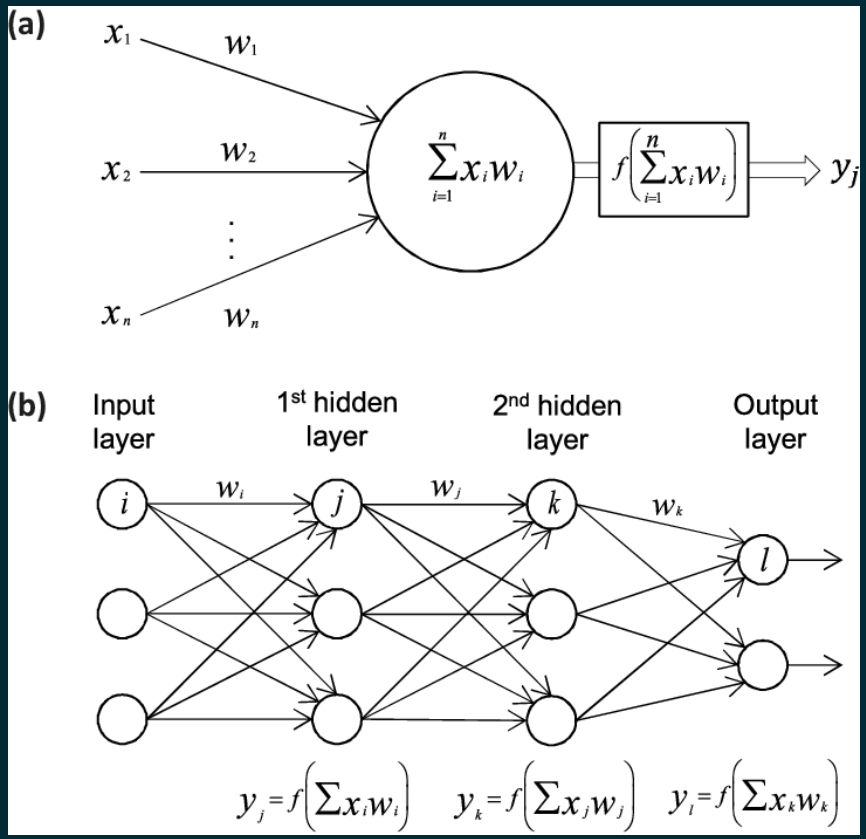


Image Source

- Feed forward networks let us have multiple output values.
- This means we can regress more than 1 output, or classify more than 1 class.
- Hidden layers also give rise to new internal representations for subsequent layers.

CONVOLUTIONAL NEURAL NETWORKS

CONVOLUTIONAL NEURAL NETWORKS

WORKING WITH HIGH DIMENSIONAL DATA

- Let us assume we want to use a neural network with a color image of size 300 pixels x 300 pixels.
- What a computer sees when presented with this image is $\vec{x} \in \mathbb{R}^{H \times W \times 3}$.
- Flattening the image (making it a vector) results in $300 \times 300 \times 3 = 270,000$ inputs.
- If we try and train a feed forward neural network, and our first hidden layer l consists of 100×100 neurons we have:

$$100 \times 100 \times (2.7 \times 10^5) = 2.7 \times 10^9$$

- We need 2.7 BILLION parameters for just our first layer!
- This is not computationally feasible.

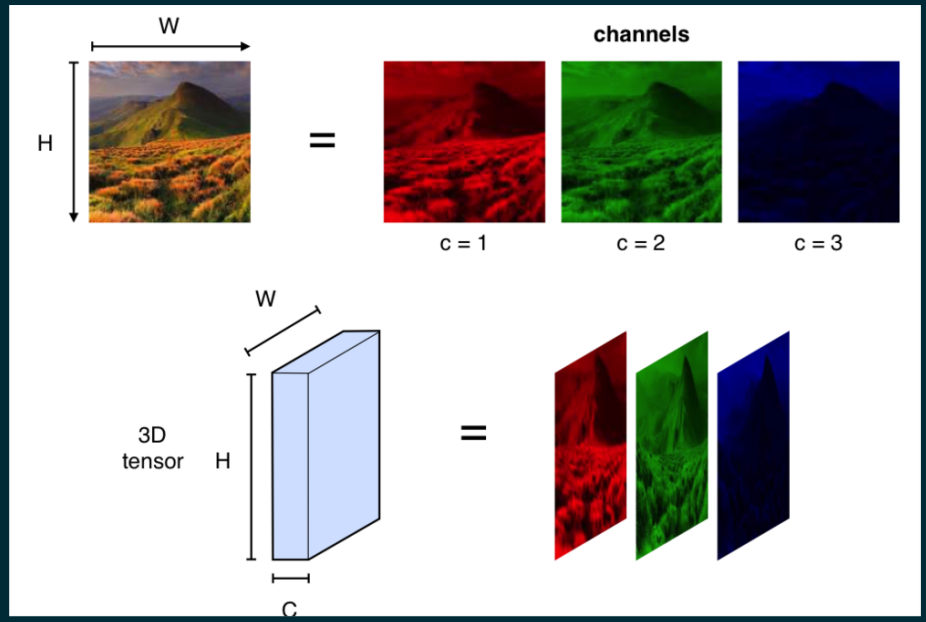


Image Source

CONVOLUTIONAL NEURAL NETWORKS

WORKING WITH HIGH DIMENSIONAL DATA

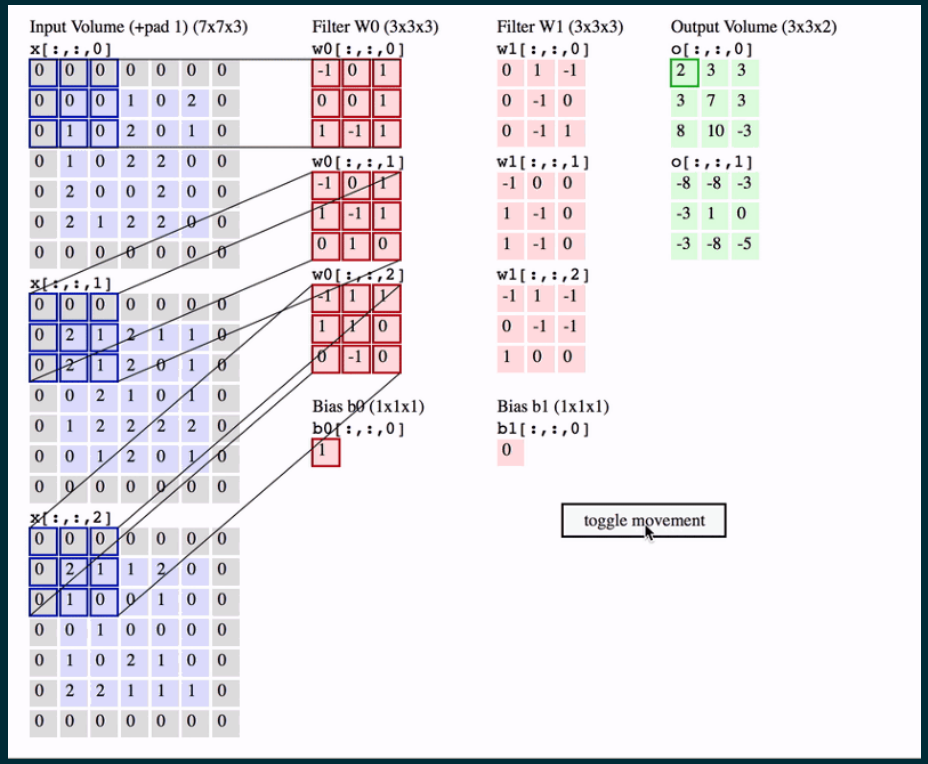


Image Source

- 2D Convolutional networks take a fixed set of neurons and *convolve* them over the input.
- In this example we have 2 neurons (referred to as filters here) which take an input of $k \times j \times d_{l-1}$ where k and j is our kernel height and width respectively, and d_{l-1} is the number of channels (depth) in the previous layer.
- Thus, both neurons have an input size of $k \times j \times d_{l-1}$. In this example that means each neuron has $3 \times 3 \times 3 = 27$ inputs and parameters

CONVOLUTIONAL NEURAL NETWORKS

WORKING WITH HIGH DIMENSIONAL DATA

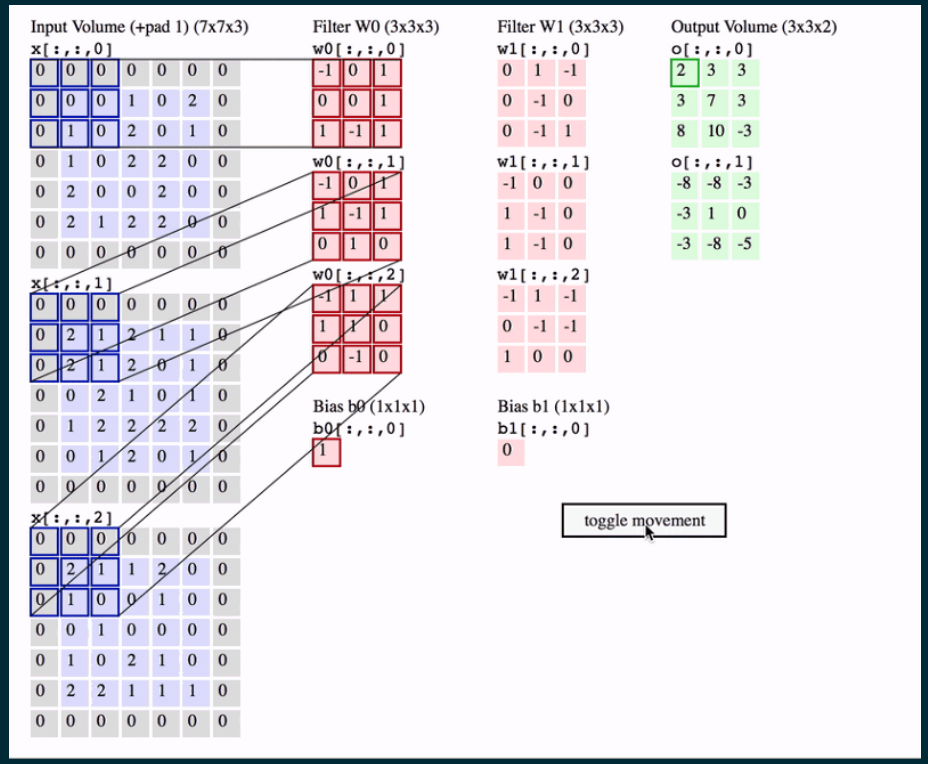


Image Source

- The stride (s) of a convolutional neuron defines how many pixels the kernel slides to the right, and after there are no more columns, how many pixels does it go down and start again. Here $s = 2$
- A convolutional layer outputs a *feature map*. This feature map captures hierarchical information about the previous layer. These feature maps form new representations.
- The height and width of a conv layer's feature map is a function of the neurons' stride and kernel size. The depth is equivalent to how many neurons are used.

CONVOLUTIONAL NEURAL NETWORKS

WORKING WITH HIGH DIMENSIONAL DATA

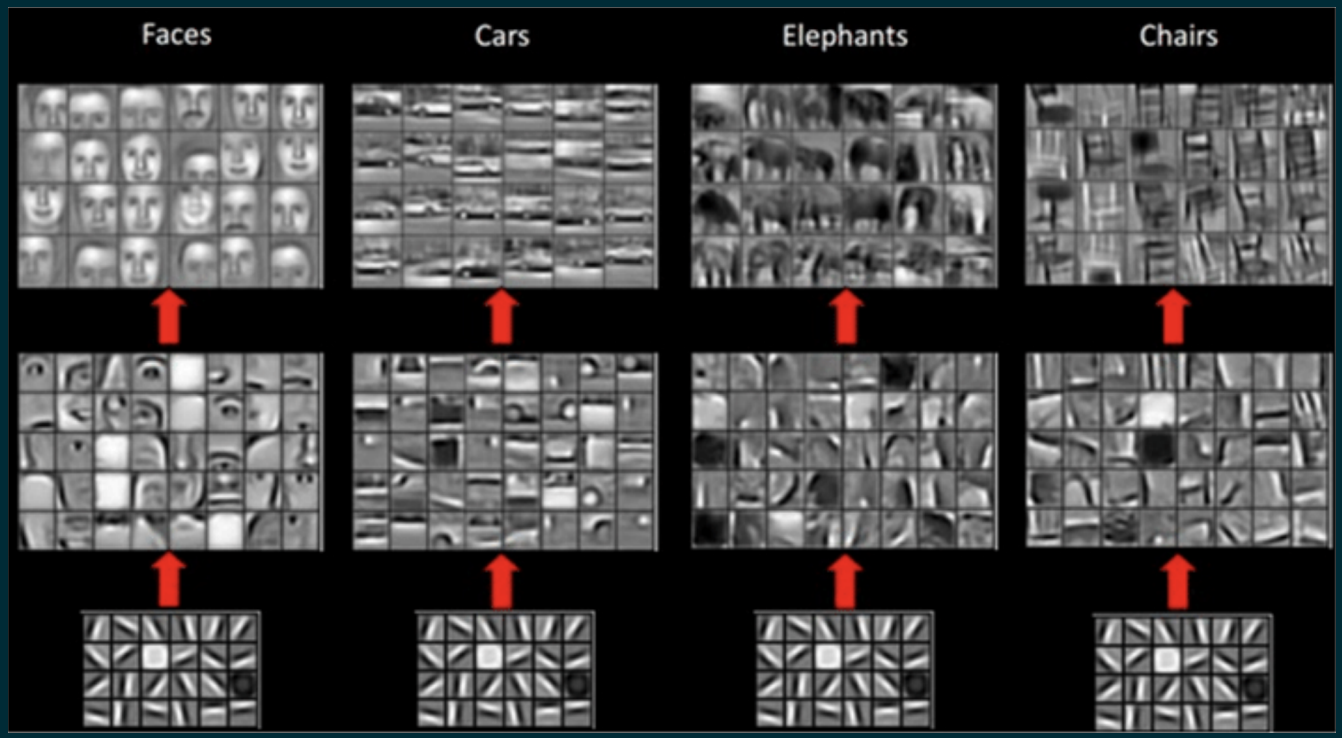


Image Source

- A convolutional layer outputs a *feature map*. This feature map captures hierarchical information about the previous layer. These feature maps form new representations.

CONVOLUTIONAL NEURAL NETWORKS

WORKING WITH HIGH DIMENSIONAL DATA

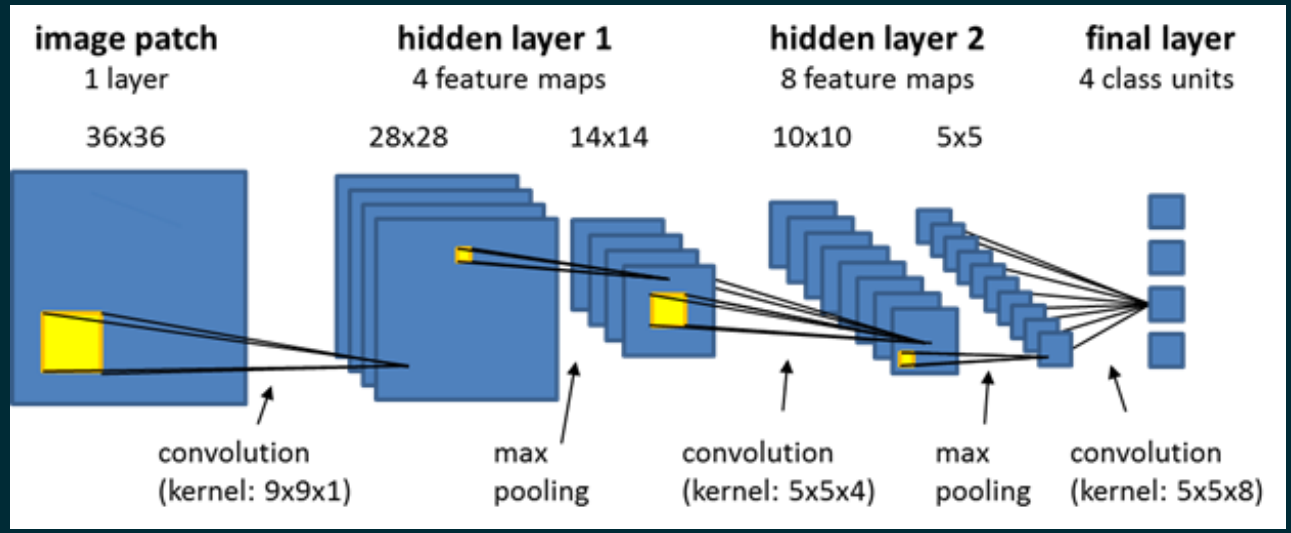


Image Source

- The height and width of a conv layer's feature map is a function of the neurons' stride and kernel size. The depth is equivalent to how many neurons are used.

OBJECT DETECTION

TYPES OF OBJECT DETECTORS

Single Shot	Two Shot
• Takes an input image, and directly predicts bounding boxes and object categories • Faster than two shot approaches, but less accurate • Ideal for realtime use cases.	• Splits prediction into two steps: 1. Generate candidate object proposals (objectness score + bounding box) 2. Categorize candidate proposals and refine bounding box • Slower than single shot approaches, but more accurate • Ideal for less time sensitive applications

INTERSECTION OVER UNION

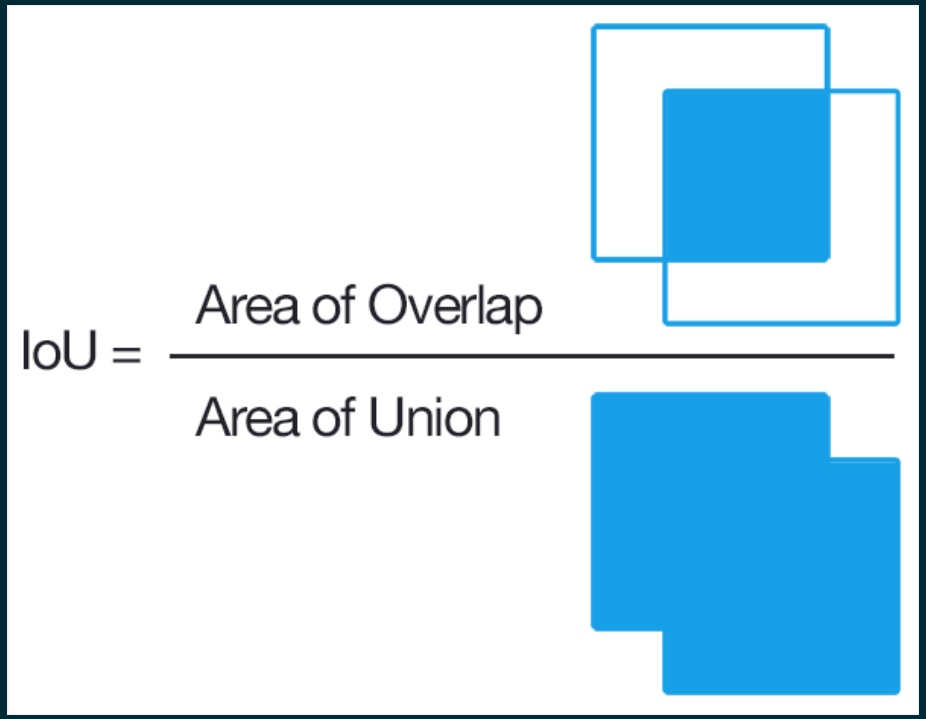


Image Source

- Used to quantify how spatially similar two bounding boxes are.
- Does not take object class into account. Only used to quantify similarity of the size and location of boxes.

$$IoU(B_1, B_2) = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$

where

$$0 \leq IoU \leq 1$$

Image Source

ANCHOR BOXES

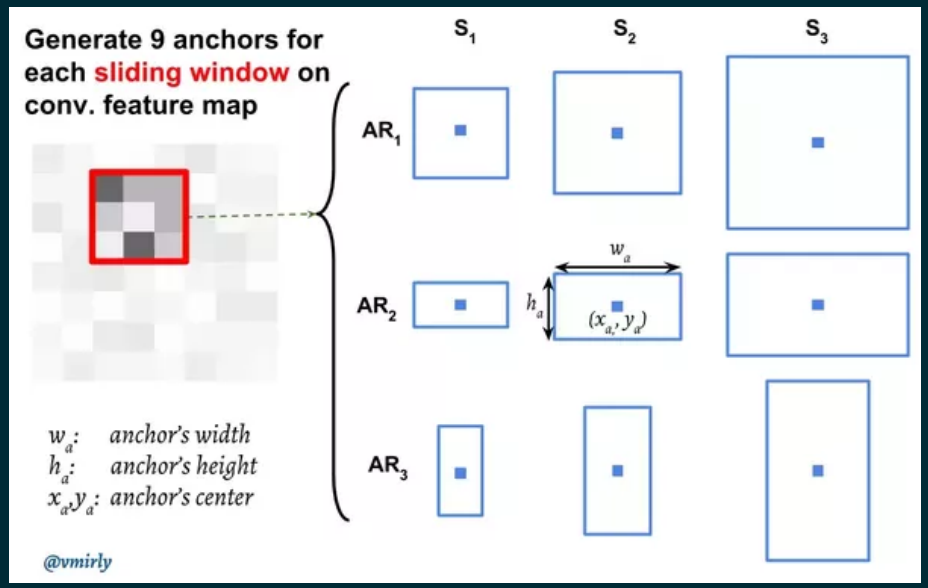


Image Source

- It is extremely difficult to train an object detector to output predictions relative to the image as a whole.
- Instead we predict relative to anchor boxes. These are known as priors.
- Anchors are tiled across the image, and each ground truth object is paired with the anchor that it has the highest IoU with.
- Additionally, any unused anchor that has an $IoU > 0.7$ with an object (this varies across object detectors) is also matched with the corresponding ground truth.
- This means we can have more than 1 anchor assigned to a single object in the image

ANCHOR BOXES

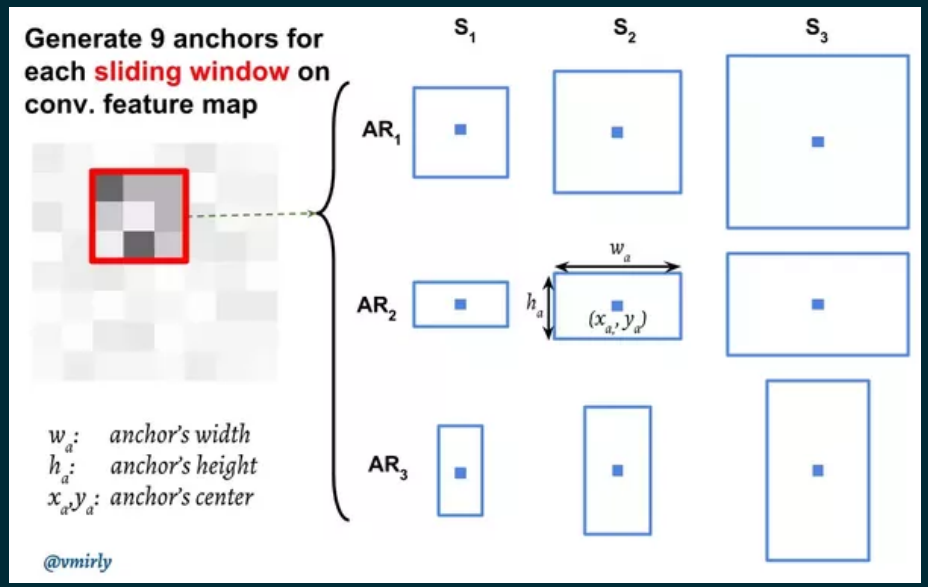


Image Source

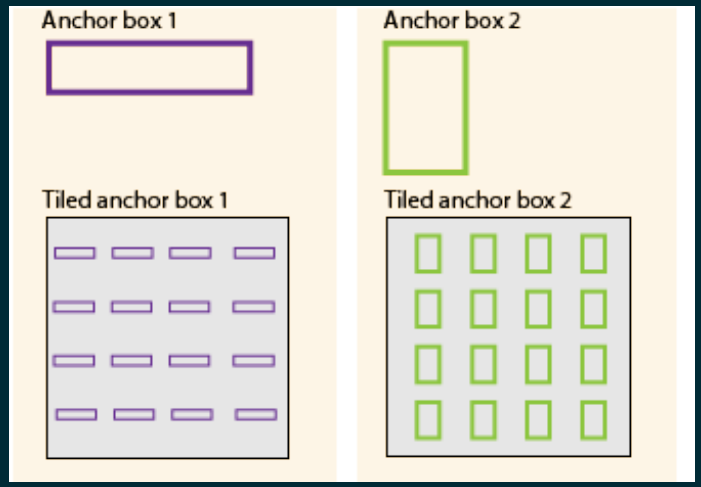


Image Source

Our regression targets are:

$$x_{\hat{y}} = \frac{x_{gt} - x_a}{w_a}$$

$$y_{\hat{y}} = \frac{y_{gt} - y_a}{h_a}$$

$$w_{\hat{y}} = \log(\frac{w_{gt}}{w_a})$$

$$h_{\hat{y}} = \log(\frac{h_{gt}}{h_a})$$

NON-MAXIMUM SUPPRESSION

NON-MAXIMUM SUPPRESSION

- Because we matched multiple anchors to a ground truth box during training, when we evaluate predictions we will have multiple predictions for the same object.
- Non-Maximum Suppression serves to filter out extraneous predictions and give us the final set.
- NMS is performed by:
  1. Sorting all predictions by class and confidence.
  2. Any prediction that is the same class as another prediction, with a lower confidence and an $IoU > 0.45$ is dropped

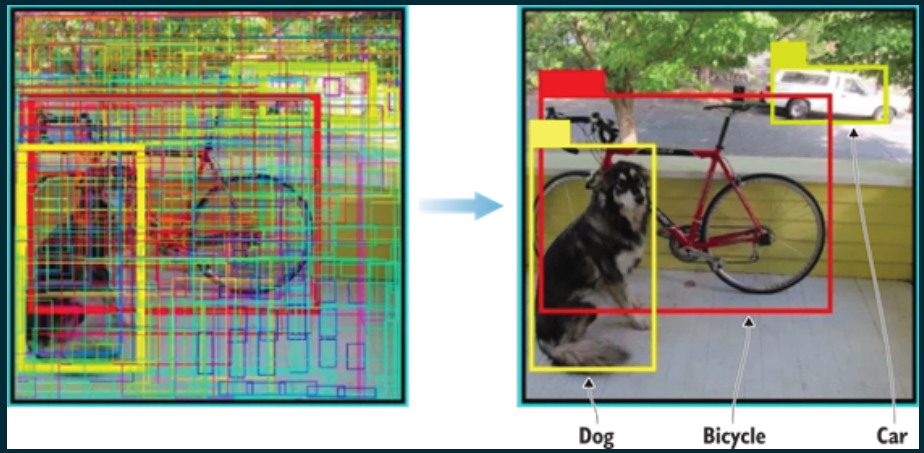


Image Source

FASTER R-CNN

FASTER R-CNN

- Faster R-CNN is a two stage object detection algorithm.
- It uses the Region Proposal Network to generate candidate predictions.
- It then uses the Fast R-CNN to classify the object and refine the bounding box.

FASTER R-CNN

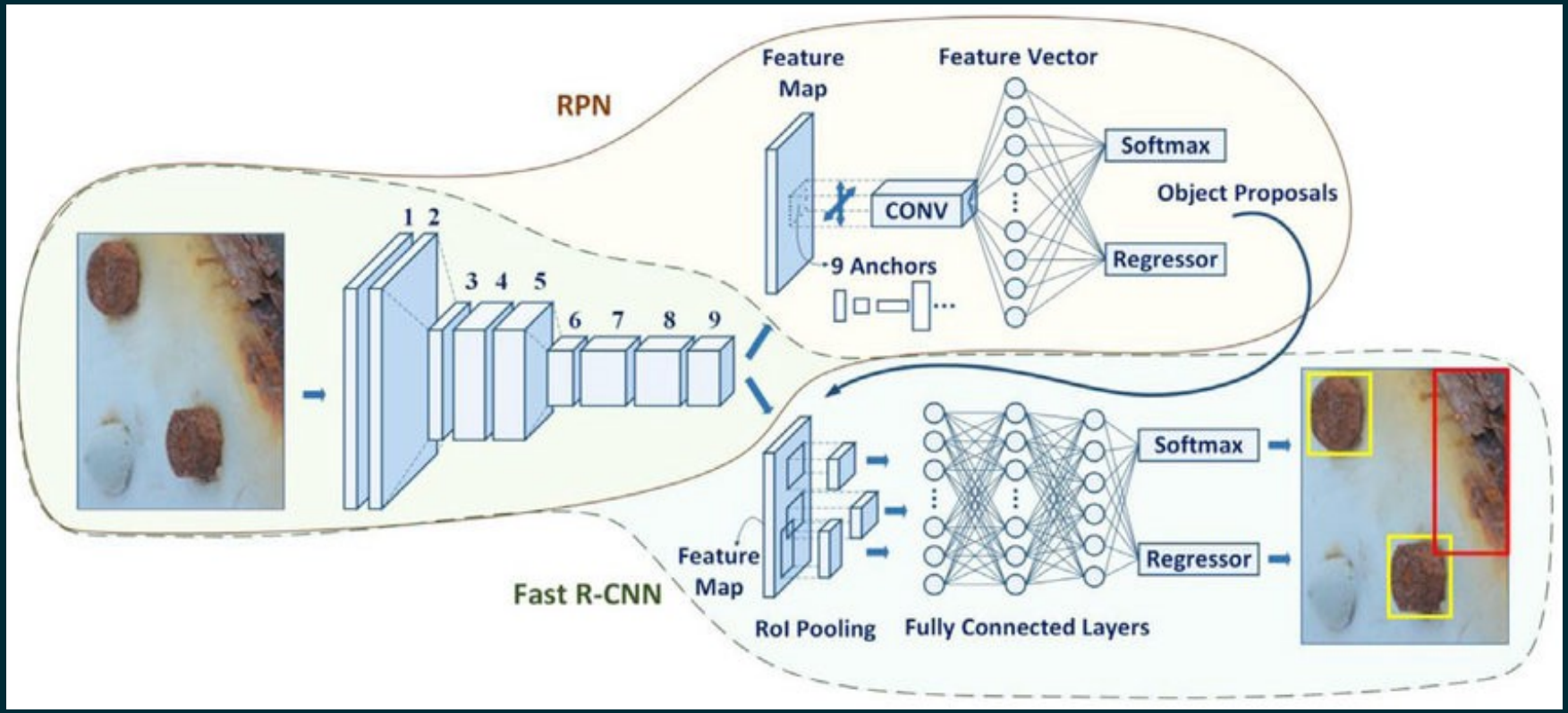


Image Source

DEMO

DEMO

1. Open the following Colab notebook
2. Hit `File > Save a copy in Drive` to save the notebook to your own google drive.
3. Follow along with me!

QUESTIONS?