THE CONVOLUTIONAL NEURAL NETWORK

VISUAL RECOGNITION

ARIHARASUDHAN

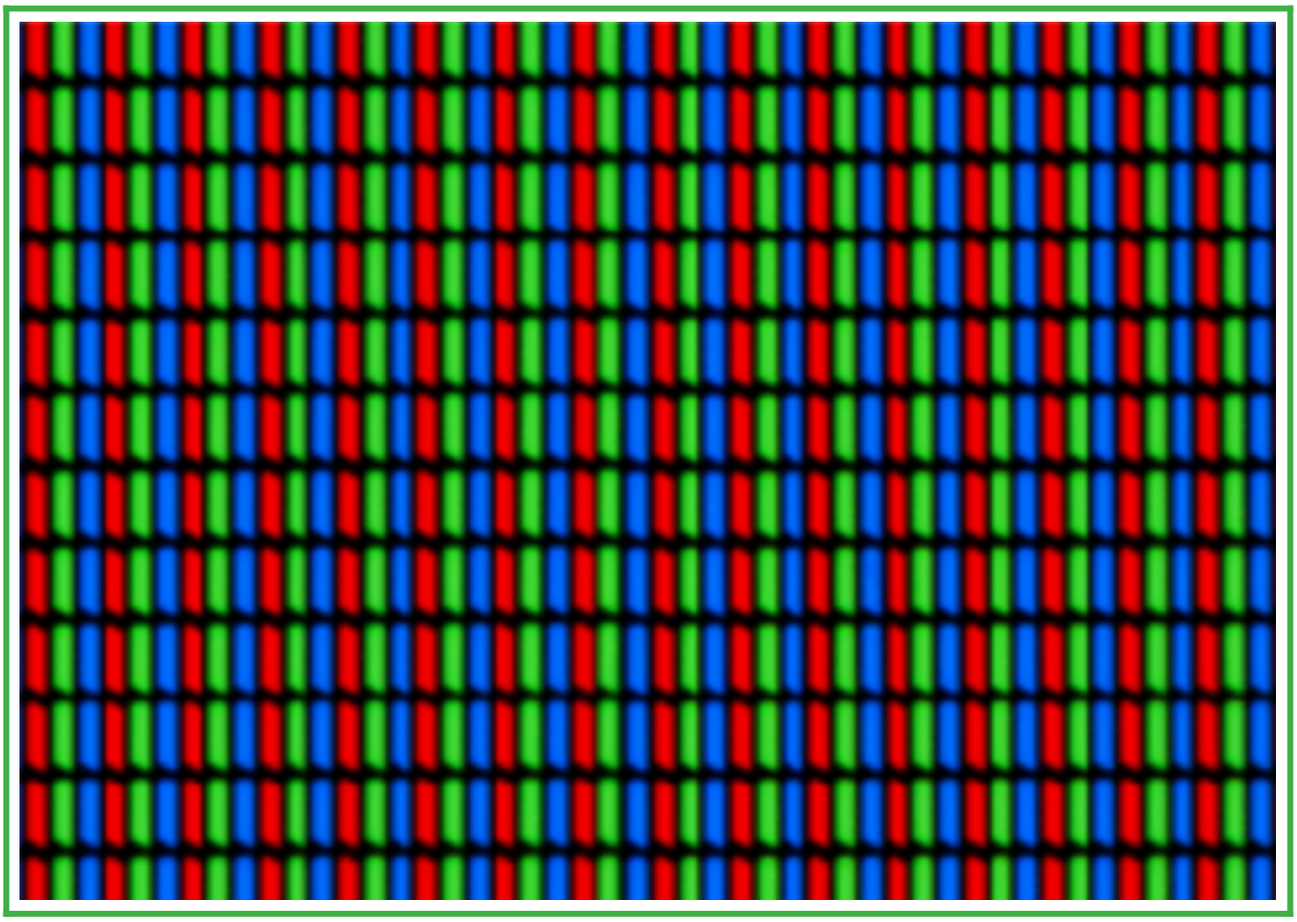
INTRODUCTION

Visual Recognition includes the process of processing the visuals and extracting insights from them. That's all for now !

DATA IN PIXEL FORMAT

The Arrival of Mobile Phones has increased the amount of data a lot. More than 85% of these data are in the format of pixels. We entered the age of VISION!

We all generate pixel data or the visual data everyday!



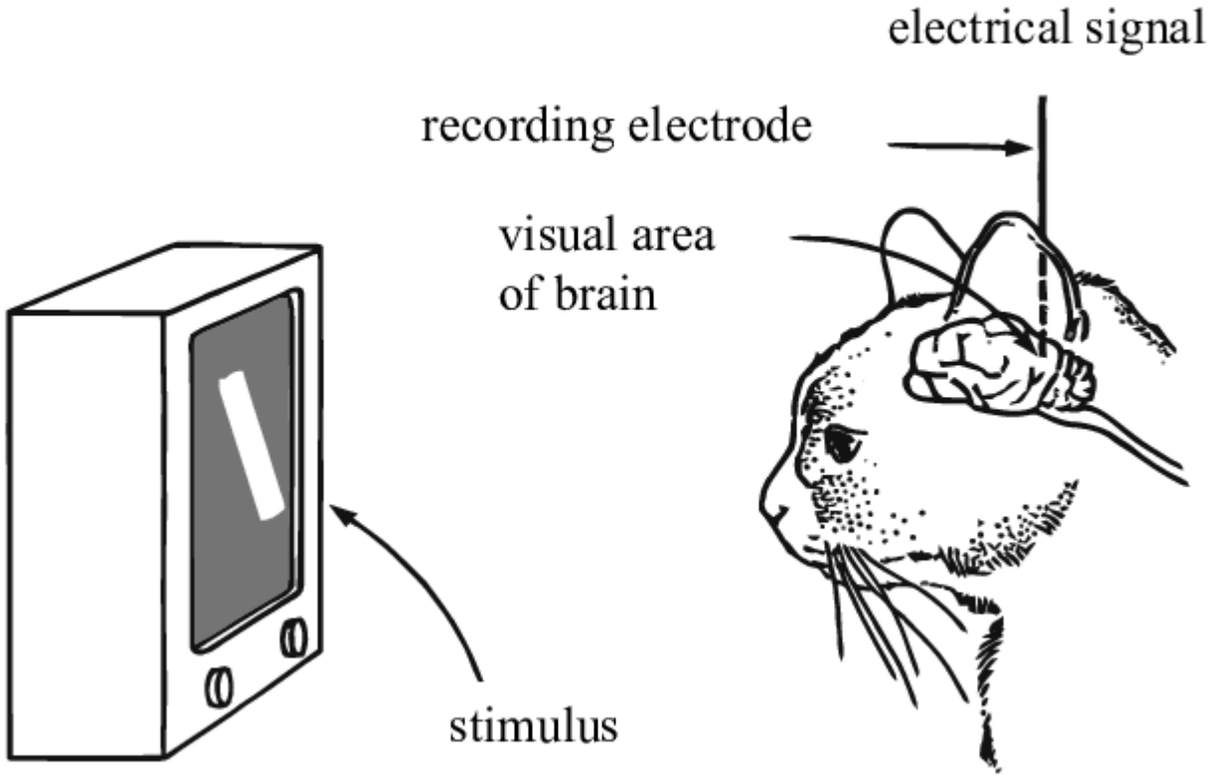
And indeed, the pixel data are really hard to process. When it is in a massive amount… What is the way ? How to perform operations over them ? Everything is obtained from the nature on observing which the solution for anything can be found.

VISION PROCESSING IN US

Brain uses 50% of it for the sake of Vision Processing. It means , it maybe a bit hard for the brain.

The vision processing part is located in the back side of the brain.

NEURAL FIRINGS

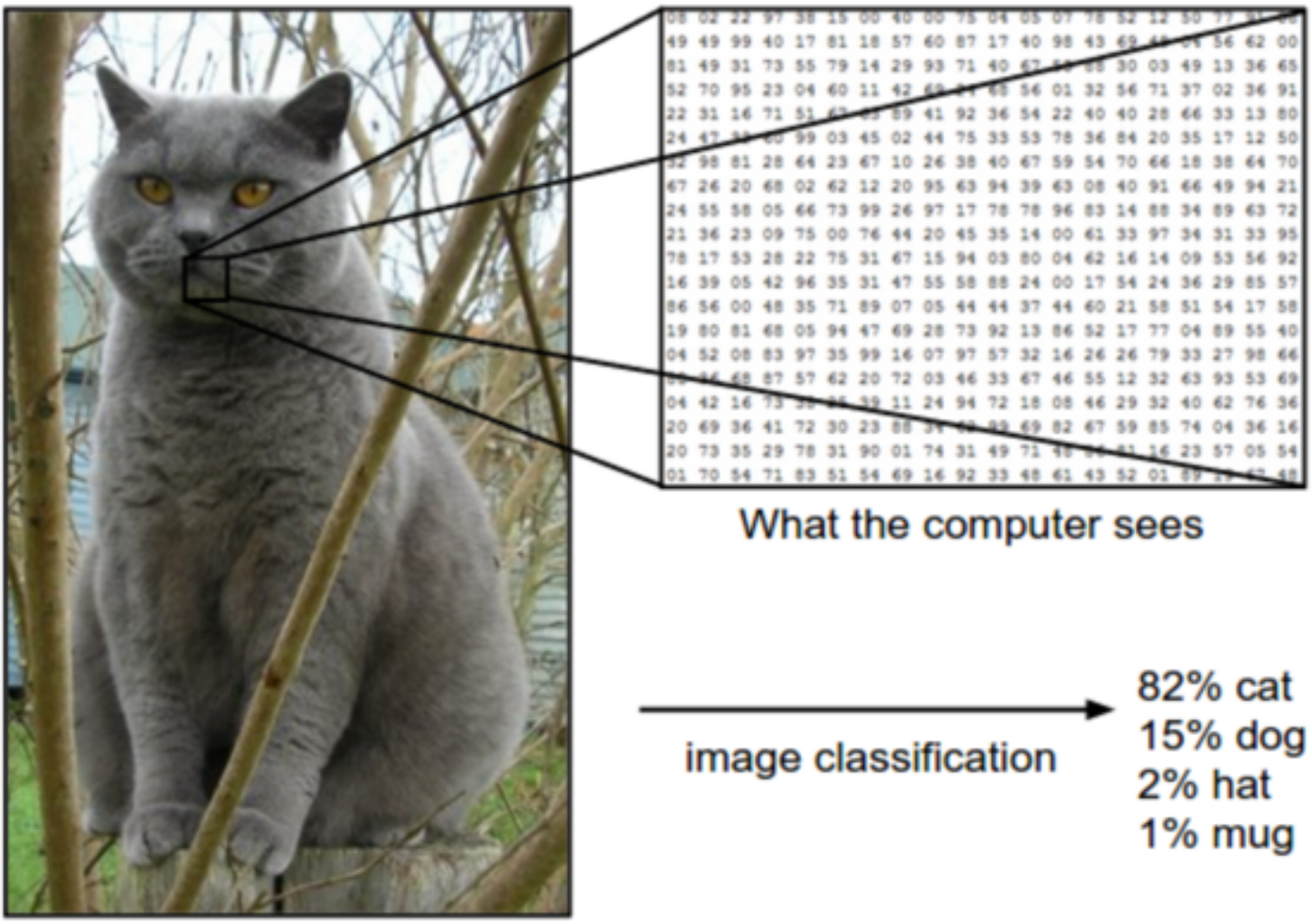


Hubel and Weisel experiments used electrodes to measure neural firings in cat brains in response to visual stimuli. They discovered that neurons in the V1 region are only sensitive to certain areas of a visual field (called "receptive fields"), and detect oriented edges in that area.

For example, they demonstrated that if you showed the cat a vertical bar and start rotating it, at a particular angle the neuron will fire. Similarly, as the bar moves away from that angle, the activation of the neuron diminishes. These activation-selective neurons were named "simple cells", for their ability to detect local features. They also discovered that if you move the bar out of the receptive field, that particular neuron doesn't fire any more, but another neuron will. There are local feature detectors corresponding to all areas of the visual field, hence the idea that the human brain processes visual information as a collection of "convolutions". Another type of neuron, which they named "complex cells", aggregate the output of multiple simple cells within a certain area.

IMAGE CLASSIFICATION

The Image Classification is a task in which we have to deal with a giant grid of numbers. The Computer Vision tasks like Object Detection, Image Captioning are just a variation of the so-called Image Classification.

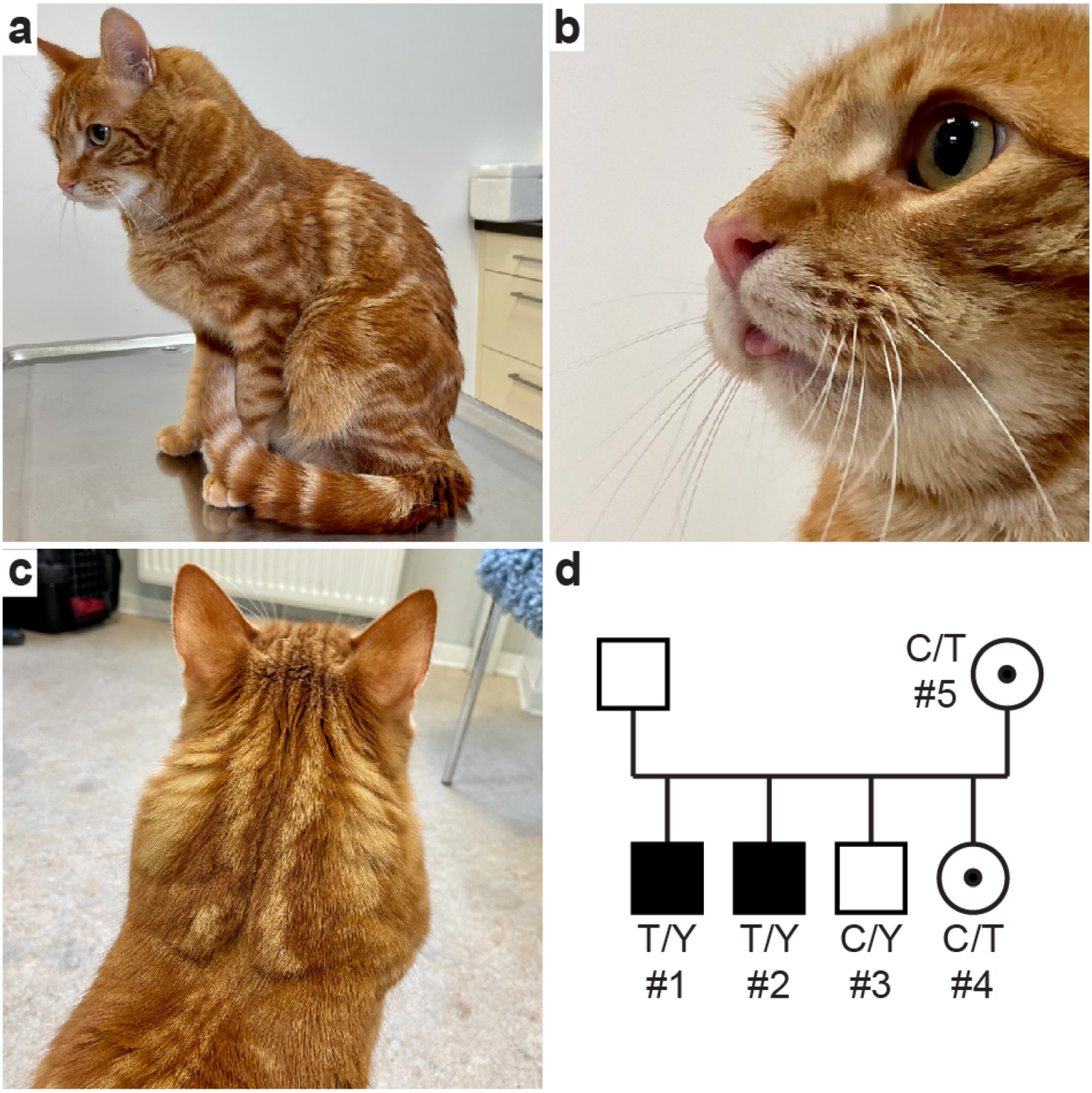


Images are represented as the 3D arrays of numbers , with integers between [0,255].

EG : 300 X 400 X 3

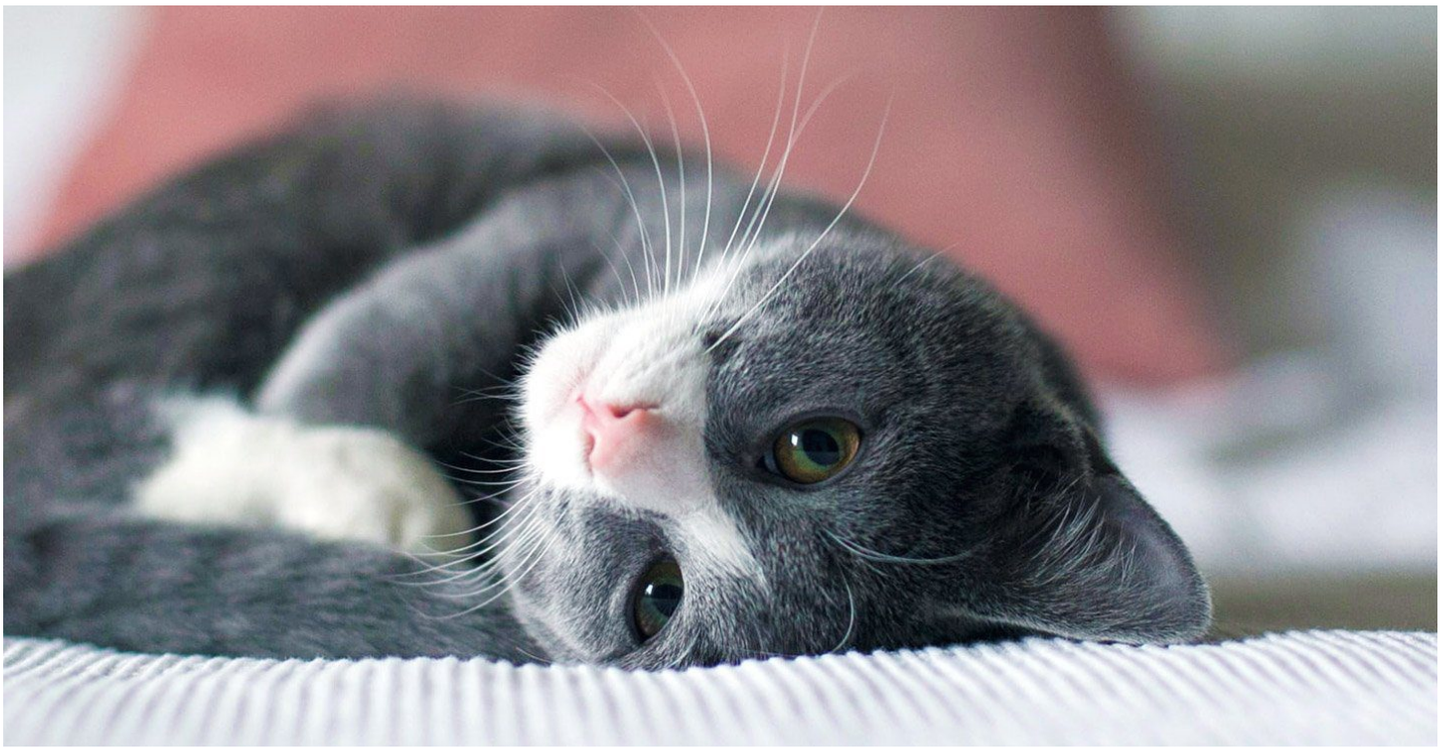
CHALLENGES IN CLASSIFICATION

★ View-Point Variations



★ Illuminations

★ Deformations



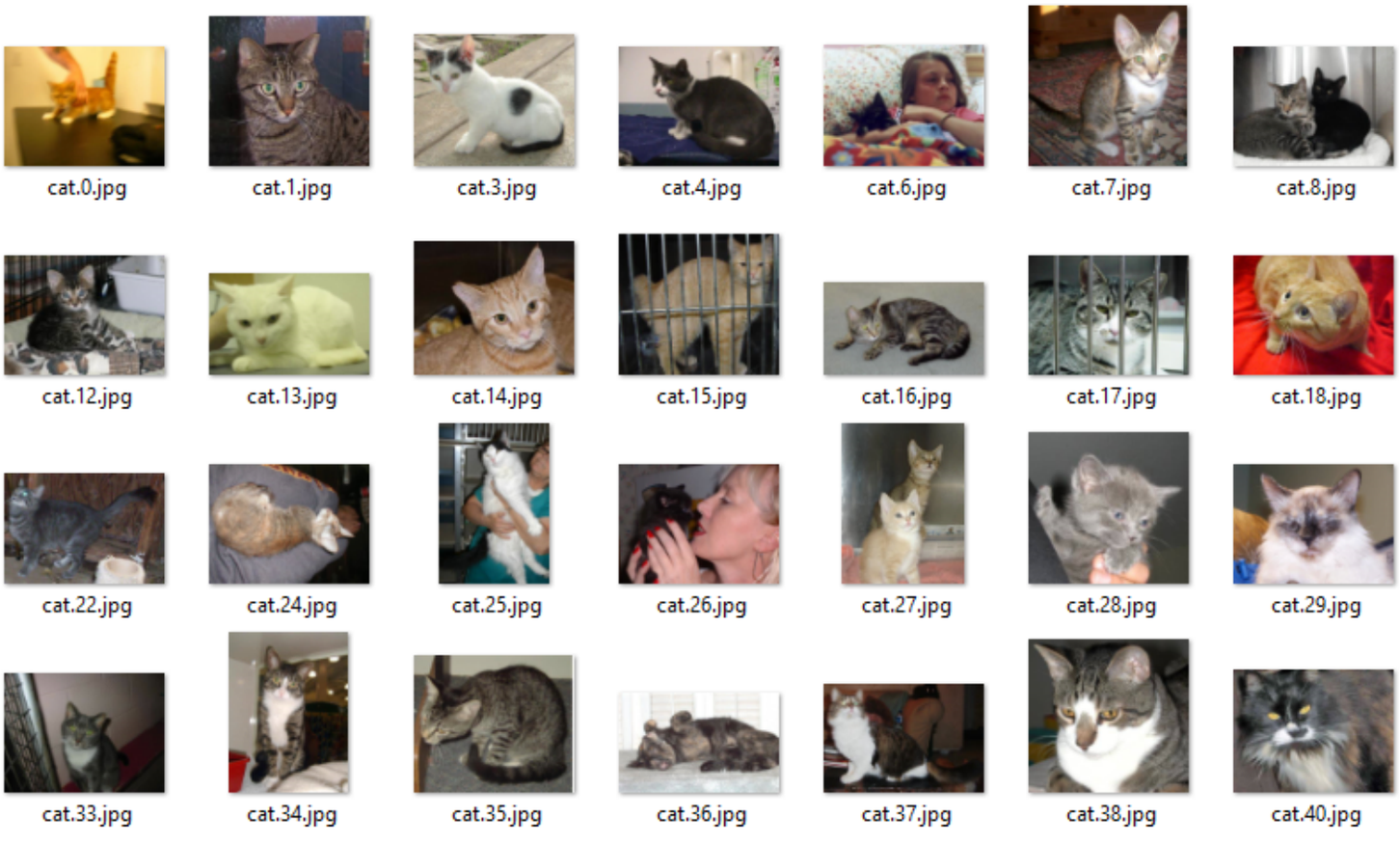
★ Occlusion

★ Background Clutter

★ Intra-Class Variations

THE DATA DRIVEN APPROACH

★ Collect a Dataset of Images & Labels



- ★ Use ML to train The Image classifier
- ★ Evaluate The Classifier with Test Set

THE KNN ALGORITHM

How can we conclude that the test image is similar to the trained one ? We, humans are able to do this very easily. What about the computer? We have some methodologies for this. We have to find out the distance metric of two images.

Distance functions

Euclidean

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Let's consider the Manhattan distance function. What we have to do is finding the element wise difference of the two images. Here, we refer the pixel as an element. On finding the individual differences of all, add them all. The result defines the similarity of both images. If it is high, the images are very less similar and vice versa.

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} - \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} a_1-b_1 & a_2-b_2 & a_3-b_3 \\ a_4-b_4 & a_5-b_5 & a_6-b_6 \\ a_7-b_7 & a_8-b_8 & a_9-b_9 \end{bmatrix}$$

SUM: (a1-b1)+(a2-b2)+(a3-b3)+(a4-b4)+(a5-b5)+(a6-b6)+(a7-b7)+(a8-b8)+(a9-b9)

We have to iteratively find out the label for which the distance metric is less. Thanks to the powerful vectorization in NumPy. One more thing to remember here is that the training is a way faster than the process of testing. It is a sort of lazy learning algorithm.

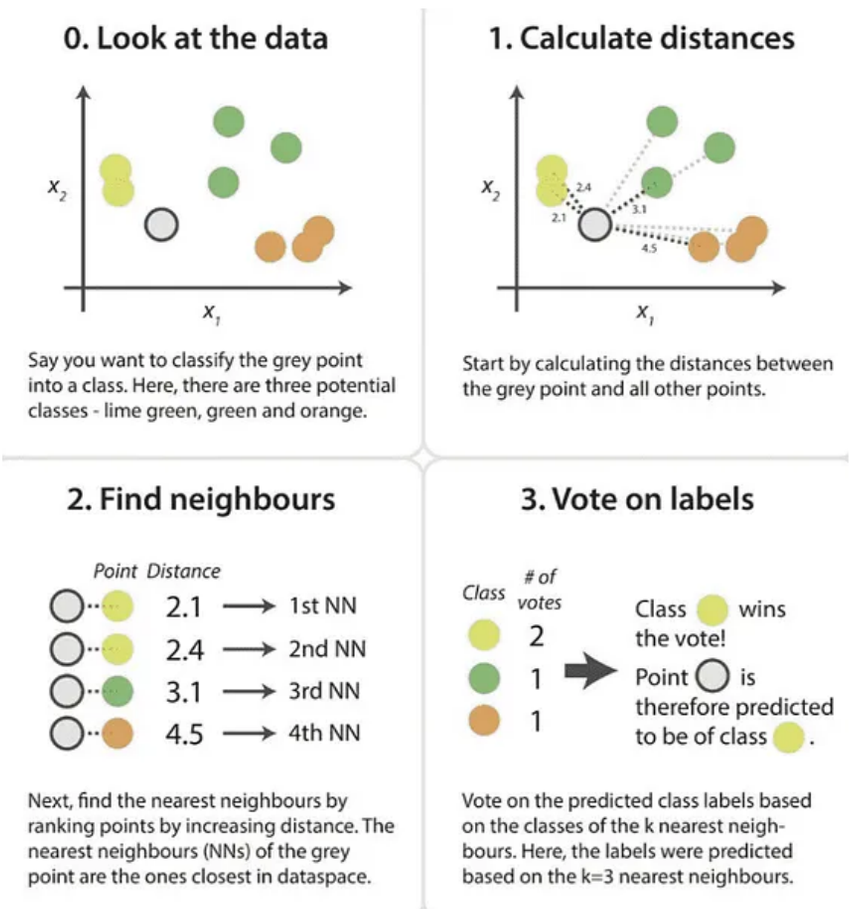
What we have learned can be applied as,

```
In [5]: import numpy as np

        class NearestNeighbour:
            def __init__(self):
                pass
            # TRAINING
            def train(self,X,Y): # (Images,Labels)
                self.Xtr = X
                self.Ytr = Y
            # PREDICTIONS
            def predict(self,X):
                num_test = X.shape[0]
                Y_pred = np.zeros(num_test, dtype = self.Ytr.dtype)
                for i in xrange(num_test):
                    distances = np.sum(np.abs(self.Xtr-X[i,:]),axis=1)
                    min_index = np.argmin(distances)
                    Y_pred[i] = self.Ytr[min_index]
                return Y_pred  # PREDICTIONS
```

In a single line of code, we compare every train-images with the test-image. This is the power of VECTORIZATION. The k value in the KNN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. It is definitely a sort of VOTING PROCESS. The choice of k , The choice of distance are Hyper-Parameters.

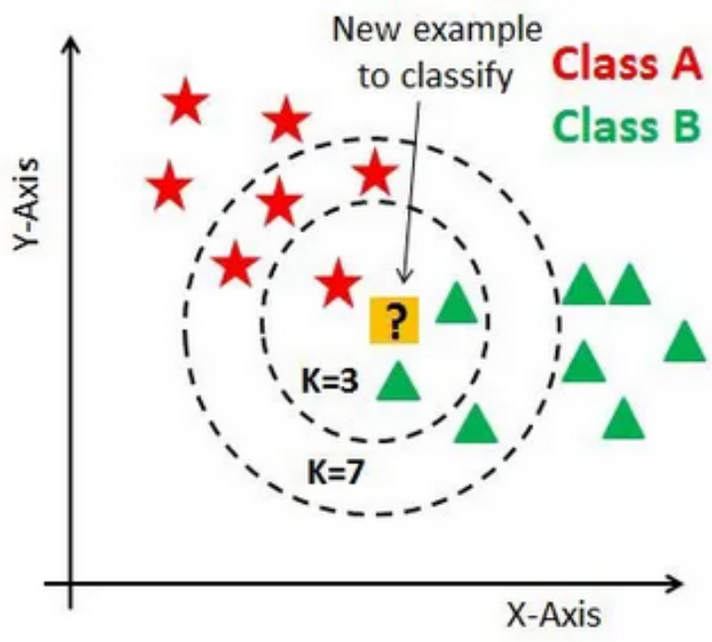
STEPS FOLLOWED ARE,



The K-NN working can be explained on the basis of the below algorithm :

- **Step-1: Select the number K of the neighbors**
- **Step-2: Calculate the distance of K number of neighbors**
- **Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.**
- **Step-4: Among these k neighbors, count the number of the data points in each category.**
- **Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.**
- **Step-6: Our model is ready.**

K MATTERS A LOT



If we have the k to be 3, then the 3 closest neighbors are two triangles and one star which will decide the one to be classified as a triangle. But, when we have the k to be 7, the 7 closest neighbors are 3 triangles and 4 stars which will decide the one to be classified as a star. Can you understand the importance of the hyper-parameter k ? So how to select an optimized k value. Some suggestions are, ★ Select a substantial k value ★ Plot error-rate against the k value and select the k value such that the error-rate is minimum.

THE CROSS VALIDATION

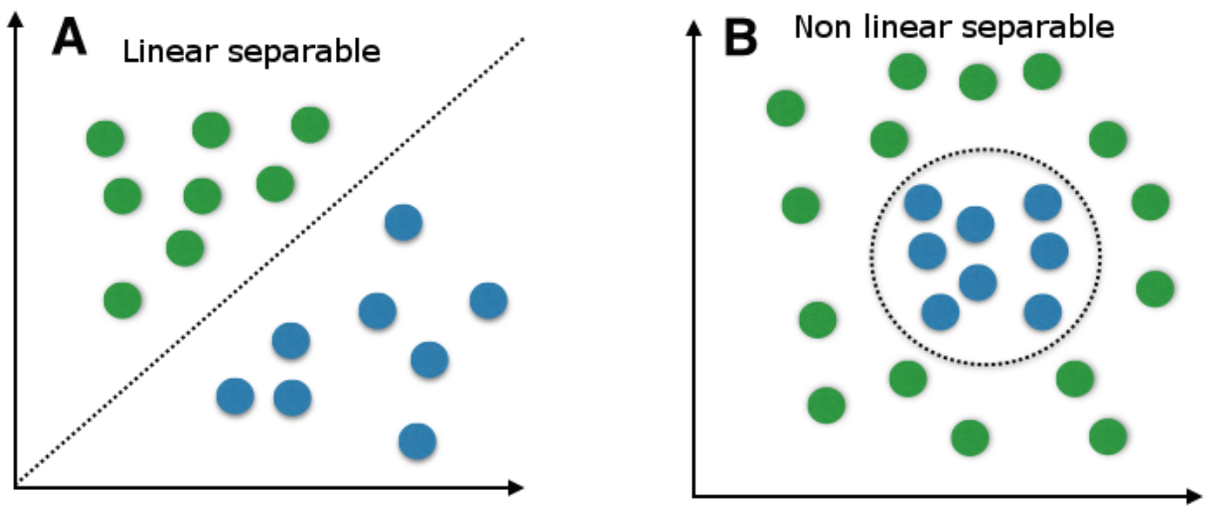
If you think about KNN, we use the test data to basically determine the right value of K and the train data to find the nearest neighbors. If we got the accuracy of 90% on the test data, Can I say that since I achieved an accuracy of 90% on my test data, I will be able to retain this approximate level of accuracy on my future unseen data? The answer is **DEFINITELY NO**.

Cross-validation is when the dataset is randomly split up into ‘k’ groups. One of the groups is used as the test set and the rest are used as the training set.

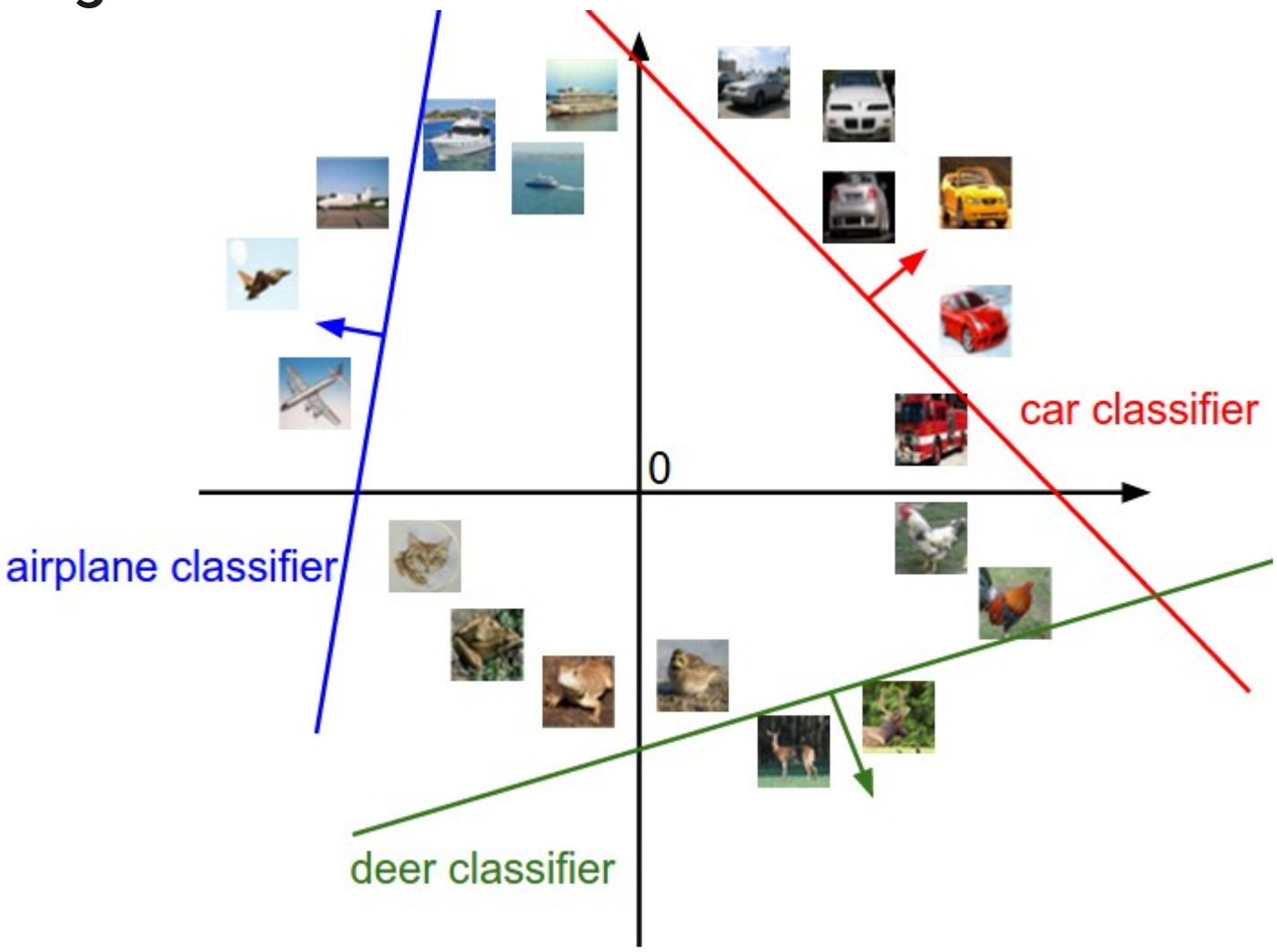
The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group has been used as the test set. For example, for 5-fold cross validation, the dataset would be split into 5 groups, and the model would be trained and tested 5 separate times.

THE LINEAR CLASSIFICATION

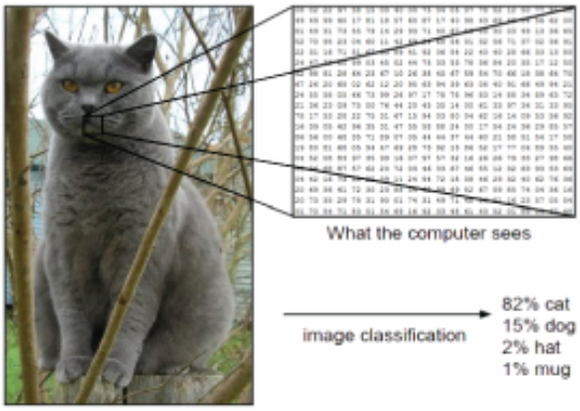
A linear classifier does classification decision based on the value of a linear combination of the characteristics. This type of classifier works better when the problem is linear separable.



The weight matrix will have one row for every class that needs to be classified, and one column for every element (feature) of x. On the picture below, each line will be represented by a row in our weight matrix.

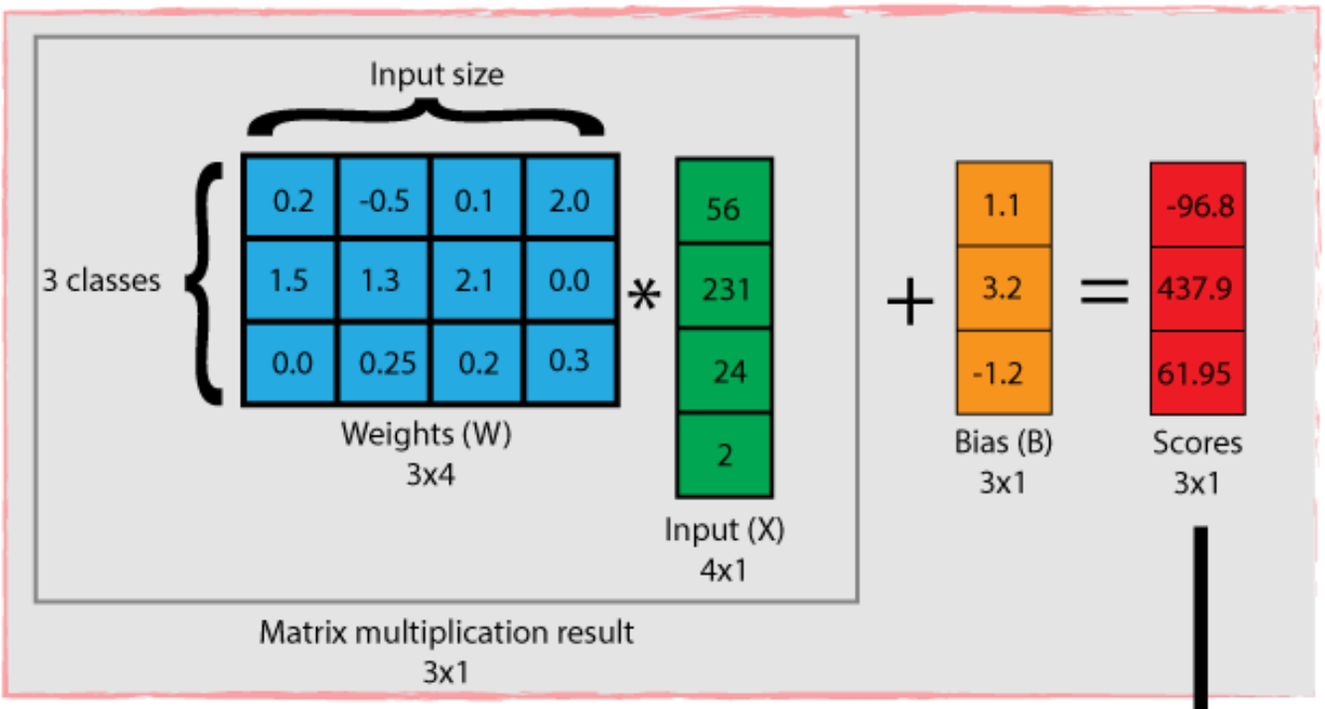


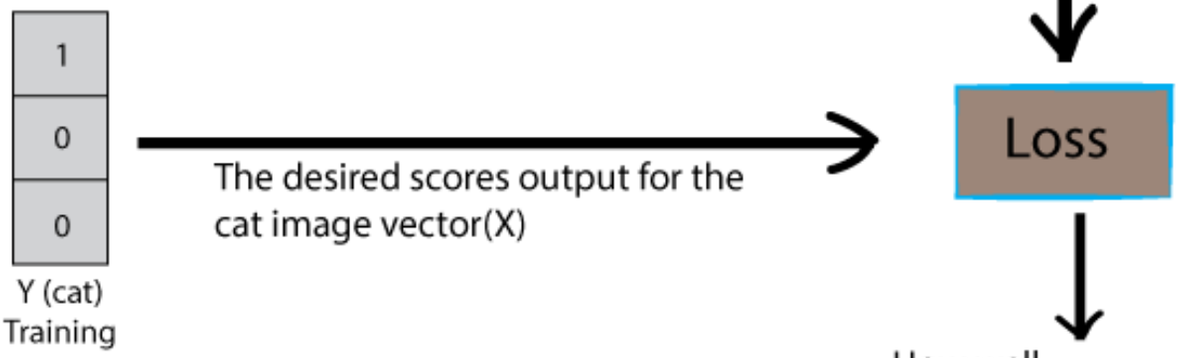
The effect of changing the weight will change the line angle, while changing the bias, will move the line left/right.



Idea transform a vector of numbers into scores for each possible class.
On this case we transform the matrix of numbers (image) on a 1d vector (All spatial information is lost)
On this example our image became an array 4x1, and we need to classify images as (cat,dog,ship)

Linear model f(x,W,b)





How well
the parameters W, B
map the input to the desired output

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Given an example (x_i,y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector : s = f(xi,W).
The SVM/Hinge loss is :

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

For the CAT Class,

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

For the CAR Class,

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

For the FROG Class,

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

Now, let's average these all numbers.

Full Training Loss

= (2.9 + 0 + 12.9)/3

= 4.6

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

What if the SUM is over all the classes ?

What will happen if so? We'll be adding a one inflating the sum.

What if we use the MEAN instead of the SUM ?

The Absolute value of the loss will be not meaningful.

What if we have SQUARED the expression ?

It is called the Squared Hinge Loss. It can also be used.

What is the MAX/MIN possible loss ?

The Minimum loss will be 0 and the Maximum loss will be infinite (when the output for correct data is much much lower).

SOME OPTIMIZATIONS

What if we have larger values of w? How will it impact ?

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Before:

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

With W twice as large:

= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

Regularization is simply simplifying the model by scaling down the weight values.

"Among competing hypotheses, the simplest is the best"

A Regularized Expression of Loss Function would be like :

Regularization

λ = regularization strength (hyperparameter)

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

We have some weight-decay regularizations namely L2 and L1-Regularizations.

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Both are used to retard the weight values.

L2 regularization makes the weight values to be spread over all the input dimensions.

$$x = [1, 1, 1, 1]$$

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

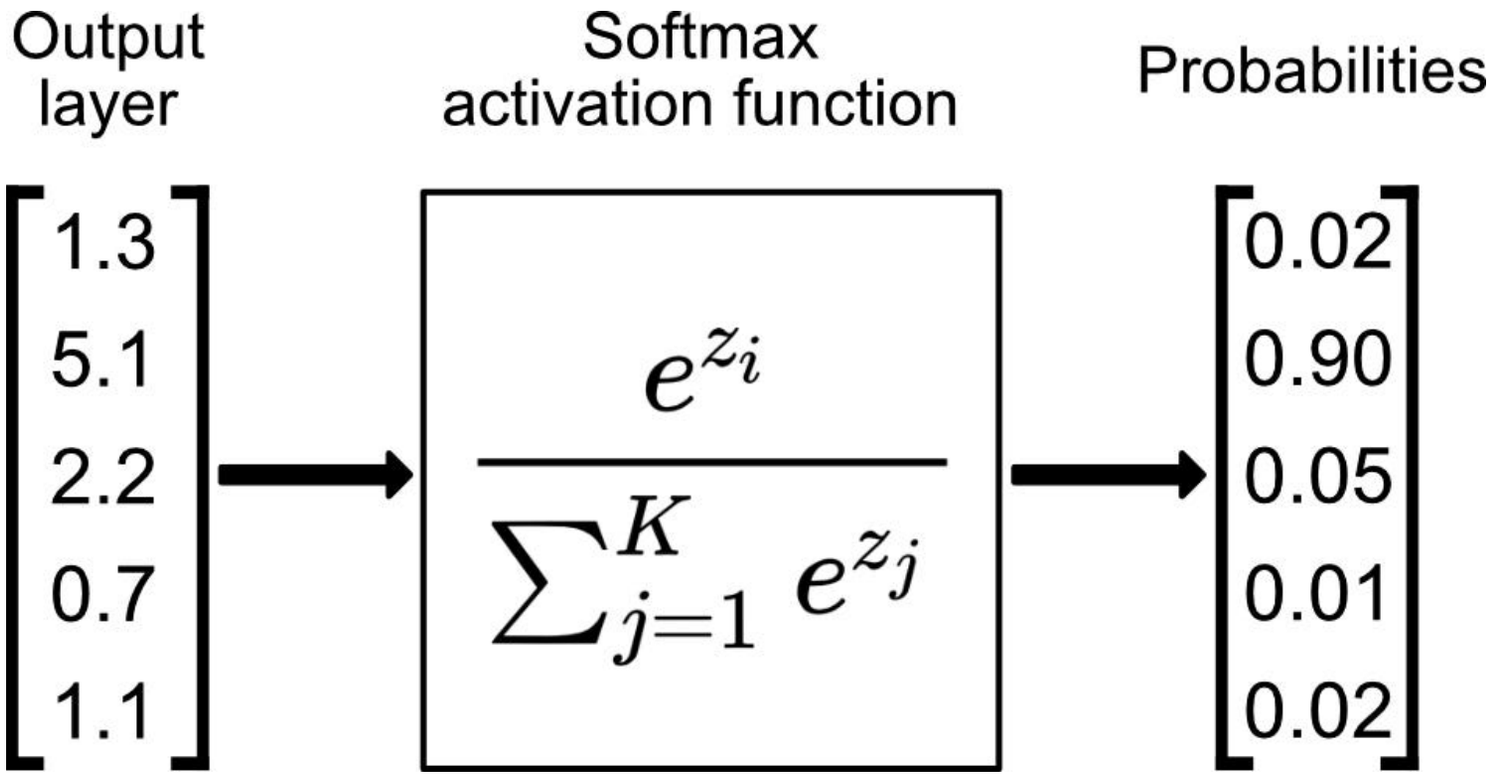
$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

We can notice that more w values are diffused in w2. The more evidence or a feature we accumulate, the more precise output we get.

THE SOFTMAX CLASSIFIER

The real values are converted into probabilities on the basis of which a particular object is recognized.



Log-Likelihood value is **a measure of goodness of fit for any model**. Higher the value, better is the model.

scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y=y_i|X=x_i)$$

---

in summary: $$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

THE LEGEND OF GRADIENT DECENT

Assume we have a ball which is put in a particular place of a mountain. We know it will definitely roll down. But, in which direction ? It's the direction on going which the ball will land quickly. This is the so-called Gradient Decent. We have to find out this. One of the worst approach is using a Random Search which is a trial and error algorithm which takes a long time to find out the appropriate solution. But, in simple words, the gradient decent is a function of weight. We can find the gradient numerically which takes a long time and approximate.

(1.25322 - 1.25347)/0.0001
= -2.5

$$\boxed{\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

We can use calculus to find out the analytic gradient which should be checked out with the numeric gradient.

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

THE STOCHASTIC GRADIENT DECENT

Full sum is expensive when N is large! Approximate the sum using a mini-batch of examples 32 / 64 / 128 common. In essence, we can define it as, “Using only a set of the training set to compute the gradient decent”.

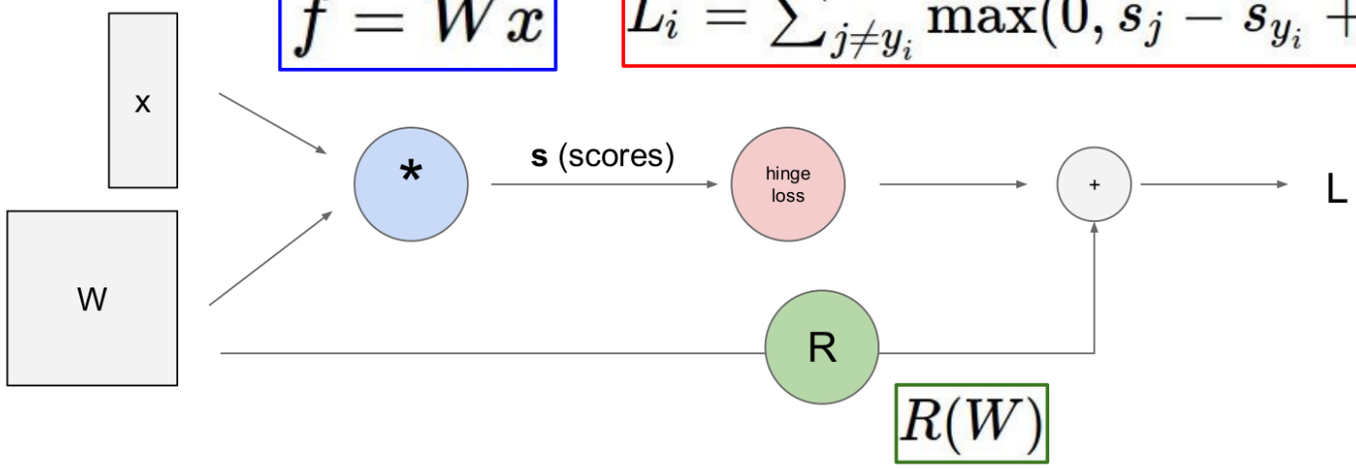
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

I AM A BACK-PROPAGATION NINJA

$$f = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



Let's find out the derivatives of the f with respect to the intermediates.

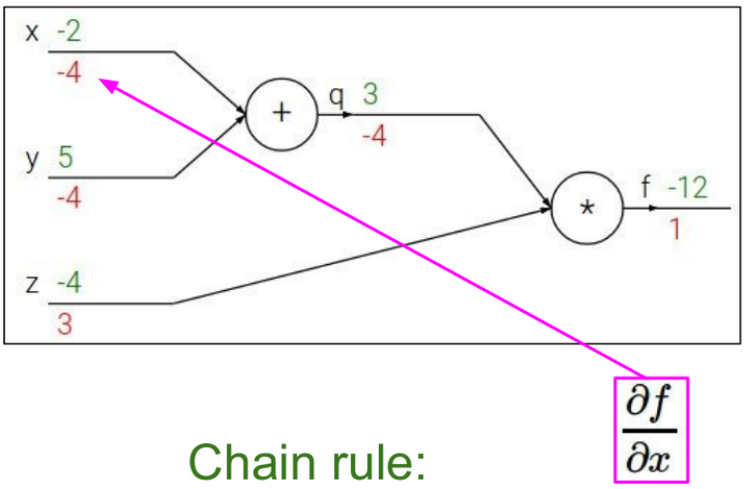
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

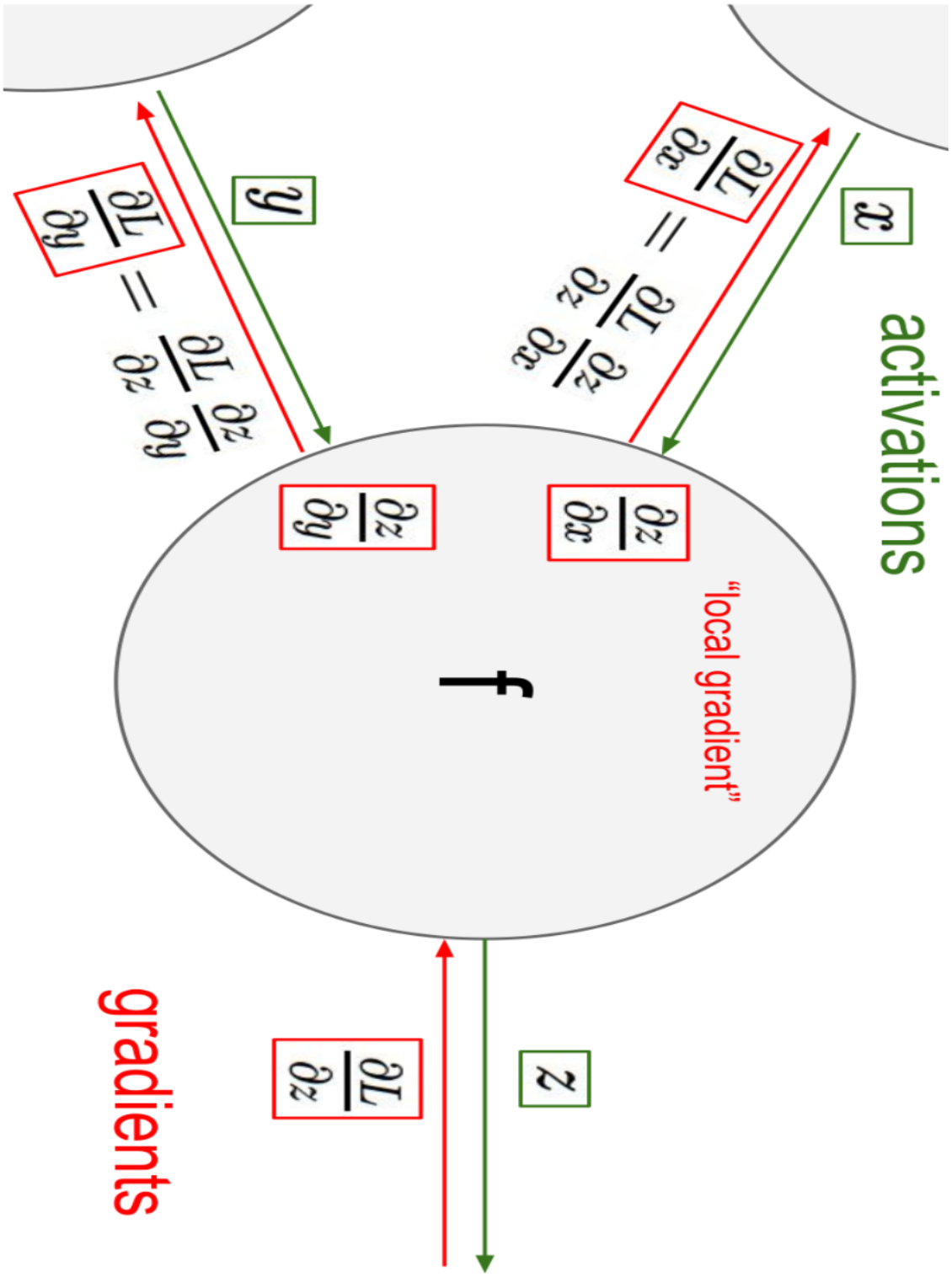


$$\frac{\partial f}{\partial x}$$

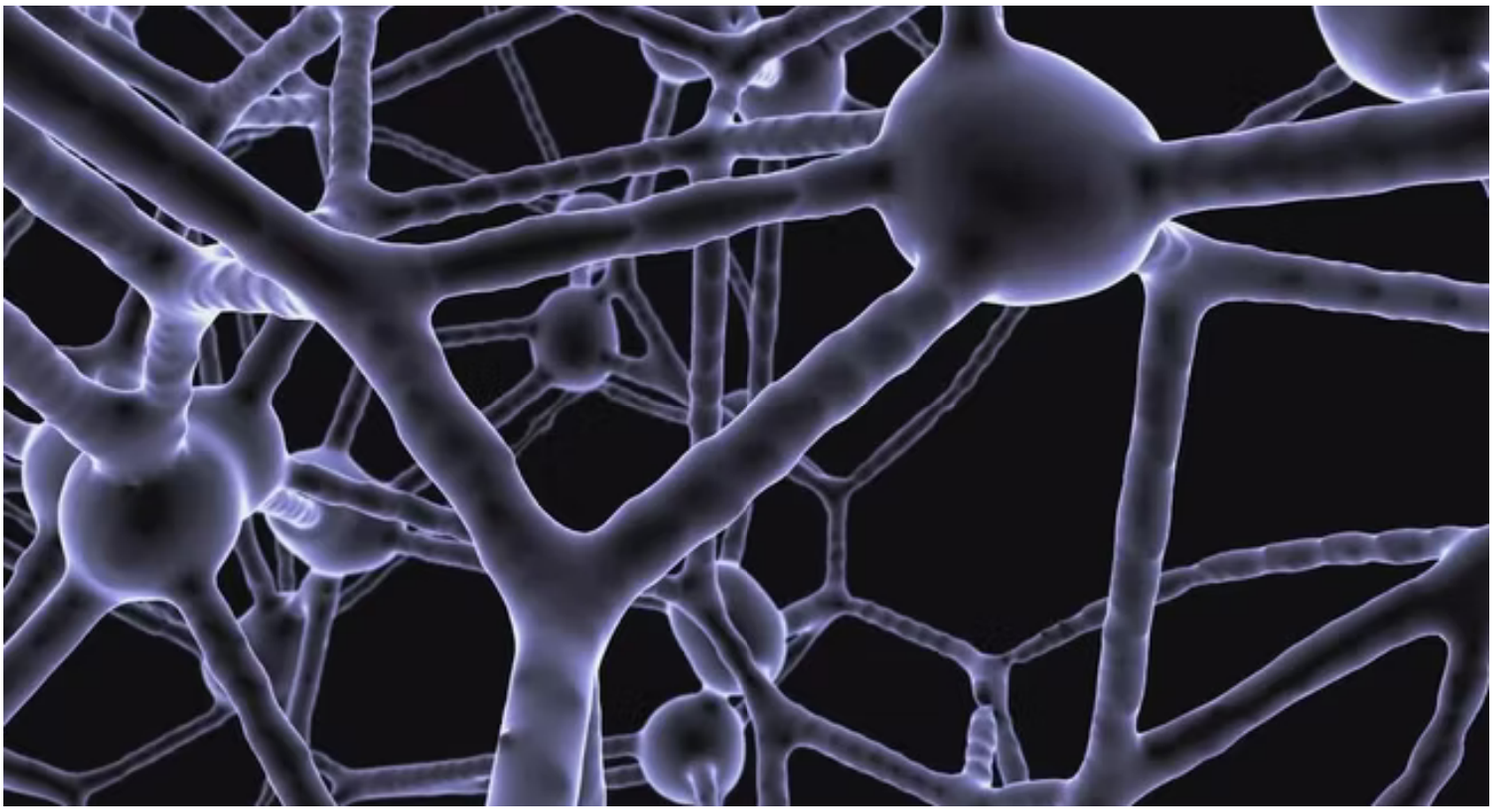
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

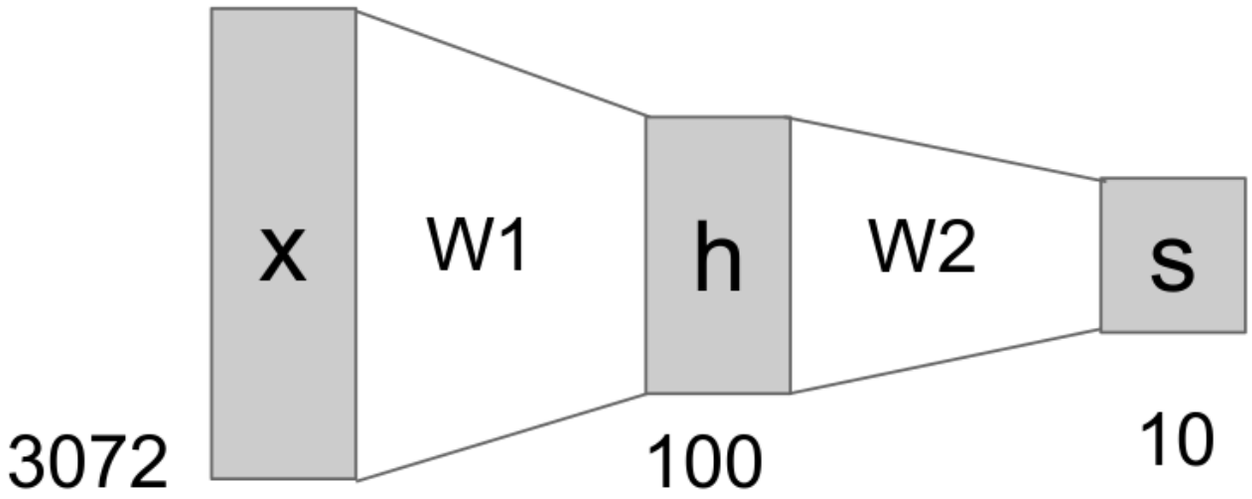
And remember that the change in one layer will definitely affect the layers next to it.



THE NEURAL NETWORK



Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are **a subset of machine learning and are at the heart of deep learning algorithms**. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.



It is a simple two layer network. Learn more about neural network at :

arihara-sudhan.github.io/books.html

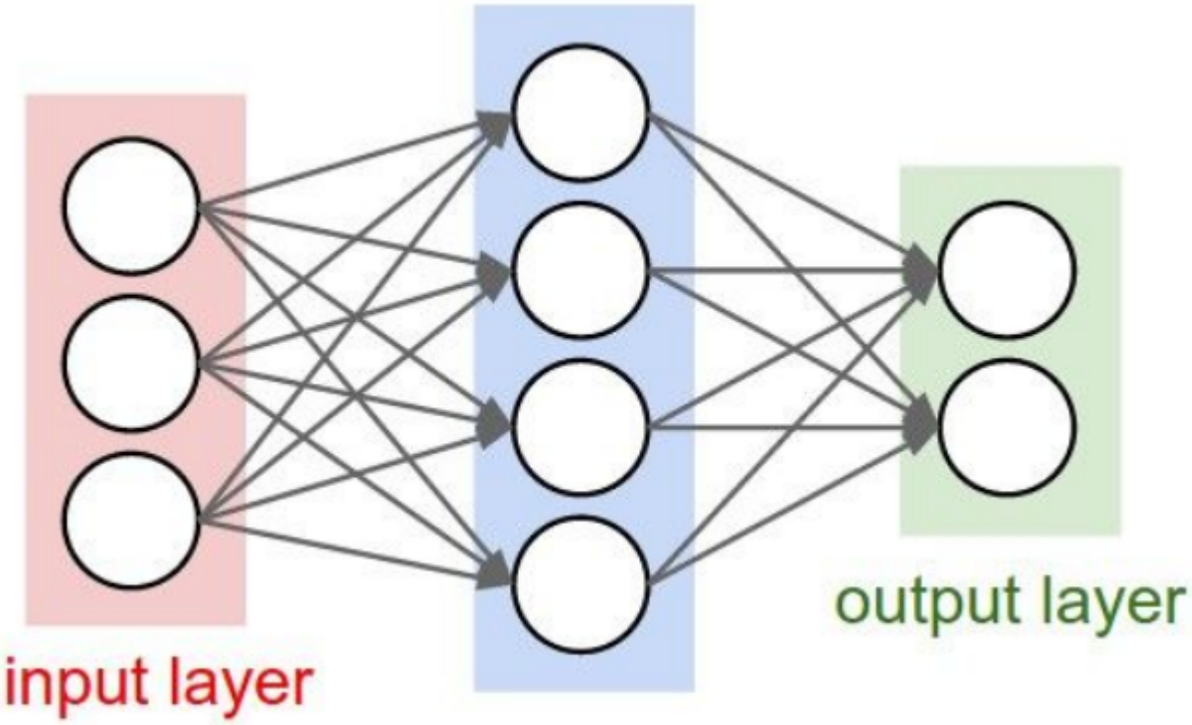
We can train the neural network in just 11 lines of Python Code.

```
X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
y = np.array([[0,1,1,0]]).T
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,1)) - 1
for j in xrange(60000):
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
    l2_delta = (y - l2)*(l2*(1-l2))
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
    syn1 += l1.T.dot(l2_delta)
    syn0 += X.T.dot(l1_delta)
```

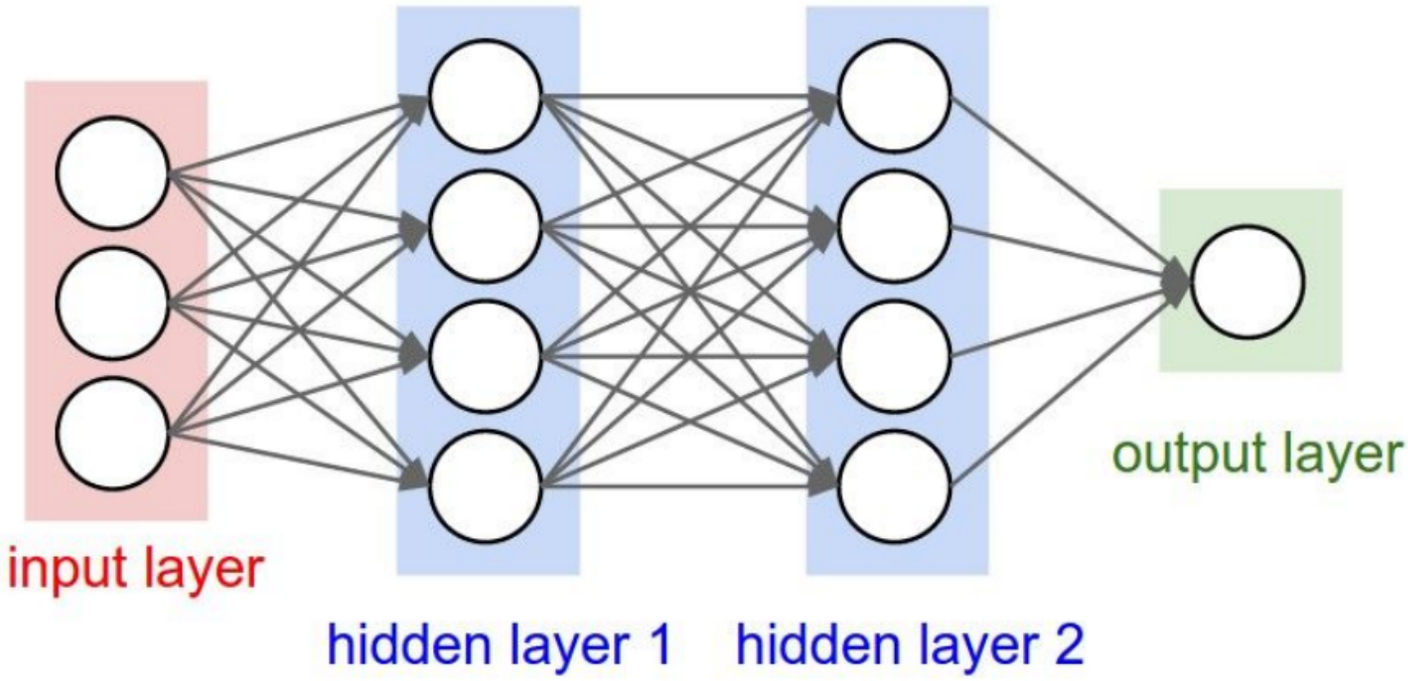
We can add non-linearity to our neural network using The Activation Functions.

arihara-sudhan.github.io/articles.html

A Neural Network has the following architecture styles.



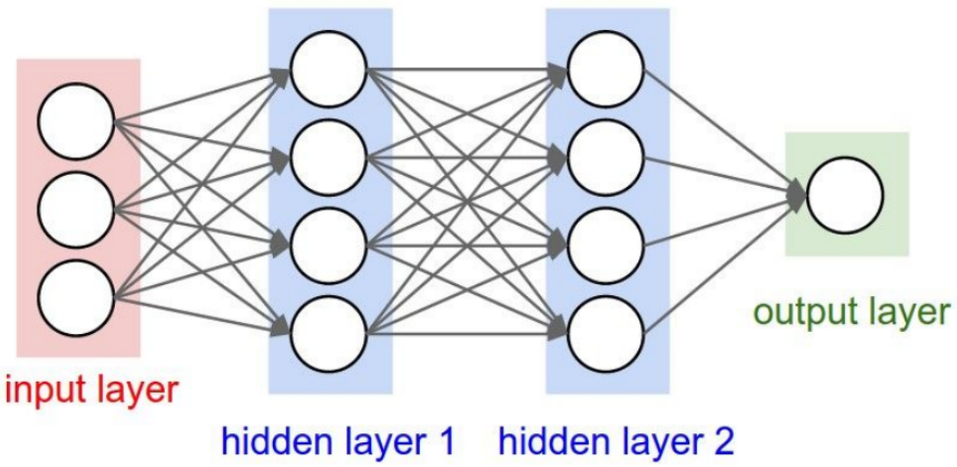
The one shown above is called as , Fully Connected Neural Network. Every Neurons in one layer is connected to Every Neurons in the next layer.



The one shown below is called as, A 3 Layer Neural Network or as A 2 Hidden Layer Neural Network.
An Example for the Feed Forward Network can be given as,

```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

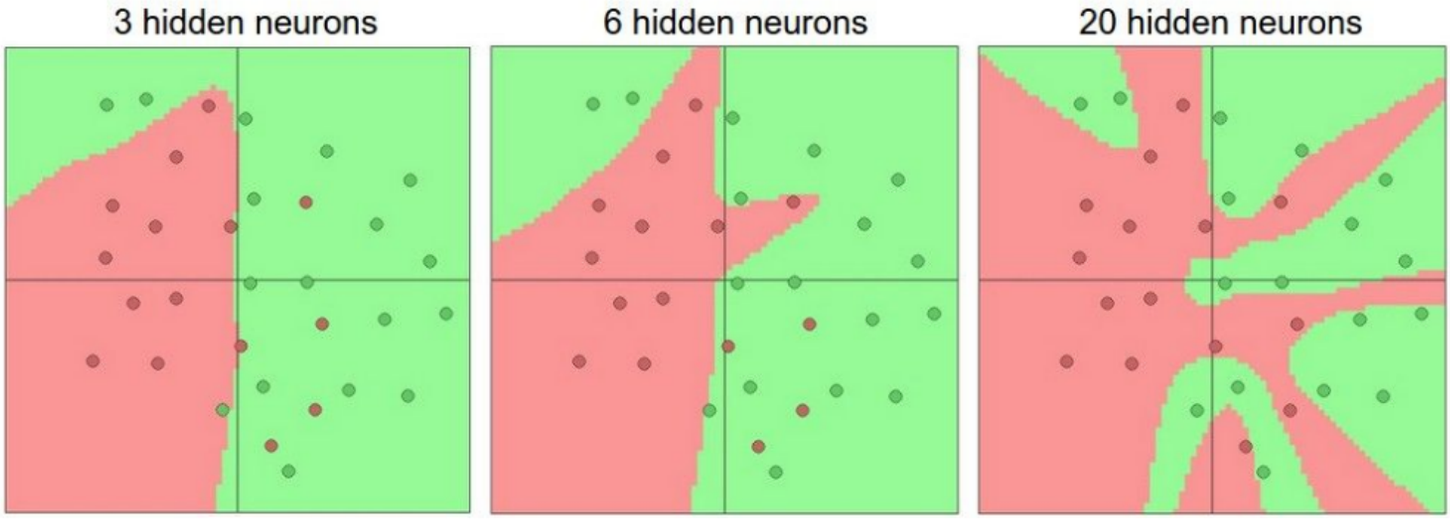
If we have a two layer neural network, the forward pass would be like,




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

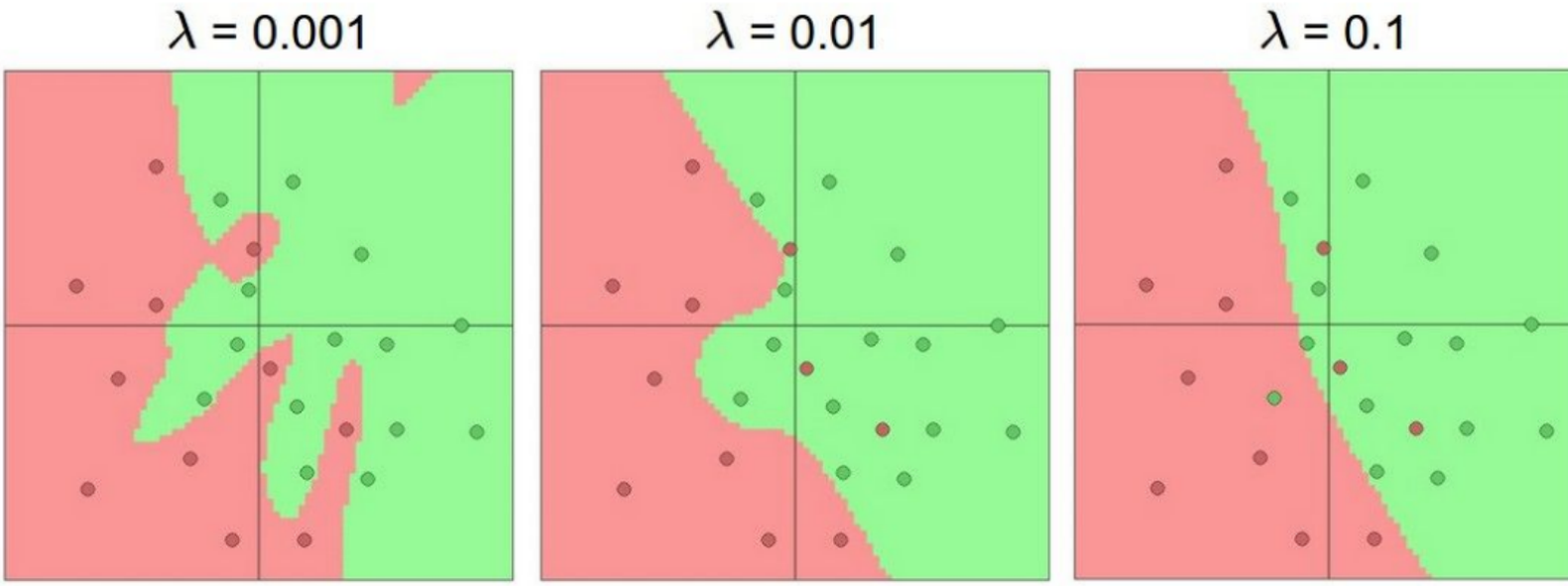
Why do we need this layered structure for a neural network ? Simply, to achieve the vectorized parallel computing.

More Neurons give us more capacity!



As we can observe from these figures, when we have many hidden neurons, the more detailed result we get!

Penalizing your weight also matters!



THE WEIGHT INITIALIZATION

It is very very important to initialize the weights properly to obtain expected result. Variance is a measure of how data points differ from the mean. According to Layman, a variance is a measure of how far a set of data (numbers) are spread out from their mean (average) value. Variance means to find the expected difference of deviation from actual value. Weights should be small but not too small as it gives problems like vanishing gradient problem. That is it will take forever to converge to global minima. Weights can't be too high as gives problems like exploding Gradient problem (weights of the model explode to infinity), which means that a large space is made available to search for global minima hence convergence becomes slow.To prevent the gradients of the network's activations from vanishing or exploding, we need to have following rules: The *mean* of the activations should be zero. The *variance* of the activations should stay the same across every layer.

```
W = 0.01* np.random.randn(D,H)
```

Weights should be different, these can't be same as it gives problems like all neurons will behave in same way. Also if these will be initialized as zero they will not learn anything.

THE BATCH NORMALIZATION

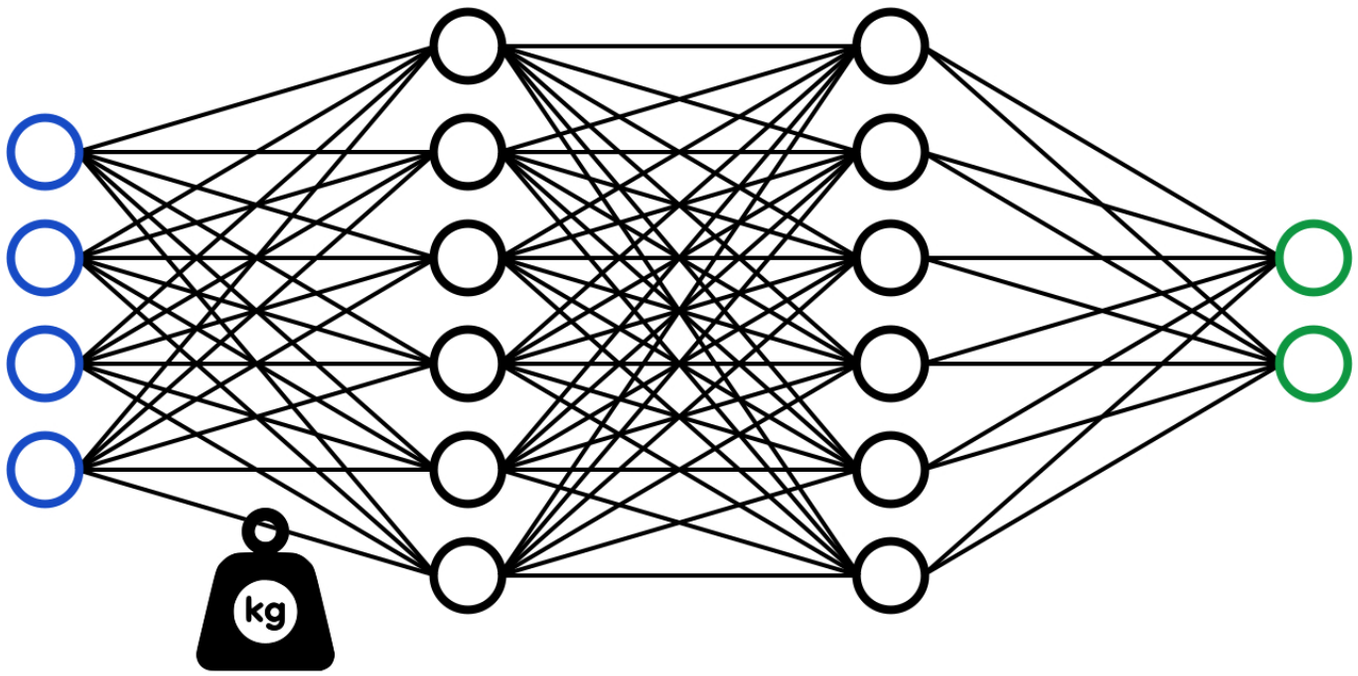
Both of the following have the same goal of put all the data-points on the same scale.
@THE NORMALIZATION : A Normalization process converts the data ranging from 0 to 100 into 0 to 1. (DONE BEFORE FEEDING IN)

@THE STANDARDIZATION : It consists of the processes of subtracting the mean from the data-point and dividing by the standard deviation.

Why should we normalize the data ?

Assume we have two features one describing the number of miles an individual has traveled in the last five years ranging from 1000 to 100000 and the another describing his age ranging from 10 to 100, we could feel a distance which is really more. When we feed these stuffs into a Neural Network, there arises an instability. The instability here means, an imbalanced gradient scenario which may lead the so-called Exploding Gradient Problem.

So, to avoid these, we put all our data in a same appropriate scale value which is OKAY for the Neural Network. Well!
Then, when is Batch Normalization? Assume we have a quite-changed value of weight in a particular layer. i.e., Let's say we have a weight in the layer1 which is greater than others.



What will happen ? This imbalance will cause some noises to the layers next to it. Here, we have to apply the Batch Normalization. The output of the activation function is batch-normalized. That output is multiplied and added with an arbitrarily selected values.

FINE TUNING

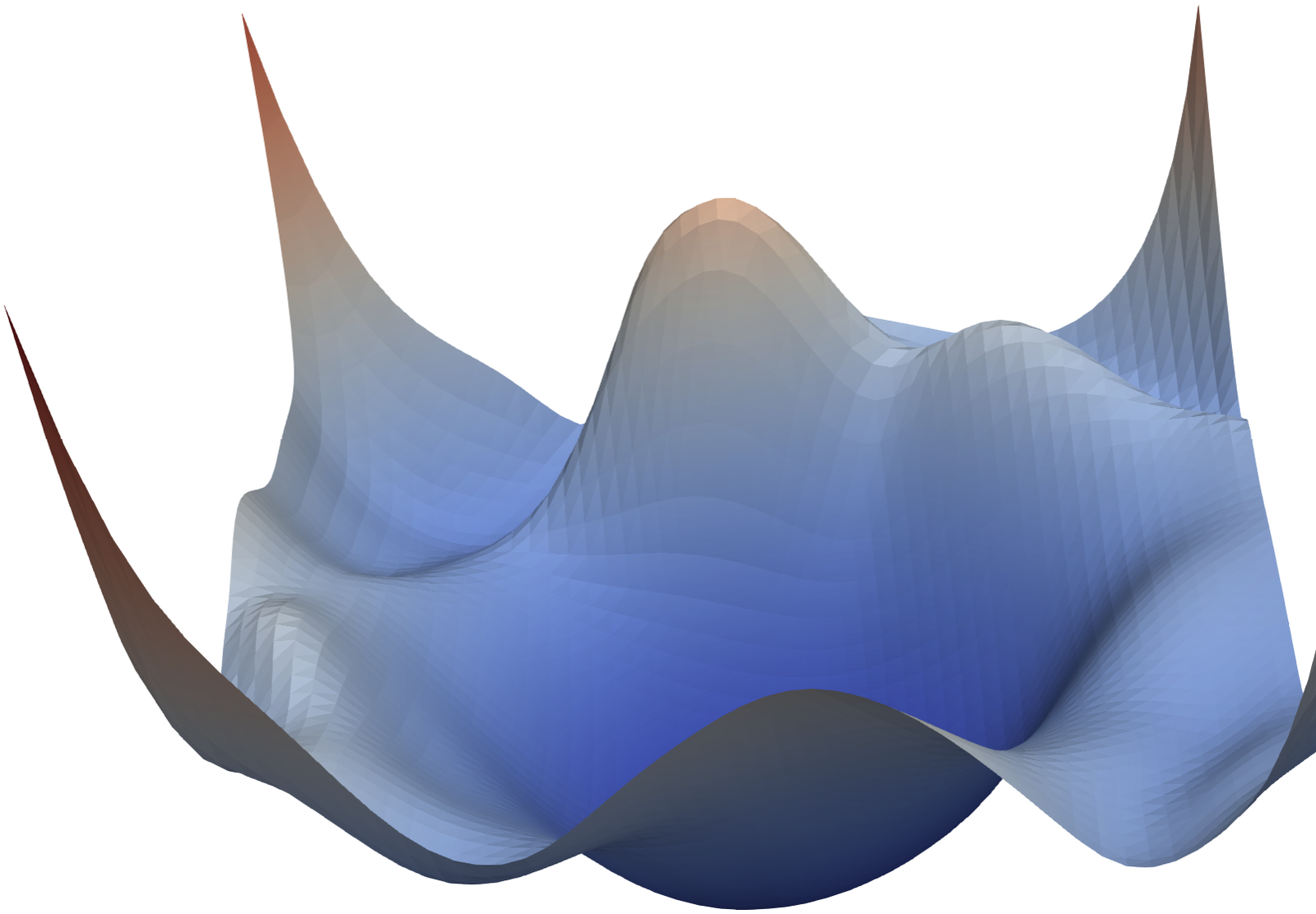
Fine Tuning is linked with the Transfer Learning. Transfer learning is a phenomenon in which the knowledge acquired from solving a problem can be applied to solve another related problem.

For an instance, if our model can predict a monitor lizard, can't it predict a gecko lizard ?

Fine Tuning is the process that takes the model that was trained for a particular task and tweaking and tuning it to perform well for a second-similar task. (The New Task should be similar to the Old). The advantage is we don’t need to develop a model from the scratch. In a nutshell, fine-tuning allows us to fit our custom dataset onto a pre-trained model, so that the model can generalize its output to our particular task.

SGD WITH MOMENTUM

In deep learning, we have used stochastic gradient descent as one of the optimizers because at the end we will find the minimum weight and bias at which the model loss is lowest.

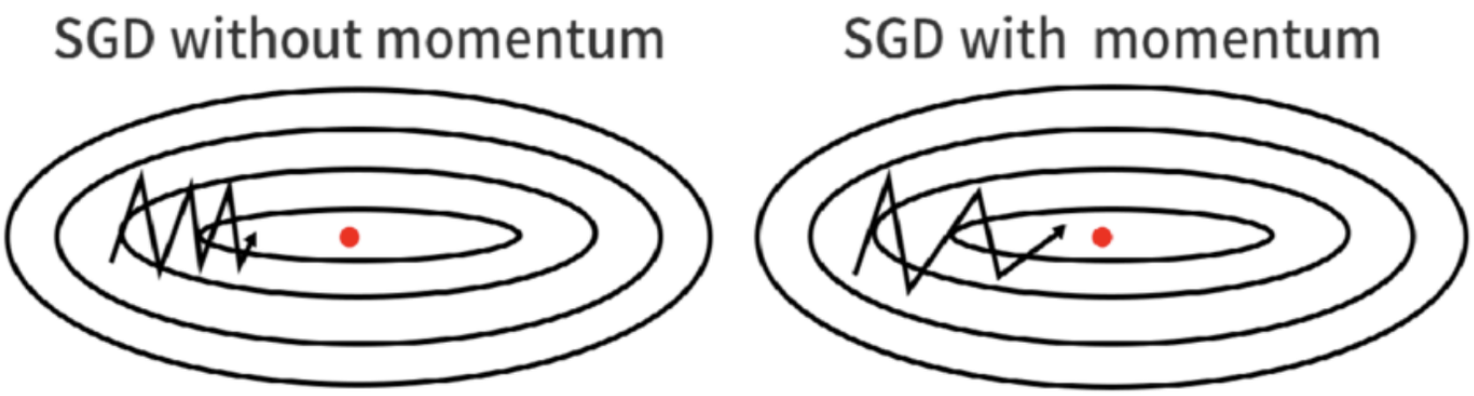


There are 3 main reasons why it does not work :

(1) We end up in local minima and not able to reach global minima At the start, we randomly start at some point and we are going to end up at the local minimum and not able to reach the global minimum.

(2) Saddle Point Problem

(3) Noisy Gradient Problem

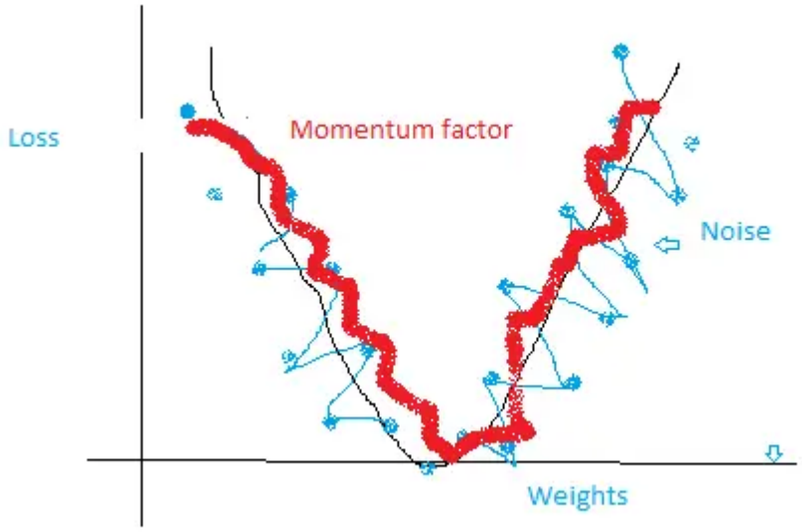


What we have to do here is a SMOOTHING OPERATION. There are some ways to achieve this.

(1) Using Moving Average (Avg of self and the previous value)

(2) Exponential Smoothing

$$Fv_t = B(FV_{t-1}+(1-B)AV_t)$$



$$\gamma V(t) = 1 * (\delta L/\delta w(old)) + \gamma * (\delta L/\delta w(old)) + \gamma^2 * (\delta L/\delta w(old))$$

"If I am repeatedly being asked to move in the same direction then I should probably gain some confidence and start taking bigger steps in that direction. Just as a ball gains momentum while rolling down a slope."

Without momentum:

$$w_{t+1} = w_t - \eta \nabla w_t$$
$$b_{t+1} = b_t - \eta \nabla b_t$$

With momentum:

$$update_t^w = \gamma \cdot update_{t-1}^w + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t^w$$
$$update_t^b = \gamma \cdot update_{t-1}^b + \eta \nabla b_t$$
$$b_{t+1} = b_t - update_t^b$$

Let's observe the expressions for the SGD with Momentum.

$$
\begin{aligned}
update_0 &= 0 \\
update_1 &= \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1 \\
update_2 &= \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2 \\
update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\
&= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \\
update_4 &= \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4 \\
&\vdots \\
update_t &= \gamma \cdot update_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_1 + ... + \eta \nabla w_t
\end{aligned}
$$

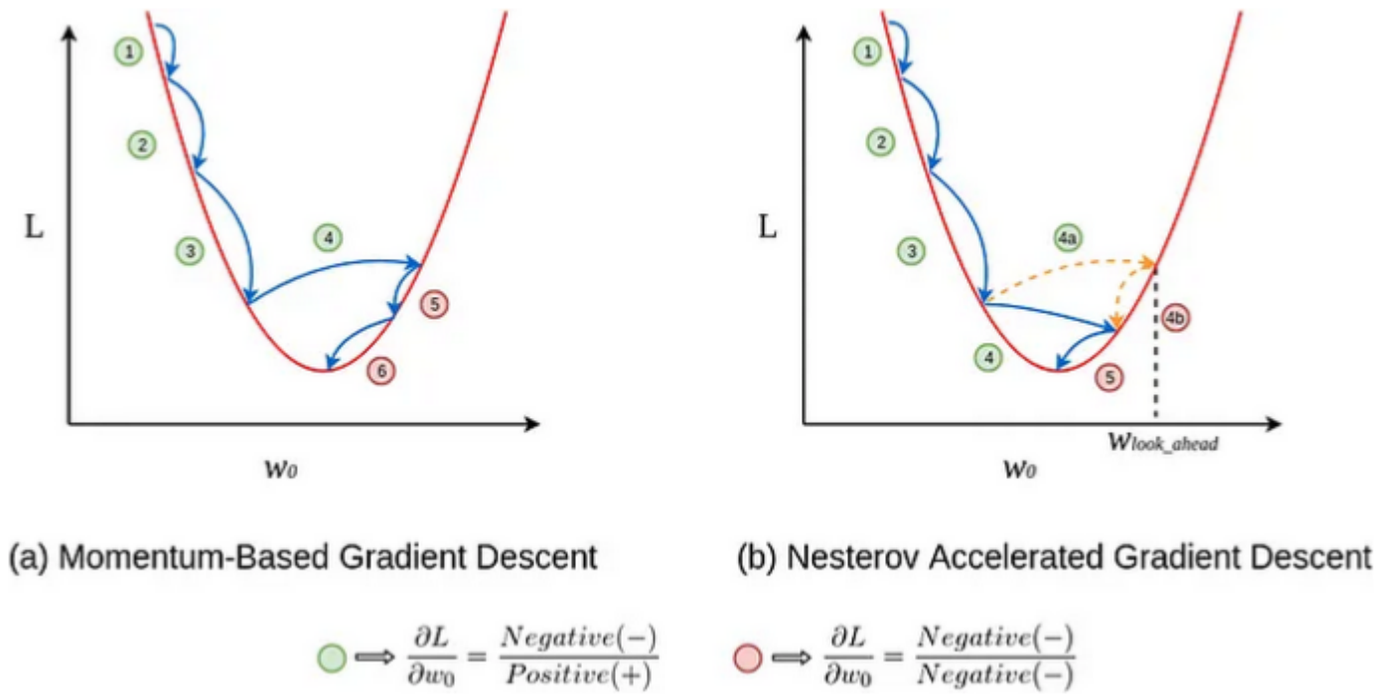
You can see that the current update is proportional to not just the present gradient but also gradients of previous steps, although their contribution reduces every time step by gamma times. And that is how we boost the magnitude of the update at gentle regions.

```
def do_momentum_gradient_descent():
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X,Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        v_w = gamma * prev_v_w + eta*dw
        v_b = gamma * prev_v_b + eta*db
        w = w - v_w
        b = b - v-b
        prev_v_w = v_w
        prev_v_b = v_b
```

SGD with momentum is an optimizer that minimizes the impact of noises in convergence to the optimal weights. Stochastic Gradient Descent momentum just helps to reduce the convergence time. Adding momentum in SGD overcome the major shortcomings of SGD over Batch Gradient Descent without losing its advantage.

WAIT... IS IT OKAY ?

On a critical note, is moving fast always good? Would there be a situation where momentum would cause us to overshoot and run past our goal? YES! To overcome this, we need something more namely Nesterov's Accelerated Gradient Descent.



(a) Momentum-Based Gradient Descent

(b) Nesterov Accelerated Gradient Descent

$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$ $\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$

In NAG's case, every update happens in two steps — first, a partial update, where we get to the *look_ahead* point and then the final update (see the NAG update rule), see figure (b). First 3 updates of NAG are pretty similar to the momentum-based method as both the updates (partial and final) are positive in those cases. But the real difference becomes apparent during update 4. As usual, each update happens in two stages, the partial update (4a) is positive, but the final update (4b) would be negative as the calculated gradient at ***w_lookahead*** would be negative (convince yourself by observing the graph). This negative final update slightly reduces the overall magnitude of the update, still resulting in an overshoot but a smaller one when compared to the vanilla momentum-based gradient descent. NAG heps this way.

AdaGrad

Adaptive **Grad**ient algorithm, or AdaGrad for short, keeps track of the sum of gradient *squared* and uses that to adapt the gradient in different directions.

THE LEARNING RATE

Learning rate is probably the most important aspect of gradient descent and also other optimizers as well. Imagine the cost function as a pit, you will be starting from the top and your objective is to get to the bottom of the pit. You can think of learning rate as the step that you are going to take to reach the bottom(global minima) of the pit. If you choose a large value as learning rate, you would be making drastic changes to the weights and bias values, i.e you would be taking huge jumps to reach the bottom. There is also a huge probability that you will overshoot the global minima(bottom) and end up on the other side of the pit instead of the bottom. If you choose a small value as learning rate, you lose the risk of overshooting the minima but your algorithm will longer time to converge, i.e you take shorter steps but you have to take more number of steps. Hence, you would have to train for a longer period of time.

$$delta = - learning_rate * gradient$$

Also, if the cost function is non-convex, your algorithm might be easily trapped in a local minima and it will be unable to get out and converge to the global minima. There is no generic right value for learning rate. It comes down to experimentation and intuition.

THE RMSPROP OPTIMIZER

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

The problem of AdaGrad, however, is that it is incredibly slow. This is because the *sum of gradient squared* only grows and never shrinks. RMSProp (for **R**oot **M**ean **S**quare **Prop**agation) fixes this issue by adding a decay factor as shown above and below.

*sum_of_gradient_squared = previous_sum_of_gradient_squared * **decay_rate**+ gradient² * (1- **decay_rate**)*

ADAM

Adam gets the speed from momentum and the ability to adapt gradients in different directions from RMSProp. The combination of the two makes it powerful.

*sum_of_gradient = previous_sum_of_gradient * beta1 + gradient * (1 - beta1)* [Momentum]

*sum_of_gradient_squared = previous_sum_of_gradient_squared * beta2 + gradient² * (1- beta2)* [RMSProp]

*delta = -learning_rate * sum_of_gradient / sqrt(sum_of_gradient_squared)*

THE DROPOUT REGULARIZATION

The best way to reduce overfitting or the best way to regularise a fixed-size model is to get the average predictions from all possible settings of the parameters and aggregate the final output. But, this becomes too computationally expensive and isn't feasible for a real-time inference. In the overfitting problem, the model learns the statistical noise. To be precise, the main motive of training is to decrease the loss function, given all the units (neurons). So in overfitting, a unit may change in a way that fixes up the mistakes of the other units. This leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalise on the unseen dataset.

Now, if we use dropout, it prevents these units to fix up the mistake of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable. So by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach. This ensures that the model is getting generalised and hence reducing the overfitting problem. You can read more about it in the following book :

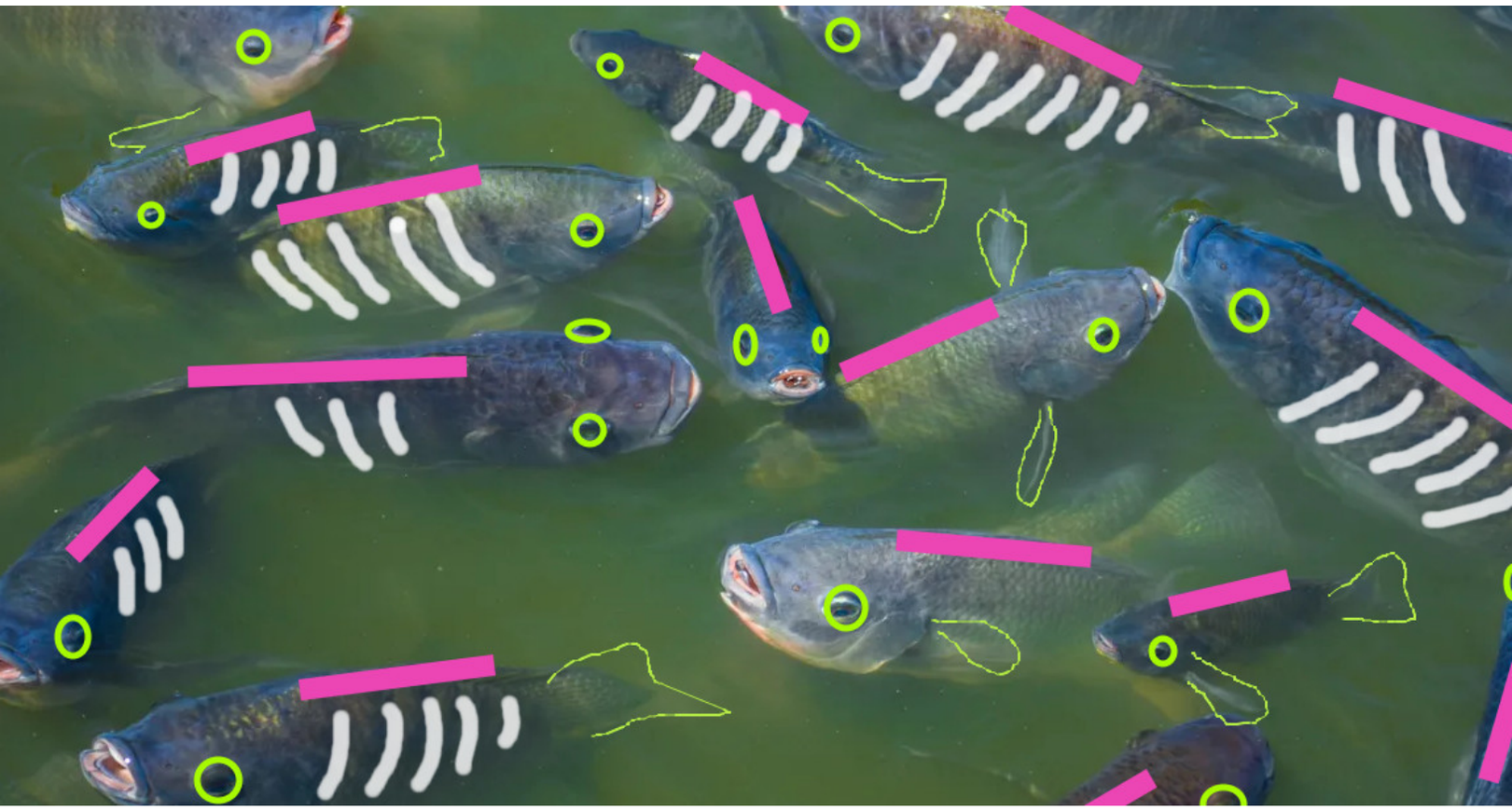
arihara-sudhan.github.io/books.html

Dropout is training a large ensemble of models (that share parameters).

CNN : Convolutional Neural Network

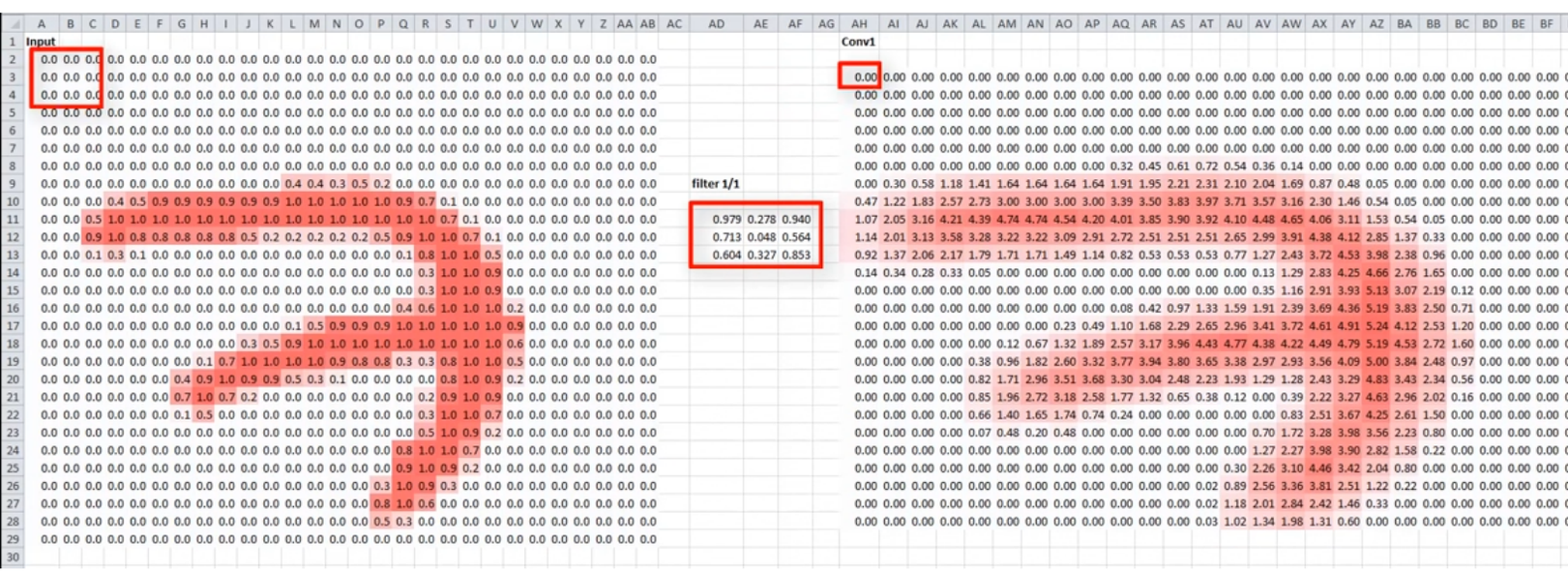
Why is it called Convolutional Neural Network ? Sometimes called ConvNets or CNNs. Convolutional neural networks get their name from a mathematical operation called convolution. This is a specialized kind of linear operation, and CNNs use this mathematical operation instead of matrix multiplication in at least one of the layers. This is what separates ConvNets from other neural networks in deep learning. The ConvNets can also be used for the process of Speech Recognition as well as Natural Language Processing. Facebook's speech recognition technology is based on convolutional neural network.

In a conventional neural network (Not Convolutional – MLP), the hidden layers are where the transformation happens. Here, in the convolutional neural network, hidden layers are the convolutional layers. We have to specify how many filters a convolutional layer has to have. Each filter is capable of recognizing a particular pattern. What is a pattern then ? Patterns are simple shapes which collectively make up a complete shape. For an instance, we have provided the shoal of Tilapia fish. We can see many fish swimming. We can also observe some shapes and textures among them. One filter can recognize the lines. One can recognize the circles. One can recognize the stripes.

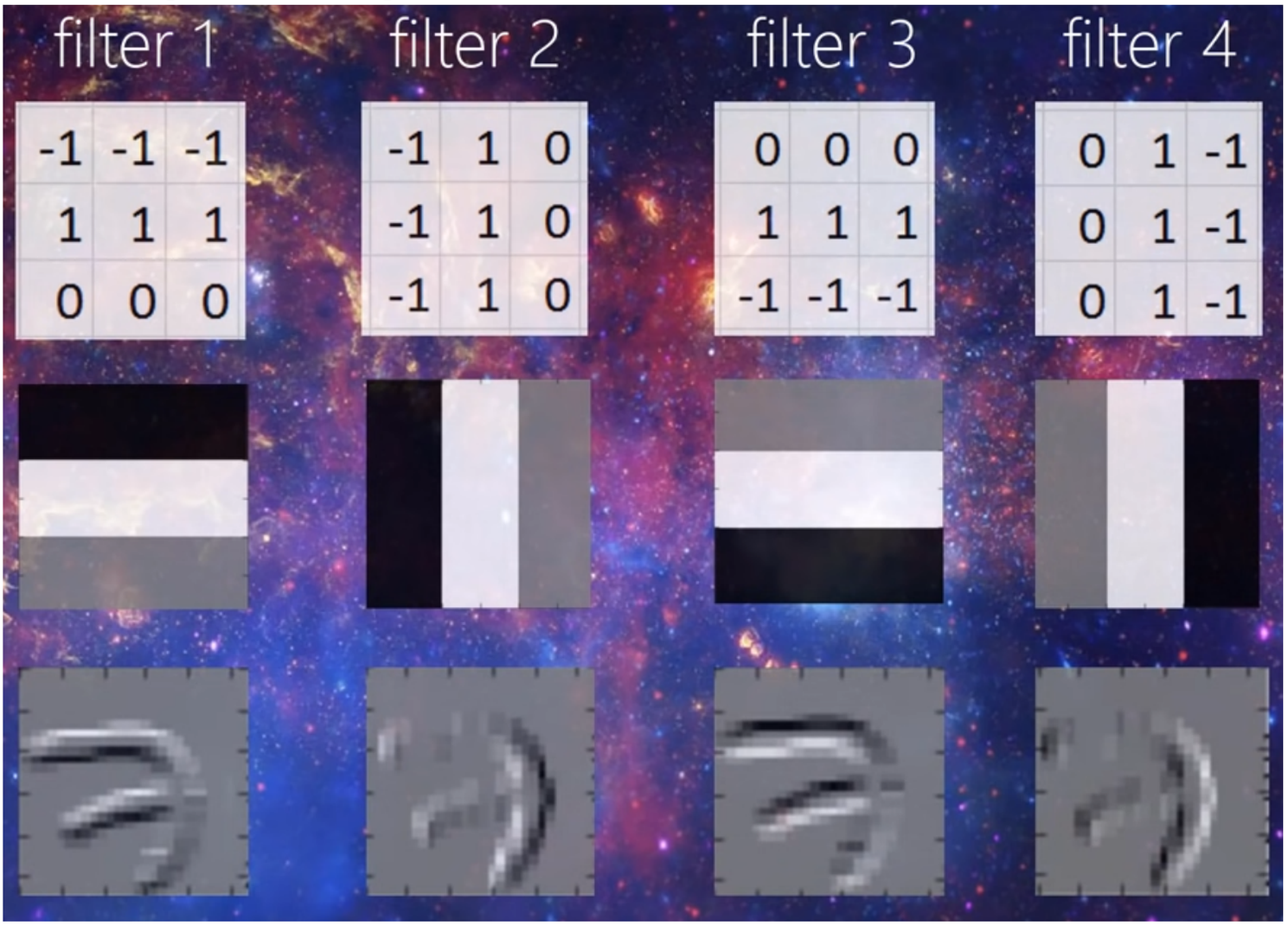


They collectively recognize what the picture is all about. The more deeper the network goes, the more sophisticated these layers are!

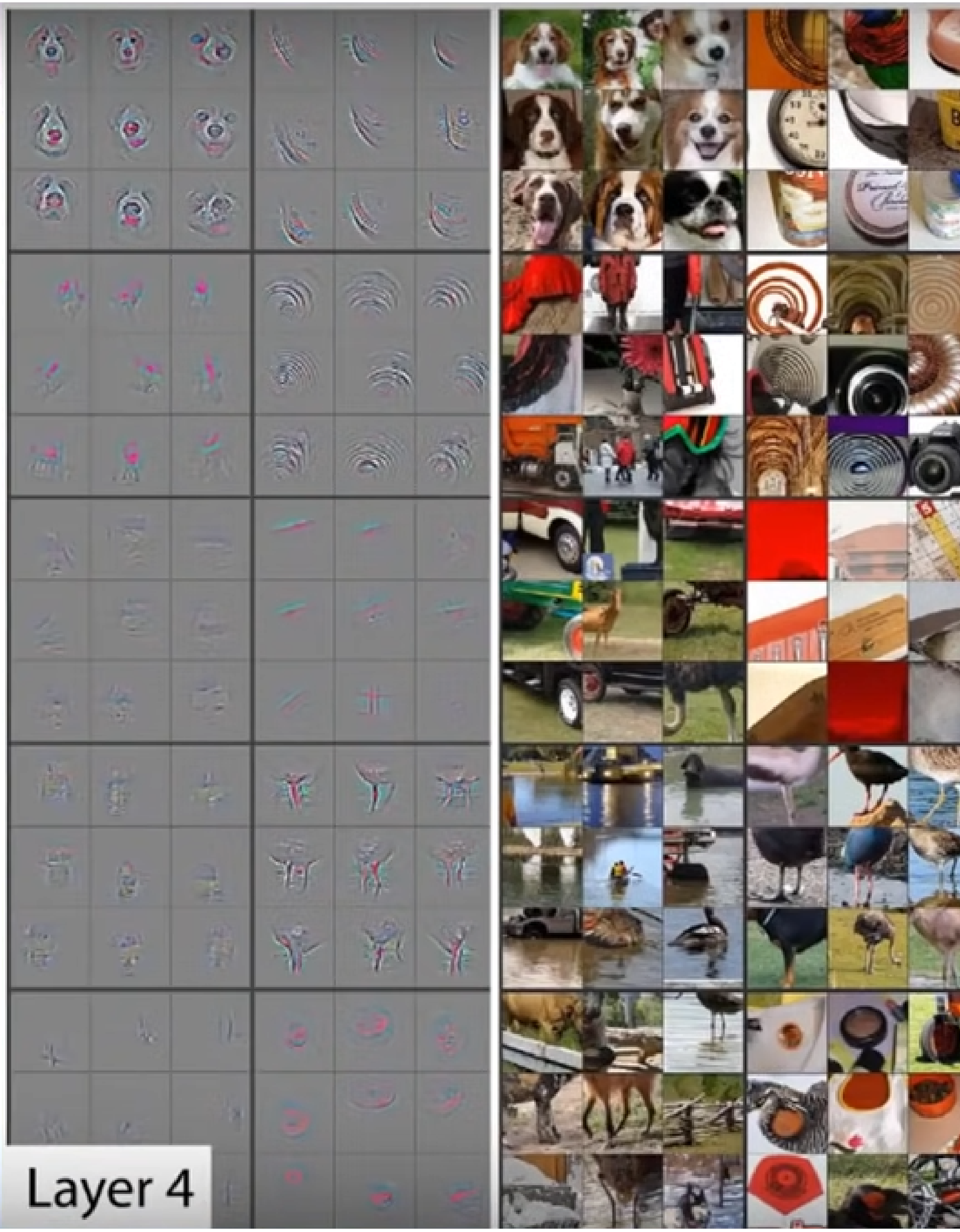
OKAY! Why do we call these layers convolutional layers ? Because, we convolve in this layer. Convolving means rolling together. Applying convolution to an image, we can decrease the image size as well as bringing all the information in the field together into a single pixel. The final output of the convolutional layer is a vector. In the process of applying convolution to an image, we first create a small randomly initialized matrix called as FILTER. If the FILTER has a dimension of 3x3, it will slide through the image by 3x3 dimensions. This sliding will convolve across each 3x3 dimension in the image. The same steps are applied to the next layer in the ConvNet. PIXELS ---------- ACTIVATION MAPS

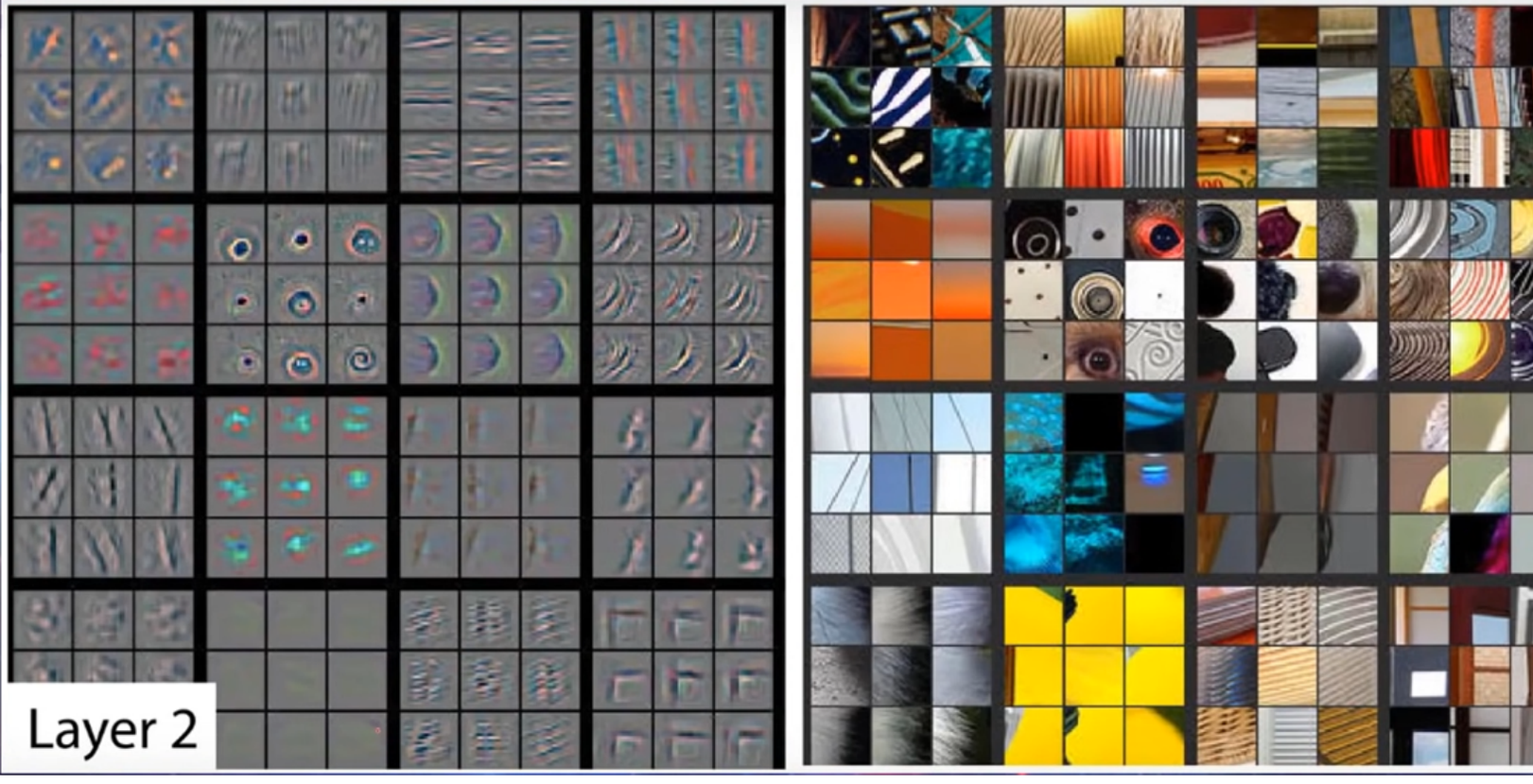


Here in the example shown above, seven is to be detected. The filter is applied to the first 3x3 dimension and the resulting dot product is stored in the result[0][0]. It is made until the sliding is completed.

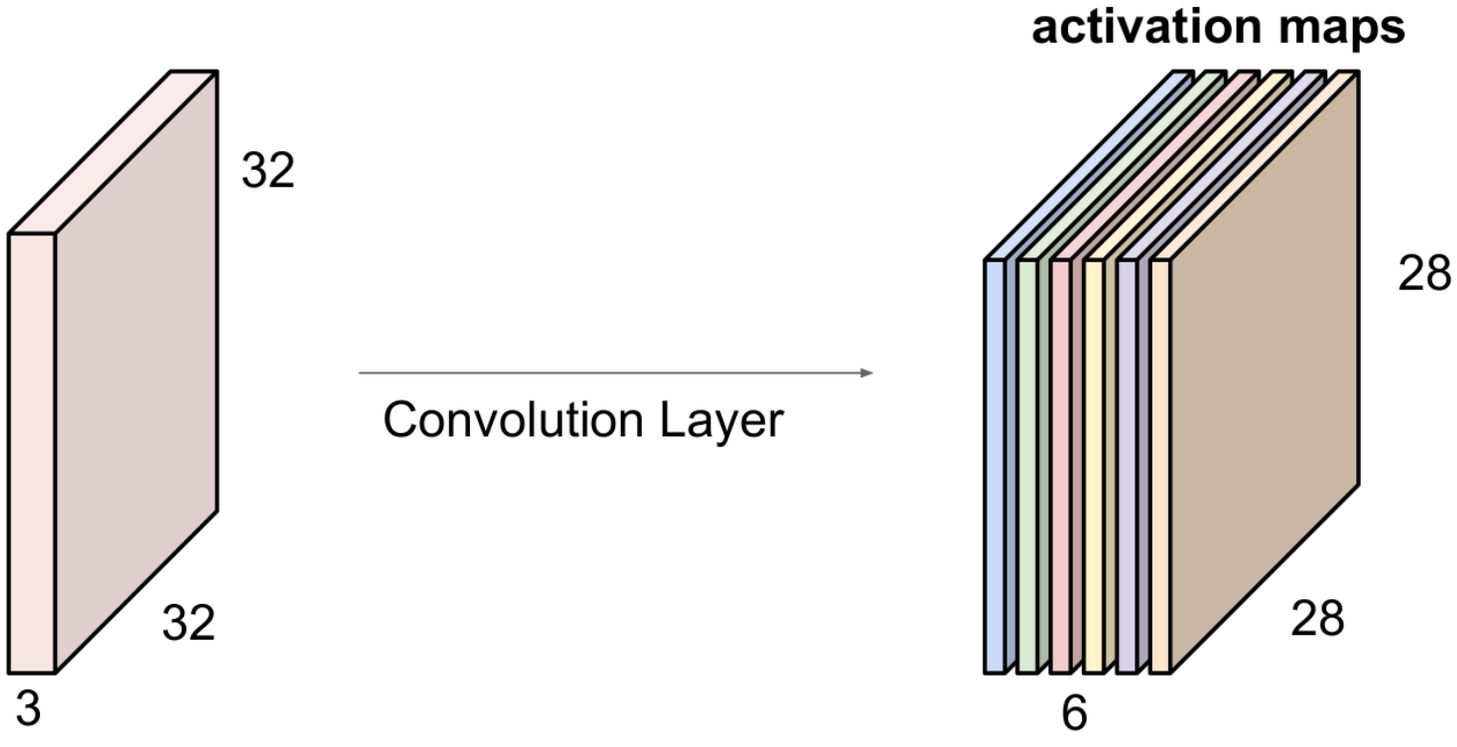


Filters actually does something. In the example given above, First one detects the top horizontal edge. The second one detects the left horizontal edge. Third one detects the bottom horizontal edge. The last one detects the right vertical edge. Using the appropriate filters, we can detect the apt shape in an image. Most complex filters are there in the deep.


Layer 4

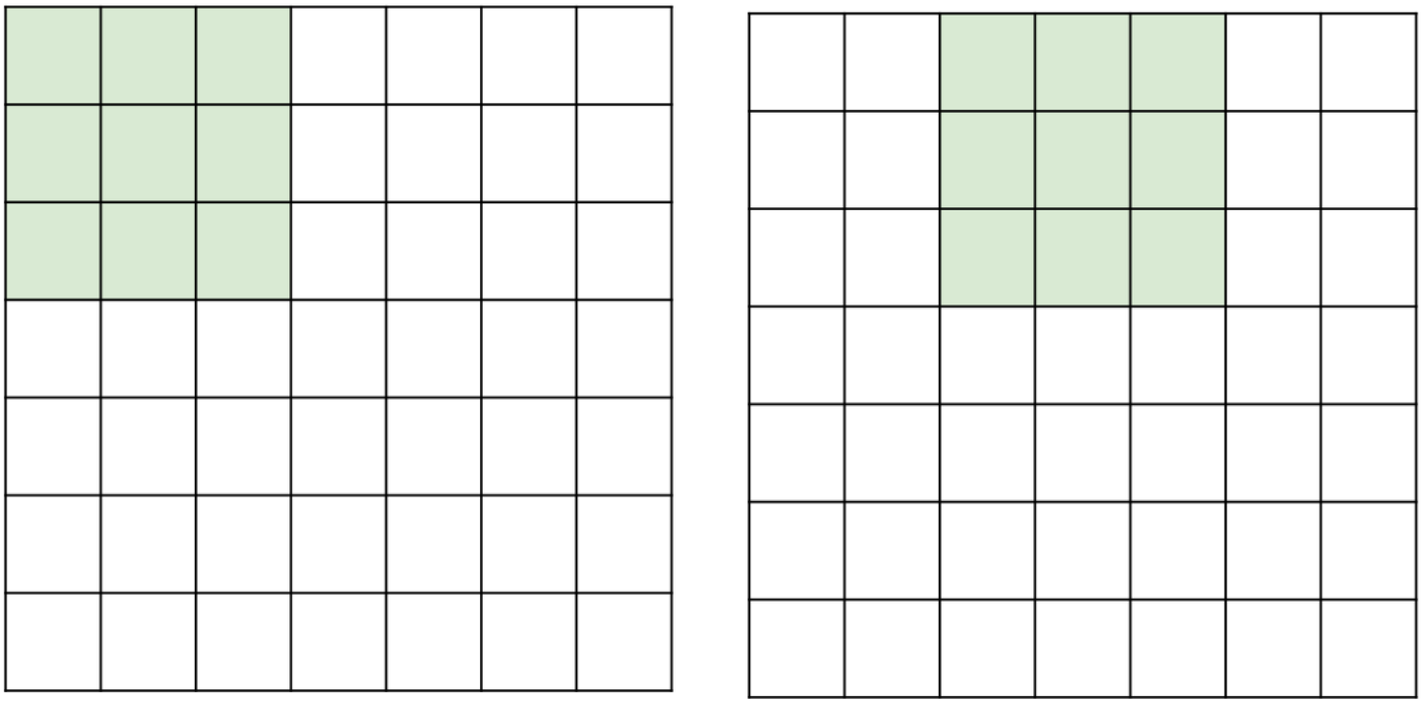


If we had six 5x5 filters, we'll get 6 separate activation maps :

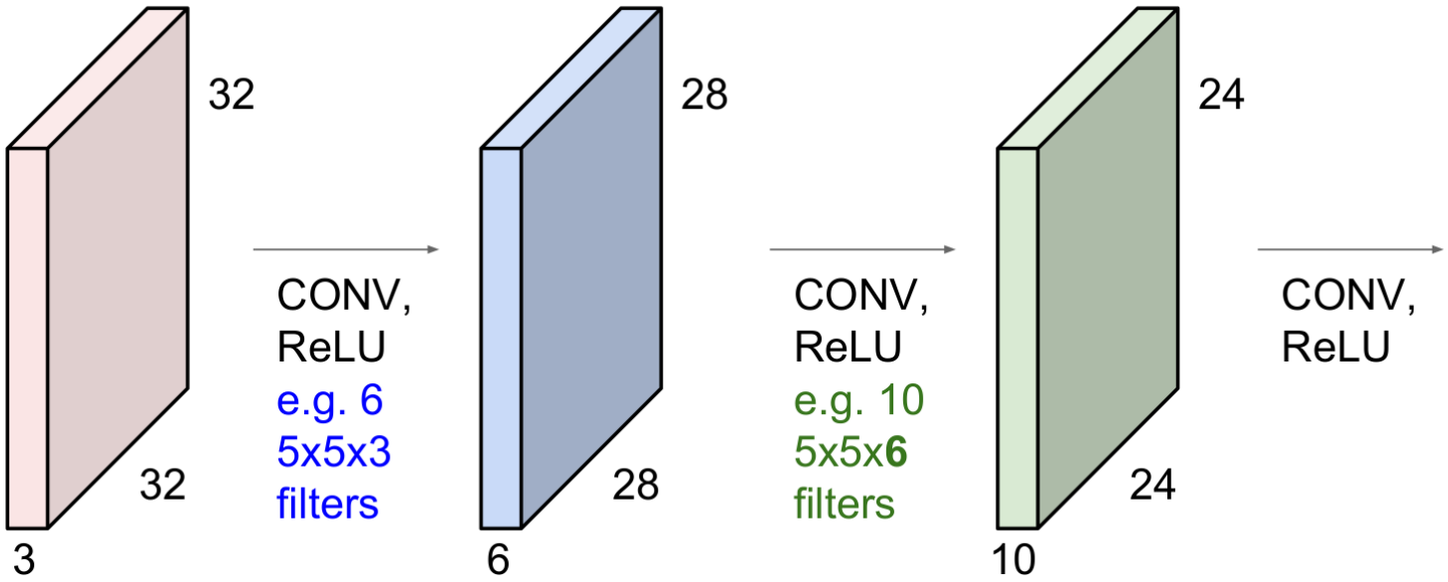


Just after the convolution is performed, we apply an activation function such as ReLU.

Stride determines the step we take towards the right side from the left most of an image array. If we have a stride of 2, our moving will be like,



The volume shouldn't shrink too fast. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). It won't work well. Padding with zero is a helpful remedy for this problem.



MAX-POOLING

Max-Pooling is a sort of operation followed after the series of convolutions. It is for reducing the dimension of the output from the convolutional layer by using a kernel which moves with a stride and captures the maximum value from the activation map.

Conv1

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.32	0.45	0.61	0.72	0.54	0.36	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.30	0.58	1.18	1.41	1.64	1.64	1.64	1.64	1.91	1.95	2.21	2.31	2.10	2.04	1.69	0.87	0.48	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.47	1.22	1.83	2.57	2.73	3.00	3.00	3.00	3.00	3.39	3.50	3.83	3.97	3.71	3.57	3.16	2.30	1.46	0.54	0.05	0.00	0.00	0.00	0.00	0.00	0.00
1.07	2.05	3.16	4.21	4.39	4.74	4.74	4.54	4.20	4.01	3.85	3.90	3.92	4.10	4.48	4.65	4.06	3.11	1.53	0.54	0.05	0.00	0.00	0.00	0.00	0.00
1.14	2.01	3.13	3.58	3.28	3.22	3.22	3.09	2.91	2.72	2.51	2.51	2.51	2.65	2.99	3.91	4.38	4.12	2.85	1.37	0.33	0.00	0.00	0.00	0.00	0.00
0.92	1.37	2.06	2.17	1.79	1.71	1.71	1.49	1.14	0.82	0.53	0.53	0.53	0.77	1.27	2.43	3.72	4.53	3.98	2.38	0.96	0.00	0.00	0.00	0.00	0.00
0.14	0.34	0.28	0.33	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	1.29	2.83	4.25	4.66	2.76	1.65	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	1.16	2.91	3.93	5.13	3.07	2.19	0.12	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.42	0.97	1.33	1.59	1.91	2.39	3.69	4.36	5.19	3.83	2.50	0.71	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.49	1.10	1.68	2.29	2.65	2.96	3.41	3.72	4.61	4.91	5.24	4.12	2.53	1.20	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.12	0.67	1.32	1.89	2.57	3.17	3.96	4.43	4.77	4.38	4.22	4.49	4.79	5.19	4.53	2.72	1.60	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.38	0.96	1.82	2.60	3.32	3.77	3.94	3.80	3.65	3.38	2.97	2.93	3.56	4.09	5.00	3.84	2.48	0.97	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.82	1.71	2.96	3.51	3.68	3.30	3.04	2.48	2.23	1.93	1.29	1.28	2.43	3.29	4.83	3.43	2.34	0.56	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.85	1.96	2.72	3.18	2.58	1.77	1.32	0.65	0.38	0.12	0.00	0.39	2.22	3.27	4.63	2.96	2.02	0.16	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.66	1.40	1.65	1.74	0.74	0.24	0.00	0.00	0.00	0.00	0.00	0.83	2.51	3.67	4.25	2.61	1.50	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.07	0.48	0.20	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.70	1.72	3.28	3.98	3.56	2.23	0.80	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.27	2.27	3.98	3.90	2.82	1.58	0.22	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30	2.26	3.10	4.46	3.42	2.04	0.80	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.89	2.56	3.36	3.81	2.51	1.22	0.22	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.18	2.01	2.84	2.42	1.46	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	1.02	1.34	1.98	1.31	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Max Pool

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.32	0.61	0.72	0.36	0.00	0.00	0.00	0.00	0.00
1.22	2.57	3.00	3.00	3.39	3.83	3.97	3.57	2.30	0.54	0.00	0.00	0.00
2.05	4.21	4.74	4.74	4.20	3.90	4.10	4.65	4.38	2.85	0.33	0.00	0.00
1.37	2.17	1.79	1.71	1.14	0.53	0.77	2.43	4.53	4.66	1.65	0.00	0.00
0.00	0.00	0.00	0.00	0.08	0.97	1.59	2.39	4.36	5.19	2.50	0.00	0.00
0.00	0.00	0.12	1.32	2.57	3.96	4.77	4.38	4.91	5.24	2.72	0.00	0.00
0.00	0.00	1.71	3.51	3.77	3.94	3.65	2.97	4.09	5.00	2.48	0.00	0.00
0.00	0.00	1.96	3.18	2.58	1.32	0.38	0.83	3.67	4.63	2.02	0.00	0.00
0.00	0.00	0.48	0.48	0.00	0.00	0.00	2.27	3.98	3.56	0.80	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.89	3.36	4.46	2.04	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	1.18	2.84	2.42	0.33	0.00	0.00	0.00

We must be thinking that Is down-scaling the images is the only use of it. There are many advantages of using Max-Pooling over other Pooling operations.

Max Pooling adds translation invariance.

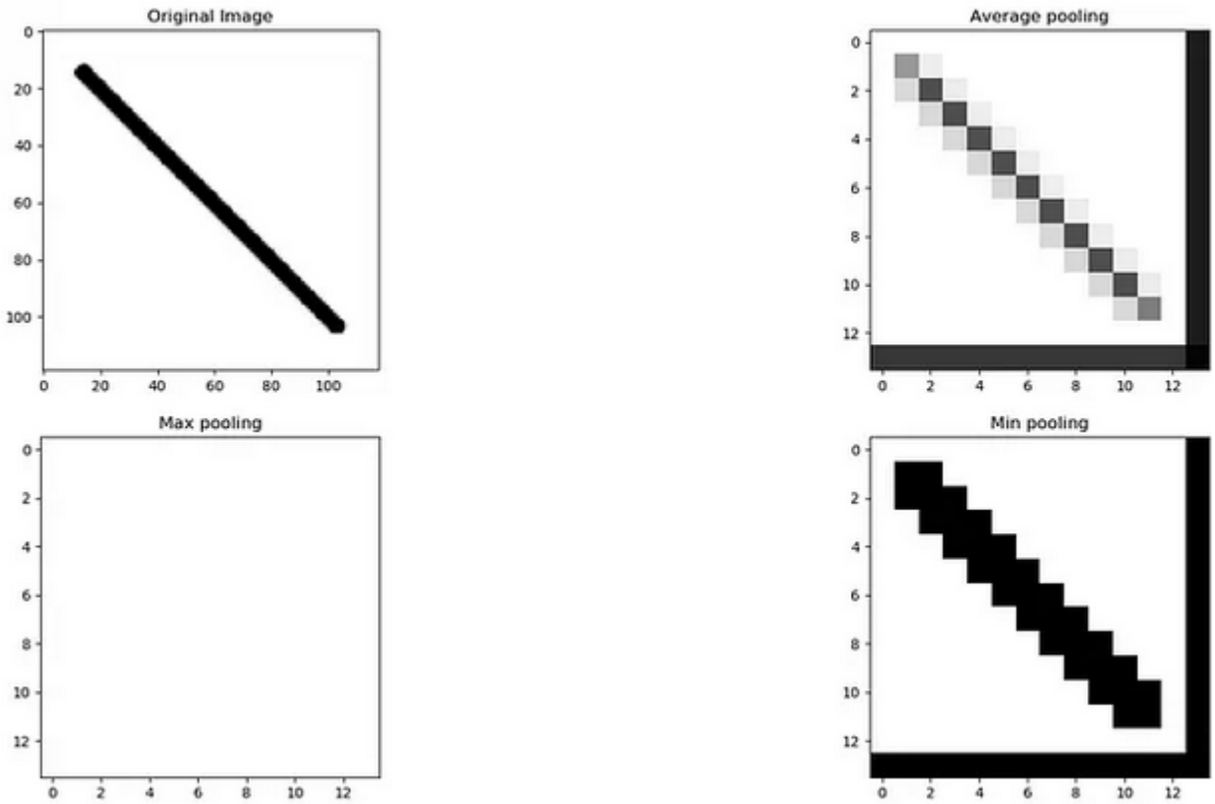
1. Shift Invariance(Invariance in Position)
2. Rotational Invariance(Invariance in Rotation)
3. Scale Invariance(Invariance in Scale(small or big))

The three types of pooling operations are:

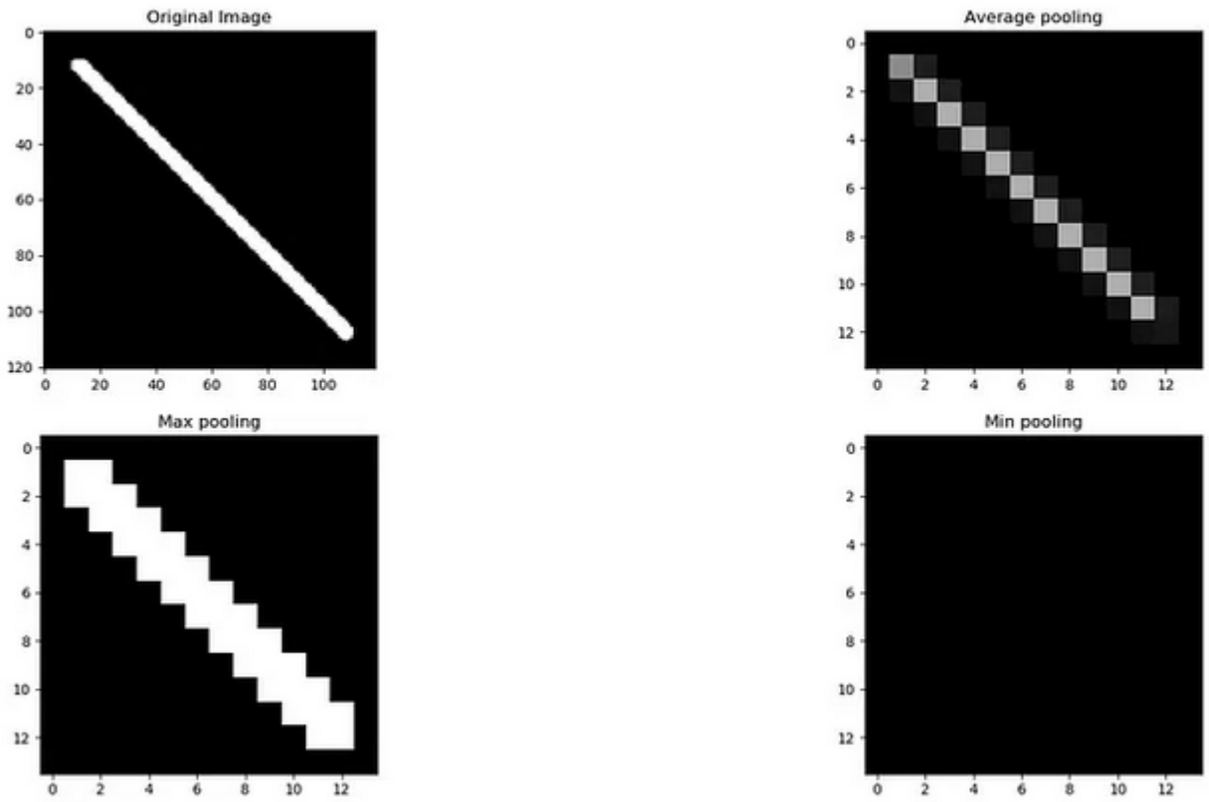
1. Max pooling : The maximum pixel value of the batch is selected.
2. Min pooling : The minimum pixel value of the batch is selected
3. Average pooling : Theaverage value of all the pixels is selected.

We cannot say that a particular pooling method is better over other generally. The choice of pooling operation is made based on the data at hand. Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. For example: in MNIST dataset, the digits are represented in white color and the background is black. So, max pooling is used. Similarly, min pooling is used in the other way round. Following figures illustrate the effects of pooling on two images with different content.

MIN-POOLING :

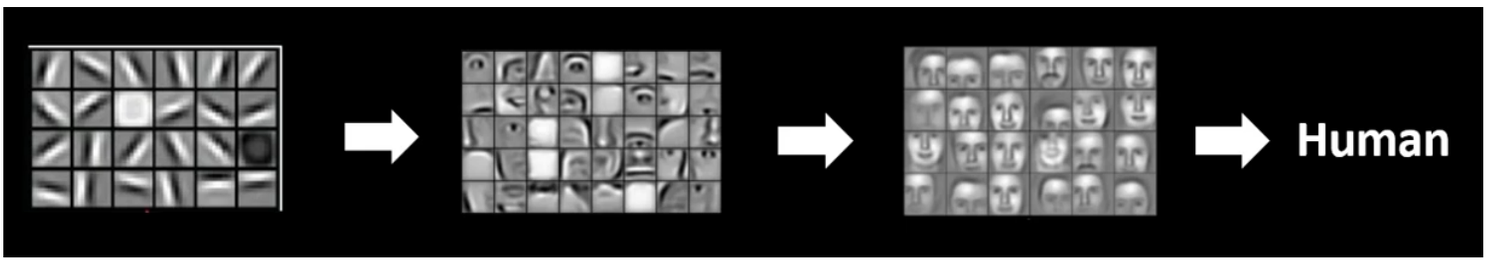


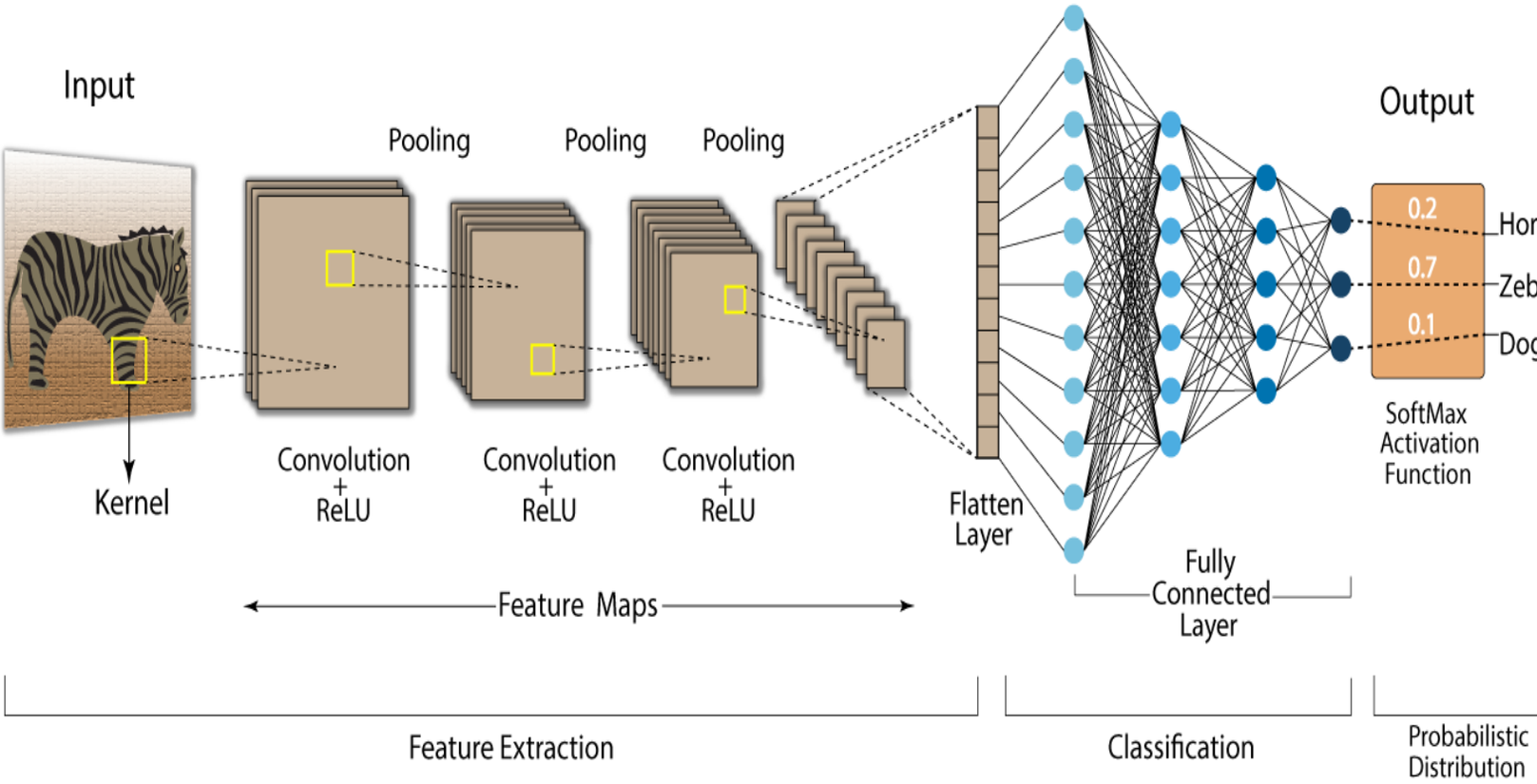
MAX-POOLING :



THE FULLY CONNECTED LAYER

Till now, we have done all the appropriate actions to extract the features from a given image. Now, how to classify the image to a particular class on the basis of the feature that we have extracted till now ? Here we go for the Fully Connected Layer in the Convolutional Neural Network.

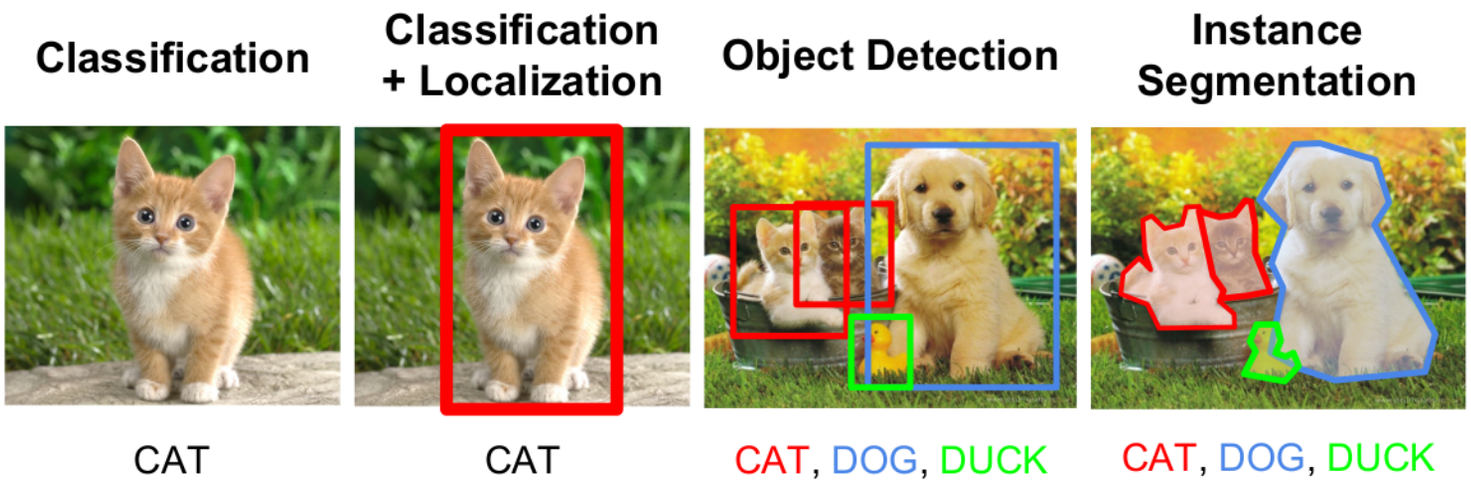




The images are flattened into a one dimensional array which will be used as an input to the fully connected layer.

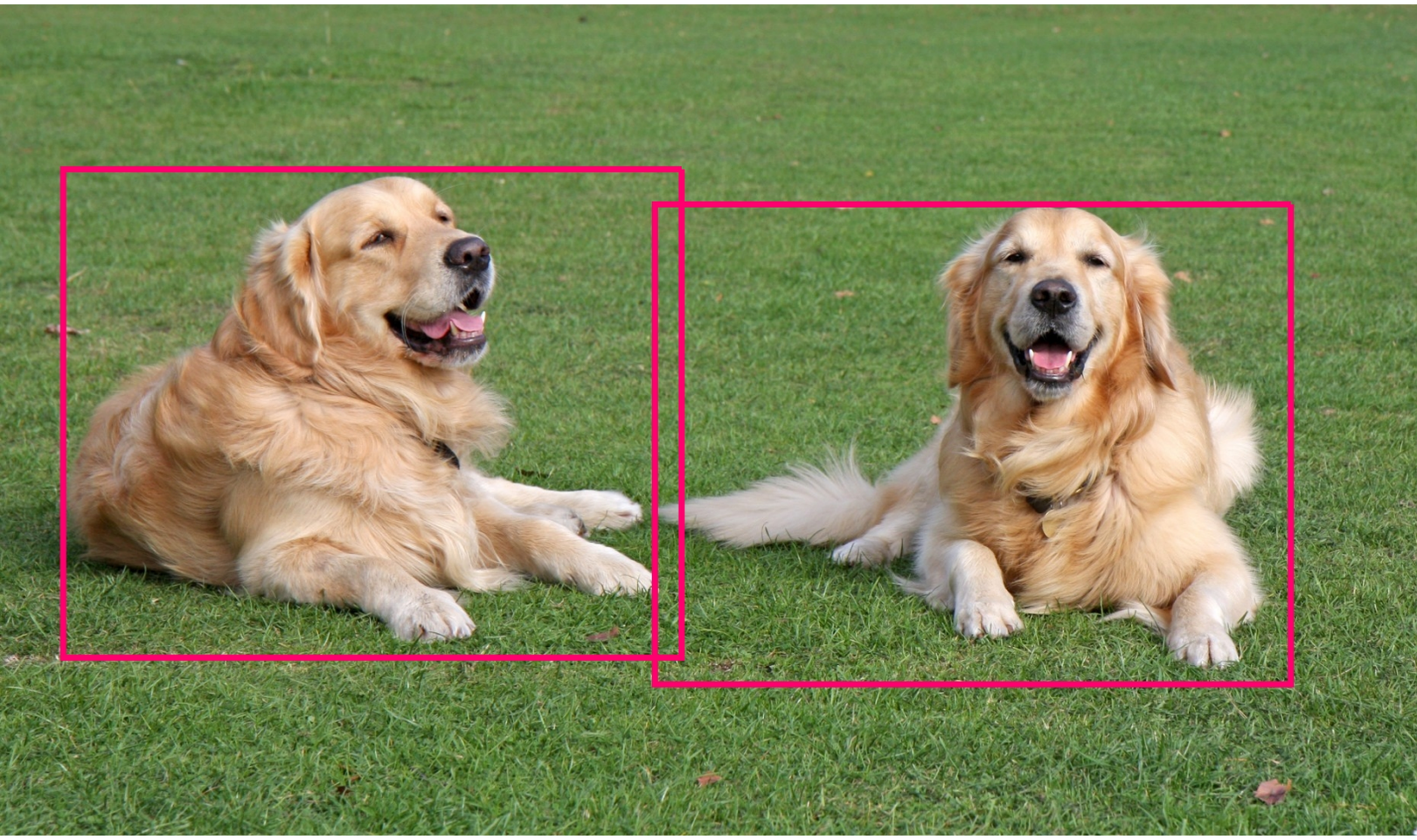
LOCALIZATION & DETECTION

Localization is the process of finding out where the intended object is located in a given image. Eventually, a bounding box will be drawn. Meanwhile, Detection is when we have variable or multiple number of objects.



A Simple strategy is that we can consider the process of localization as a regression problem where we find numeric values for drawing the bounding box. If we have only one object or we know the number of objects, it is actually trivial. We can use one convolutional neural network and train it not only to classify the image but also to output 4 coordinates for the bounding box.

In that way, we treat the localization as a simple regression problem. If the image has 2 objects, we need 8 coordinates. If it has 4 objects, we want 16. So how we build such a model?

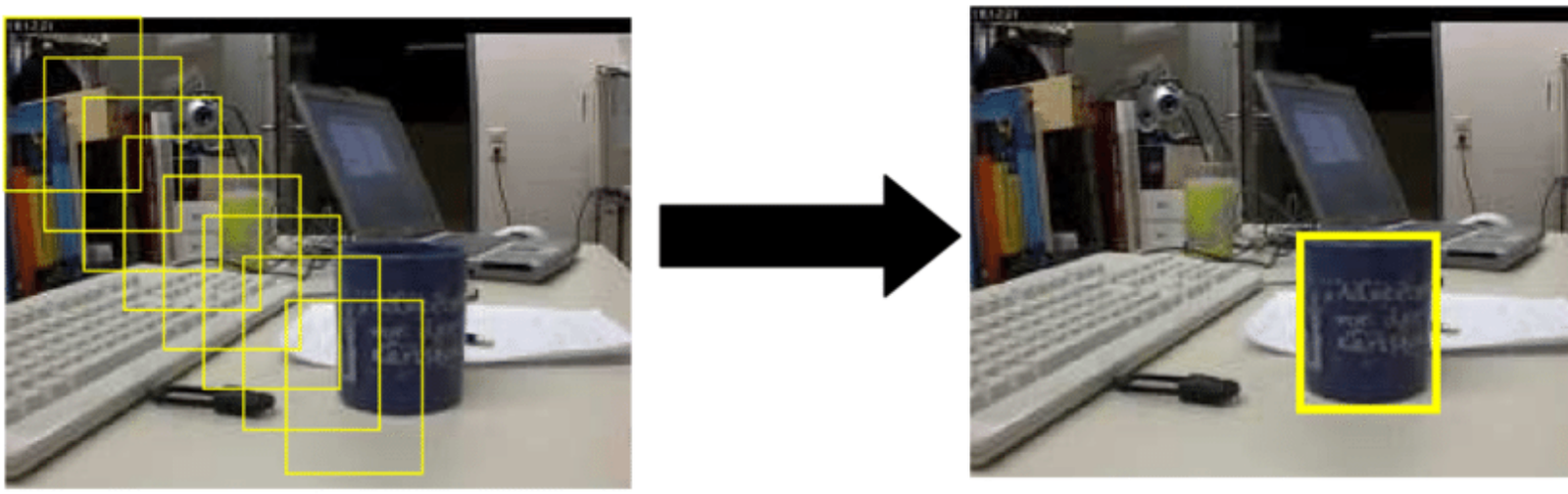


OBJECT LOCALIZATION

At the heart of all object detection algorithms is an object recognition algorithm. Suppose we trained an object recognition model which identifies dogs in image patches. This model will tell whether an image has a dog in it or not. It does not tell where the object is located.

To localize the object, we have to select sub-regions (patches) of the image and then apply the object recognition algorithm to these image patches. The location of the objects is given by the location of the image patches where the class probability returned by the object recognition algorithm is high.

SLIDING WINDOW PROBLEM



In the sliding window approach, we slide a box or window over an image to select a patch and classify each image patch covered by the window using the object recognition model. It is an exhaustive search for objects over the entire image.

REGION PROPOSAL ALGORITHMS

These region proposals can be noisy, overlapping and may not contain the object perfectly but among these region proposals, there will be a proposal which will be very close to the actual object in the image. We can then classify these proposals using the object recognition model. The region proposals with the high probability scores are locations of the object. Region proposal algorithms identify prospective objects in an image using segmentation. In segmentation, we group adjacent regions which are similar to each other based on some criteria such as color, texture etc. Unlike the sliding window approach where we are looking for the object at all pixel locations and at all scales, region proposal algorithm work by grouping pixels into a smaller number of segments. So the final number of proposals generated are many times less than sliding window approach. This reduces the number of image patches we have to classify. These generated region proposals are of different scales and aspect ratios.

An important property of a region proposal method is to have a very **high recall**. This is just a fancy way of saying that the regions that contain the objects we are looking have to be in our list of region proposals.

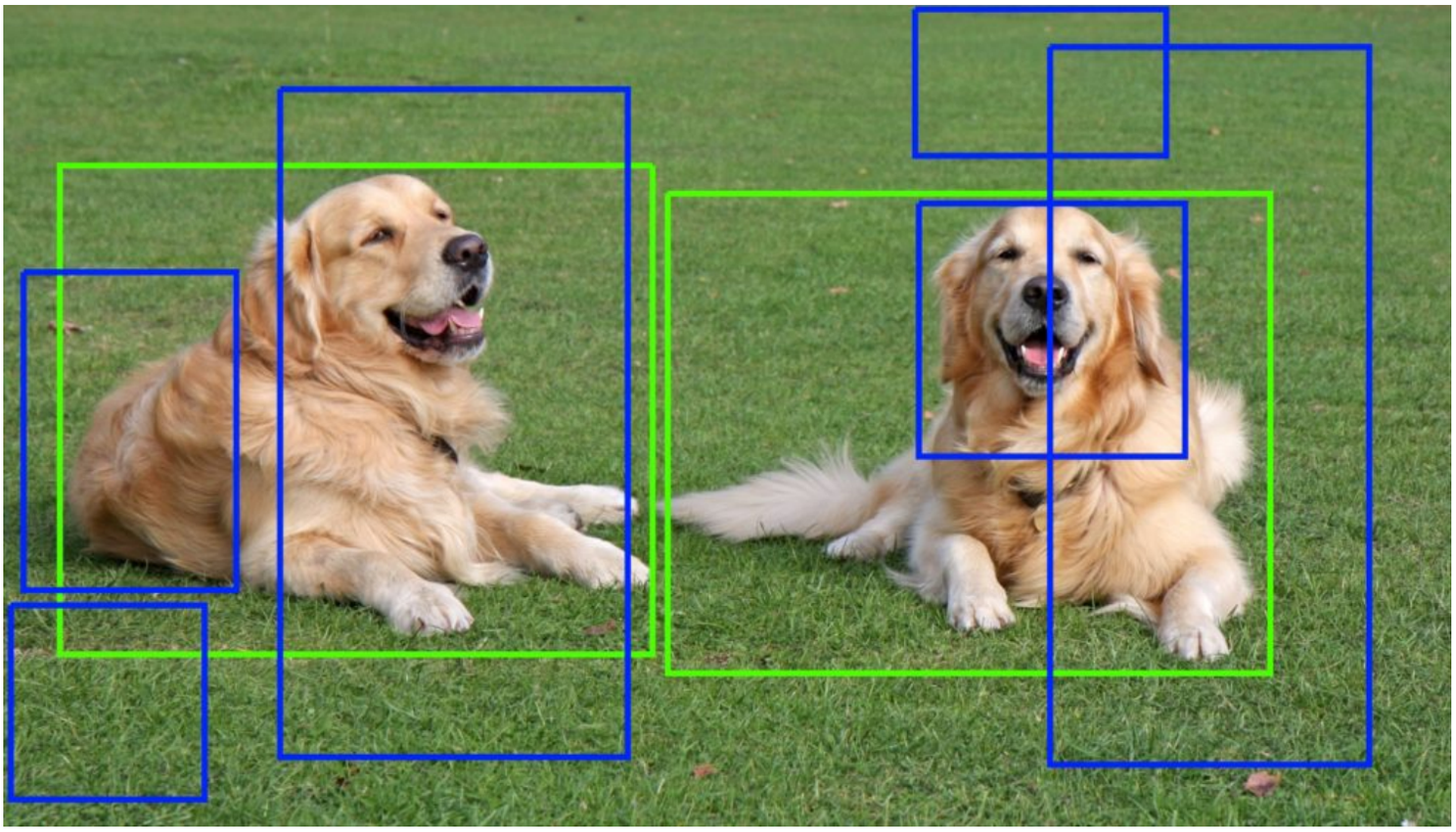
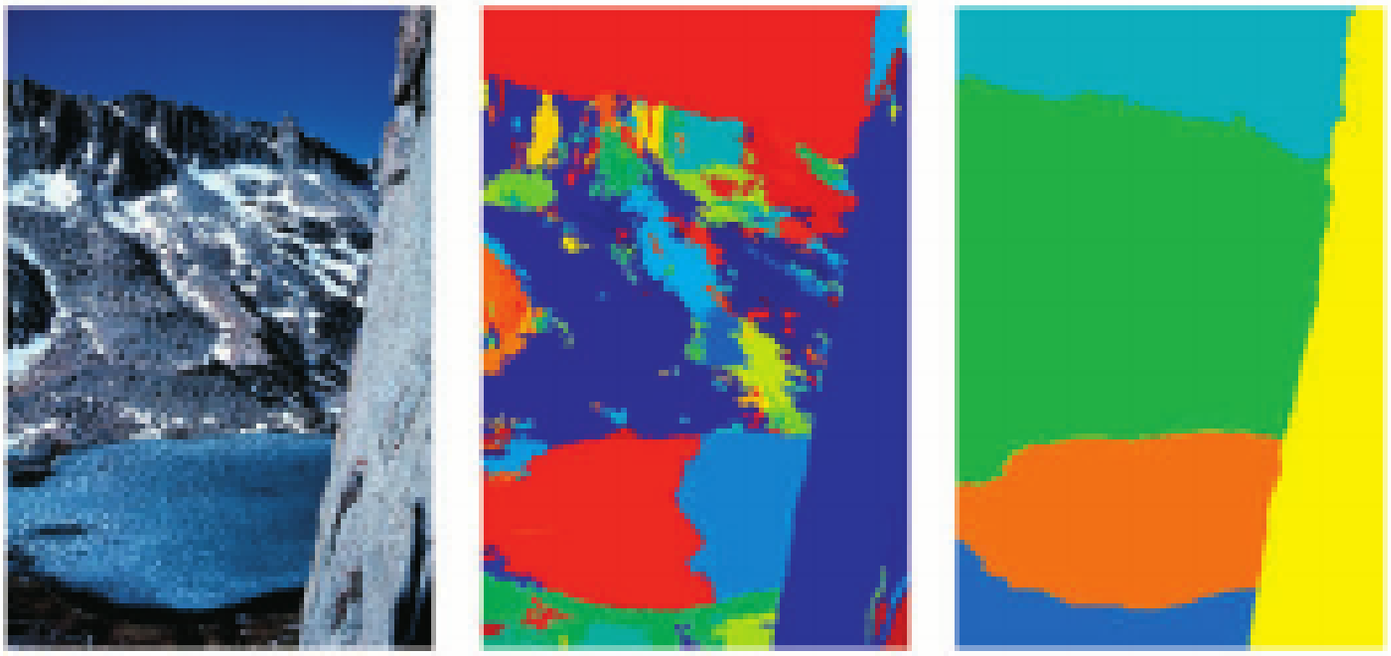


Figure 1: Blue Boxes: False Positives; Green Boxes: True Positives

THE SELECTIVE SEARCH

Selective search uses over-segments from Felzenszwalb and Huttenlocher's method as an initial seed. An over-segmented image looks like the below :



Selective Search algorithm takes these over-segments as initial input and performs the following steps

1. Add all bounding boxes corresponding to segmented parts to the list of regional proposals
2. Group adjacent segments based on similarity
3. Go to step 1

At each iteration, larger segments are formed and added to the list of region proposals. Hence we create region proposals from smaller segments to larger segments in a bottom-up approach. Selective Search uses 4 similarity measures based on color, texture, size and shape compatibility.

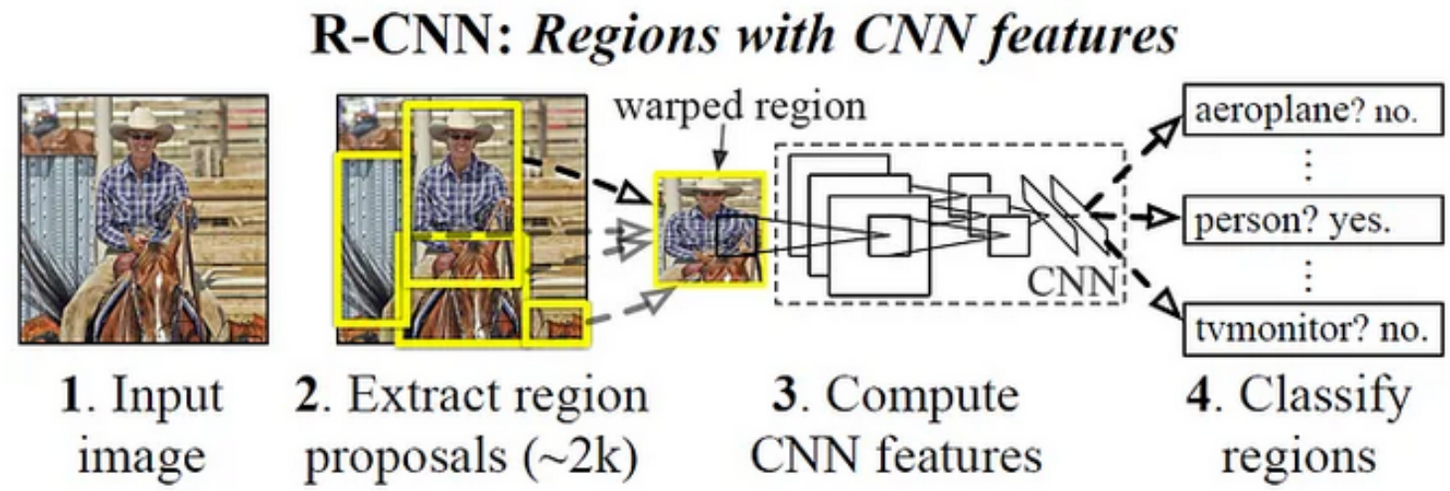
@ COLOR @ TEXTURE @ SHAPE @ SIZE

FINAL Similarity will be,

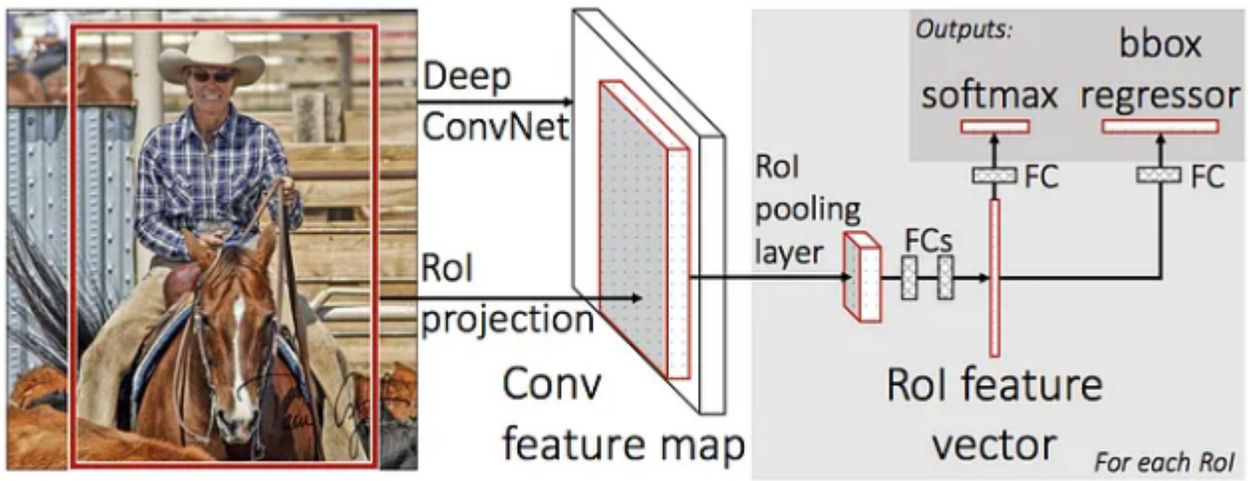
$$s(r_i, r_j) = a_1 s_{color}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j)$$

R-CNN

Instead of trying to classify a huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated using the selective search algorithm which is written below. Although the model produces good results, it suffers from a main issue. It is quite slow and computationally expensive. Imagine that in an average case, we produce 2000 regions, which we need to store in disk, and we forward each one of them into the CNN for multiple passes until it is trained.



FAST R-CNN

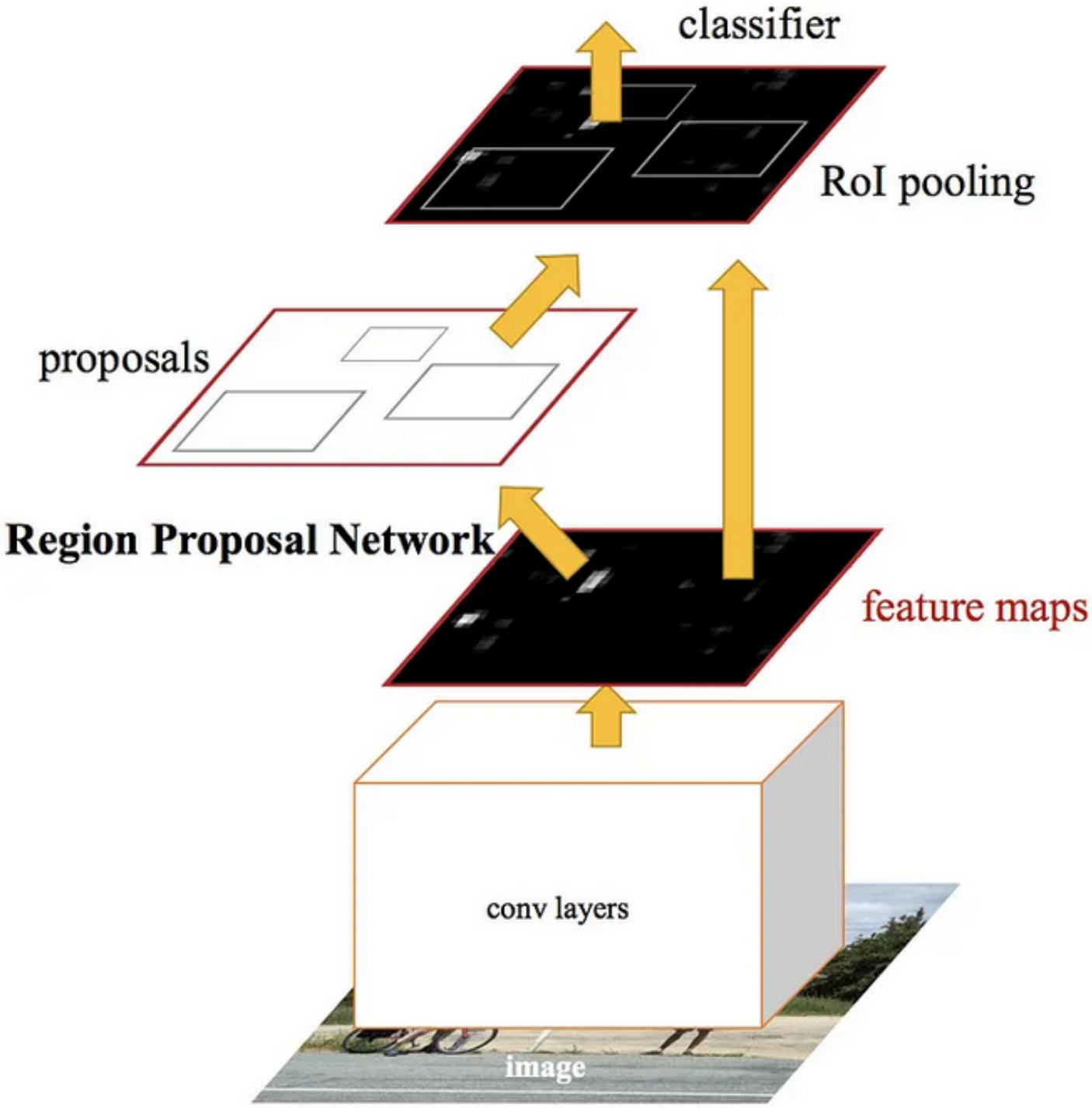


Fast R-CNN

Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

FASTER R-CNN

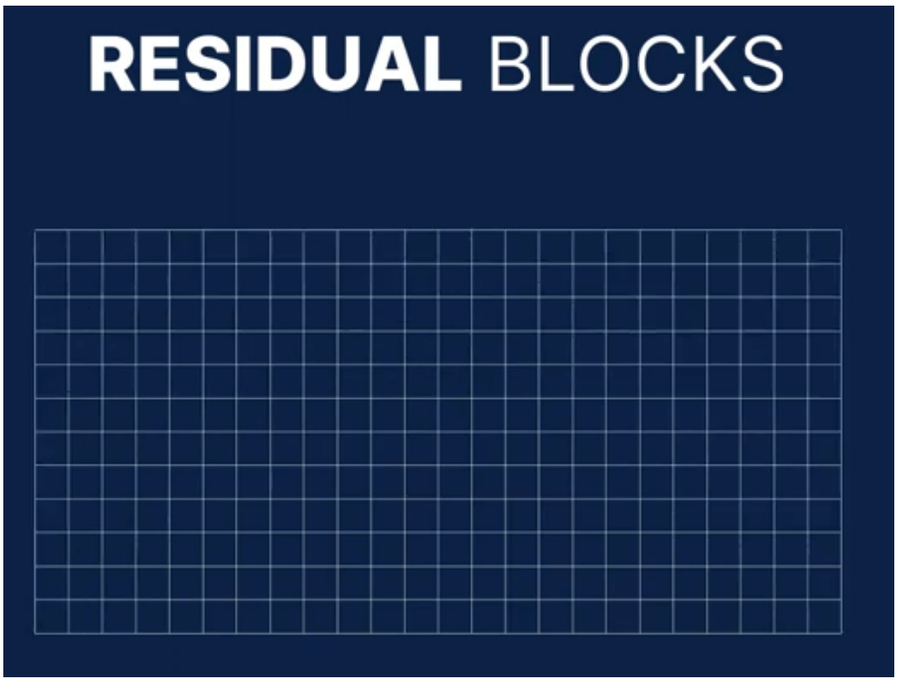


Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

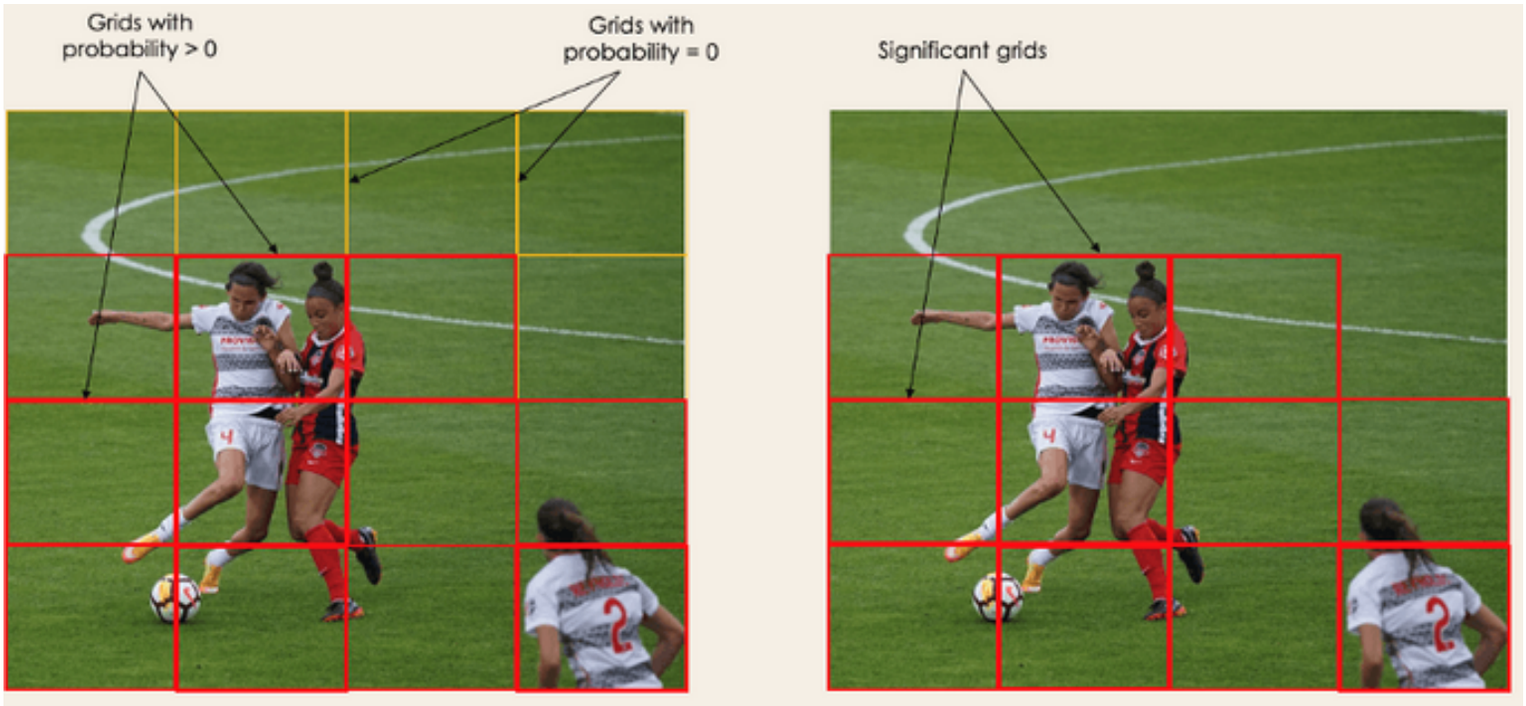
THE SO-CALLED YOLO ALGORITHM

YOLO uses a single algorithm to detect an object. It stands for YOU ONLY LOOK ONCE (As only a single forward propagation is required for object detection). It is a regression problem , giving the class probabilities of the detected objects. The CNN used produces the class probabilities and the bounding boxes simulataneously. It works in the following ways.

@ RESIDUAL BLOCKS (Each block : sxs)



@ Bounding Box Regression

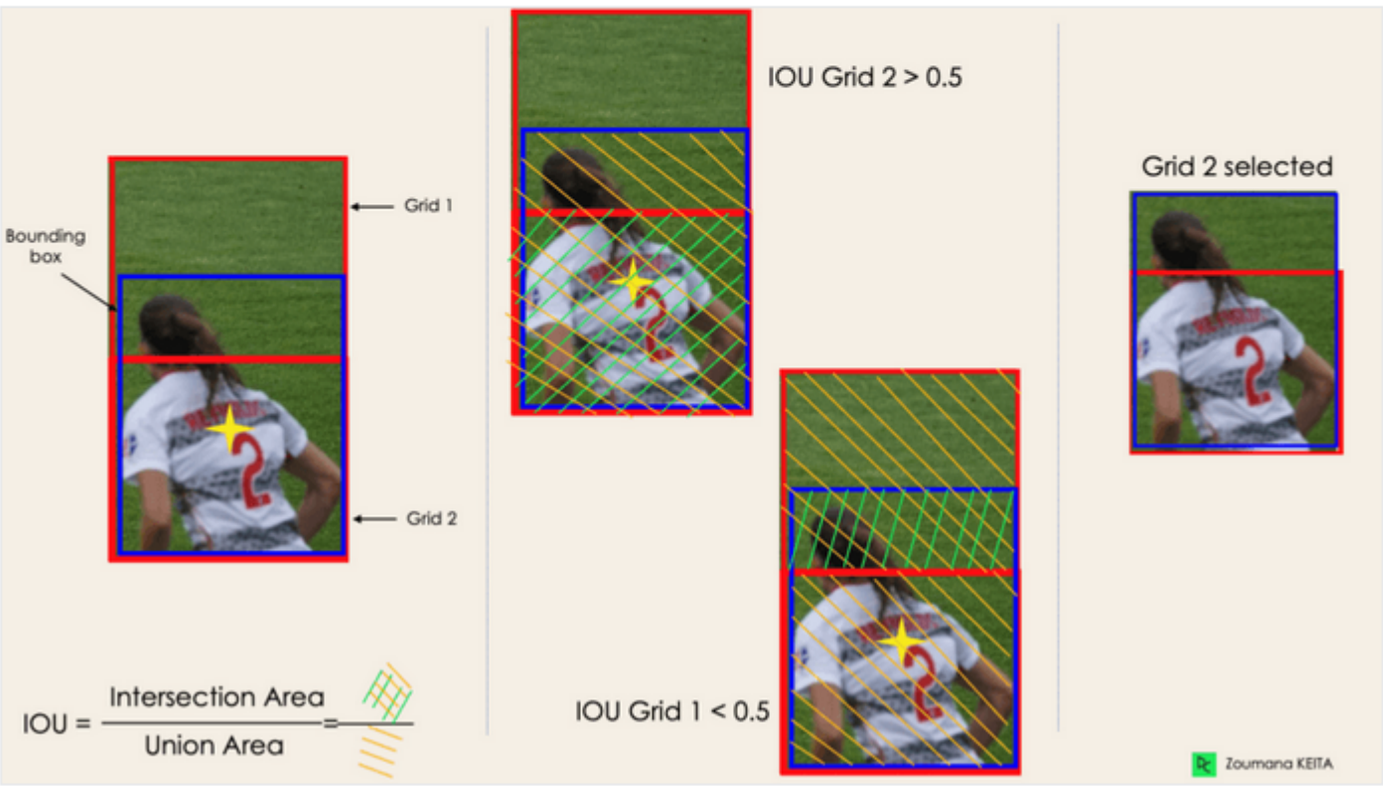


@ Intersection Over Union

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant. Here is the logic behind it :

- The user defines its IOU selection threshold, which can be, for instance, 0.5.
- Then YOLO computes the IOU of each grid cell which is the Intersection area divided by the Union Area.
- Finally, it ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.

Below is an illustration of applying the grid selection process to the bottom left object. We can observe that the object originally had two grid candidates, then only “Grid 2” was selected at the end.



VISUALIZATION

Visualization helps in ,

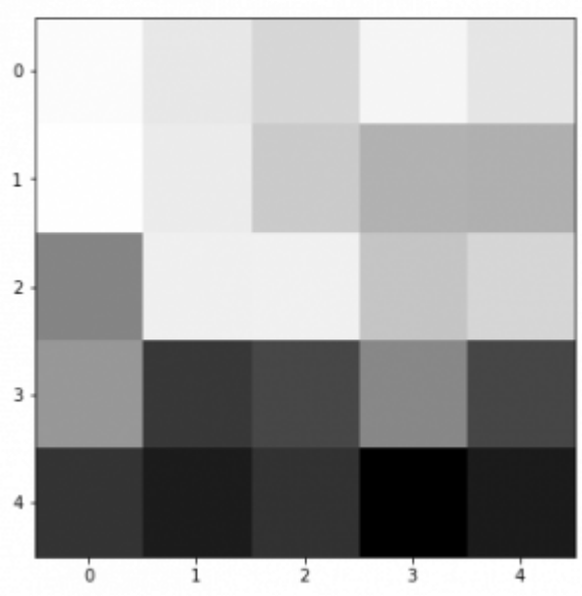
1. Understanding how the model works
2. Assistance in Hyperparameter tuning
3. Finding out the failures of the model and getting an intuition of why they fail
4. Explaining the decisions to a consumer

PRELIMINARY METHODS

1. Plotting The Model Architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
preds (Dense)                (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

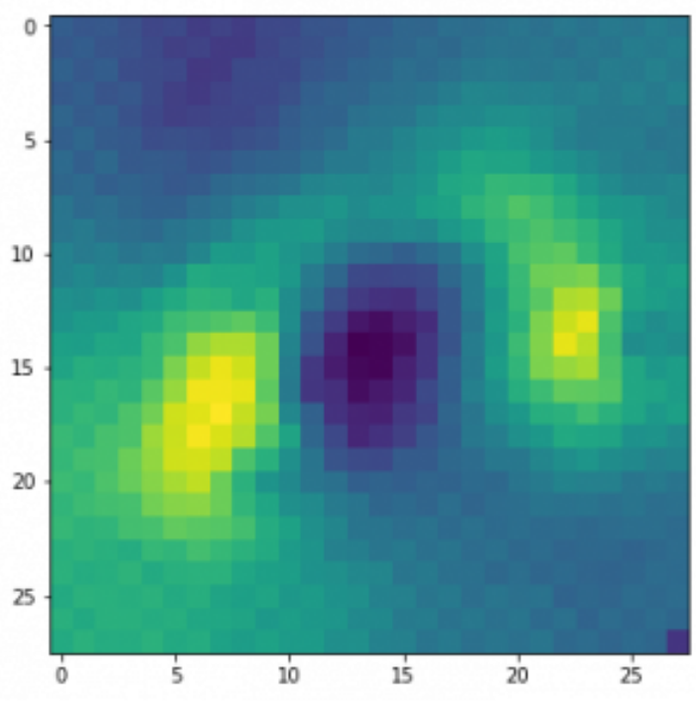
2. VISUALIZING FILTERS



ACTIVATION MAPS

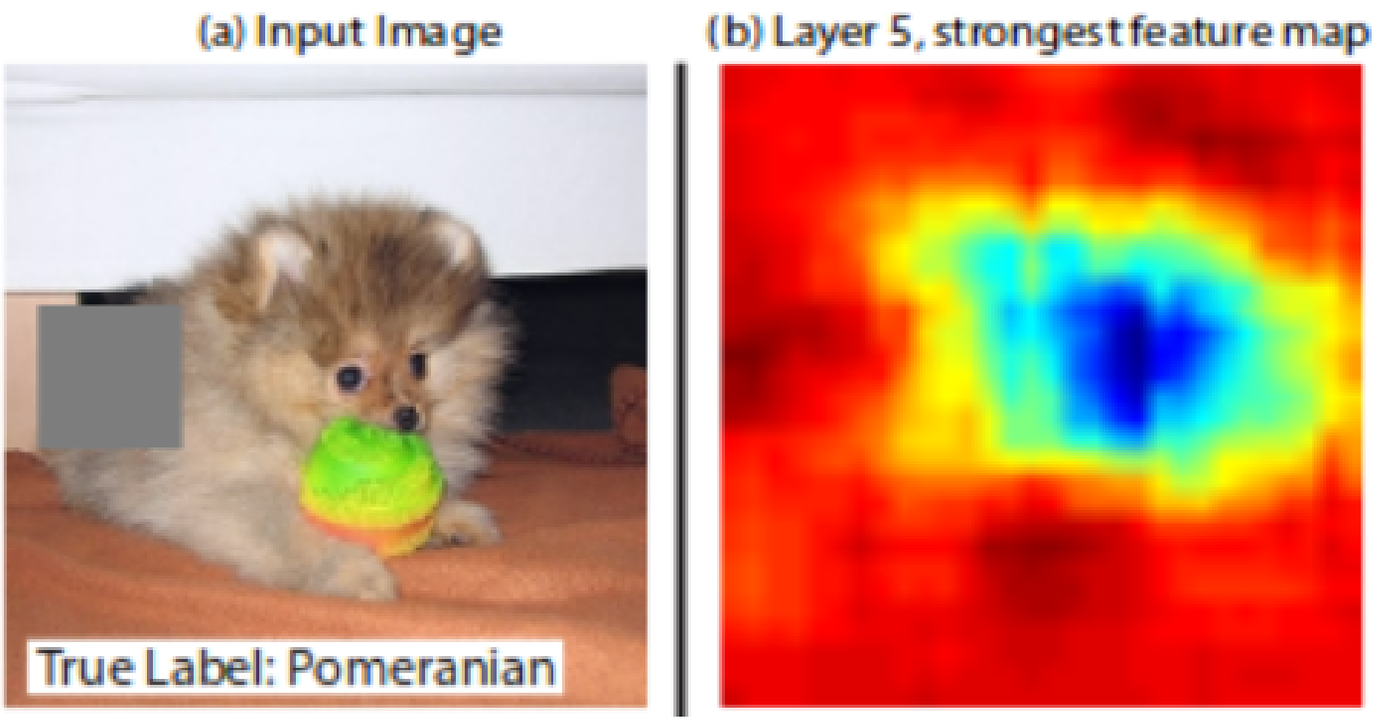
1. MAXIMAL ACTIVATIONS

This allows us to understand what sort of input patterns activate a particular filter. For example, there could be a face filter that activates when it gets the presence of a face in the image.



2. IMAGE OCCLUSIONS

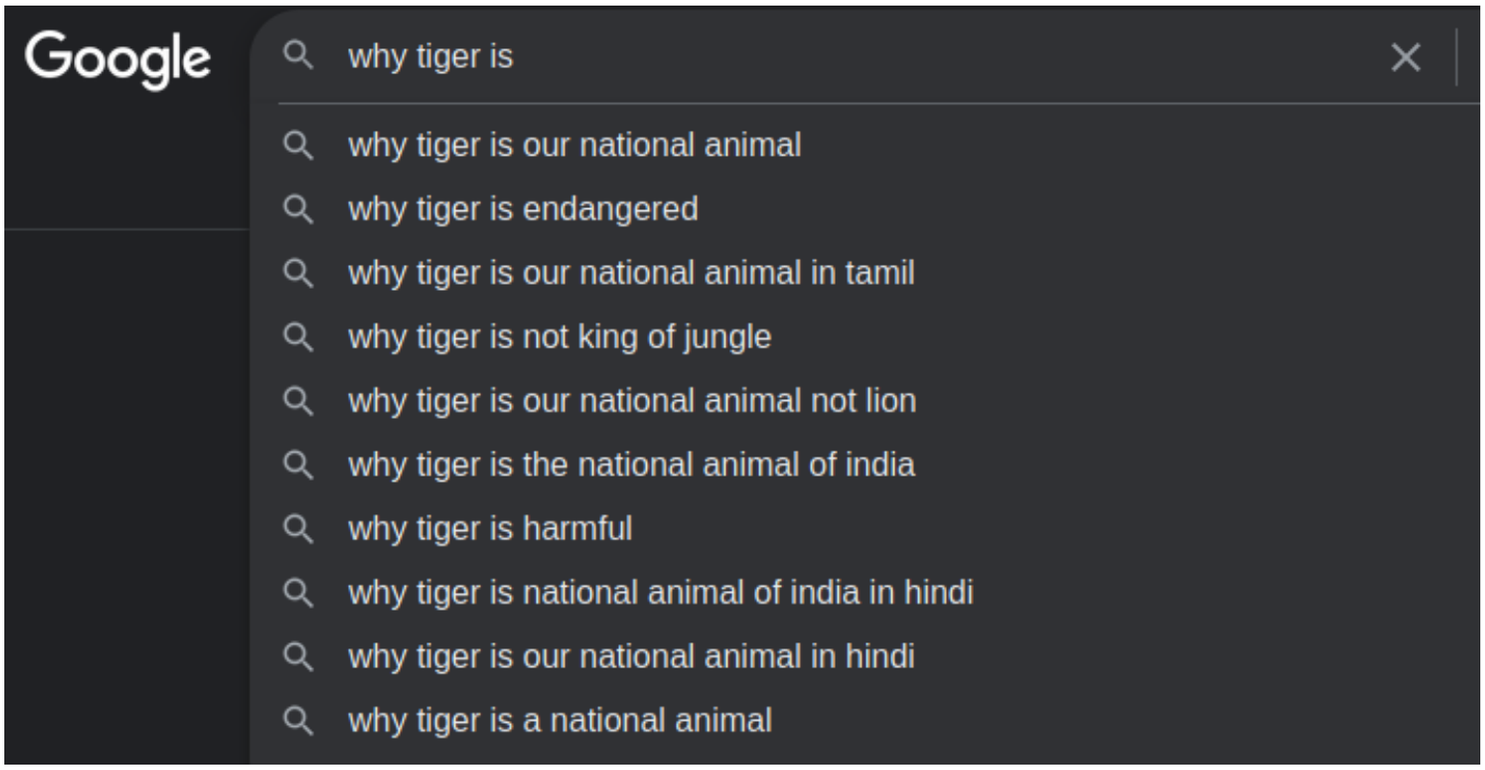
In an image classification problem, a natural question is if the model is truly identifying the location of the object in the image, or just using the surrounding context. We took a brief look at this in gradient based methods above. Occlusion based methods attempt to answer this question by systematically occluding different portions of the input image with a grey square, and monitoring the output of the classifier.



THE DE-CONVOLUTION

DeConvolution is totally opposite process from Convolution. It is also known as Transposed Convolution. In DeConvolution, the Feature map gets converted into an Image. Converting Convoluted image into Original Image is DeConvolution. Convolution adds up the information spread in various pixels to one pixel, whereas DeConvolution spreads the information present in one pixel to various pixels. Deconvolution is used for Image Segmentation. Image Segmentation is dividing an image into multiple segments or classes. Segmentation makes it easier to understand and analyze the images. Segmentation is a computationally very expensive process because we need to classify each pixel for this.

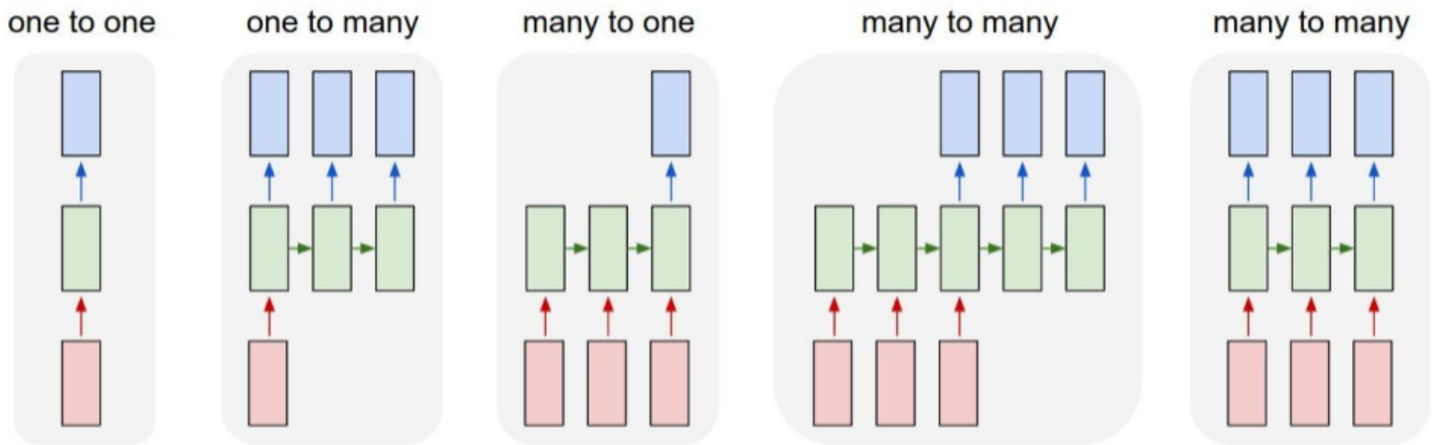
RECURRENT NEURAL NETWORK



We have shown a list of suggestions of Google. Here, we have to ponder how these suggestions were shown by the google. When we type why , the model will process it. When we type tiger , the model will collectively process tiger with the previous input why. When we type is , the model again will collectively process is with the previously processed sequence of inputs , why is. Now, the model is able to provide the good prediction for what could be next here.

RNN handles Sequential as well as Arbitary Data.

Recurrent Neural Network provides lot of flexibility.



SOME ARCHITECTURAL USES :

One To One : Vanilla Neural Network

One To Many : Image Captioning

Many To One : Sentiment Analysis

Many To Many : Machine Translation

Many To Many : Video Classification on Each Frame

How does RNN look like ?

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

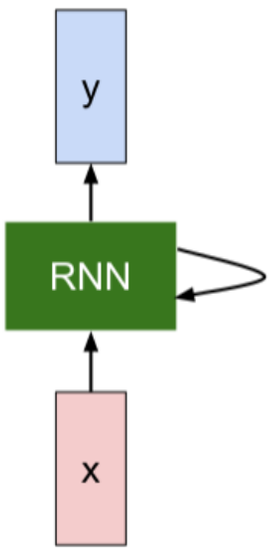
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

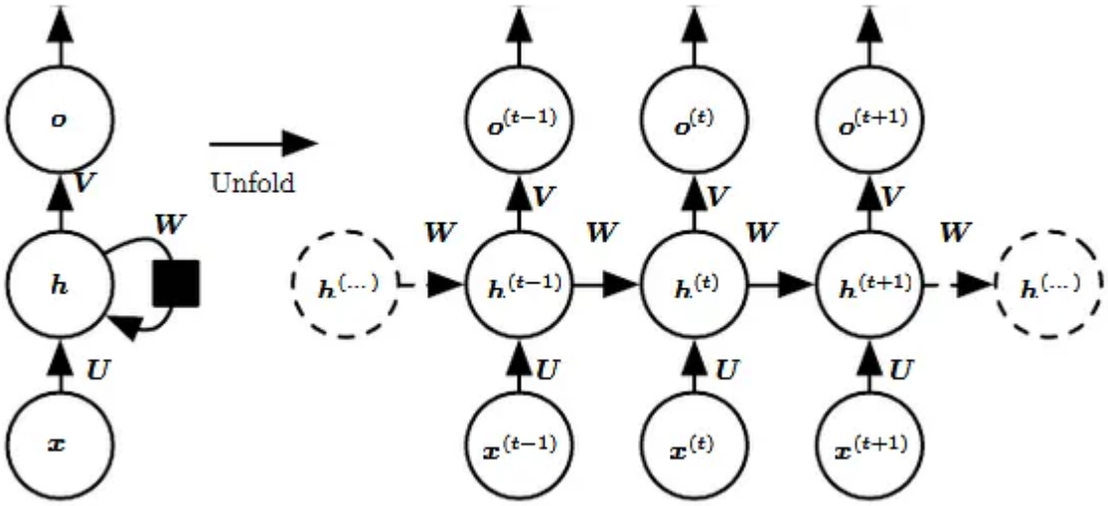
new state

old state

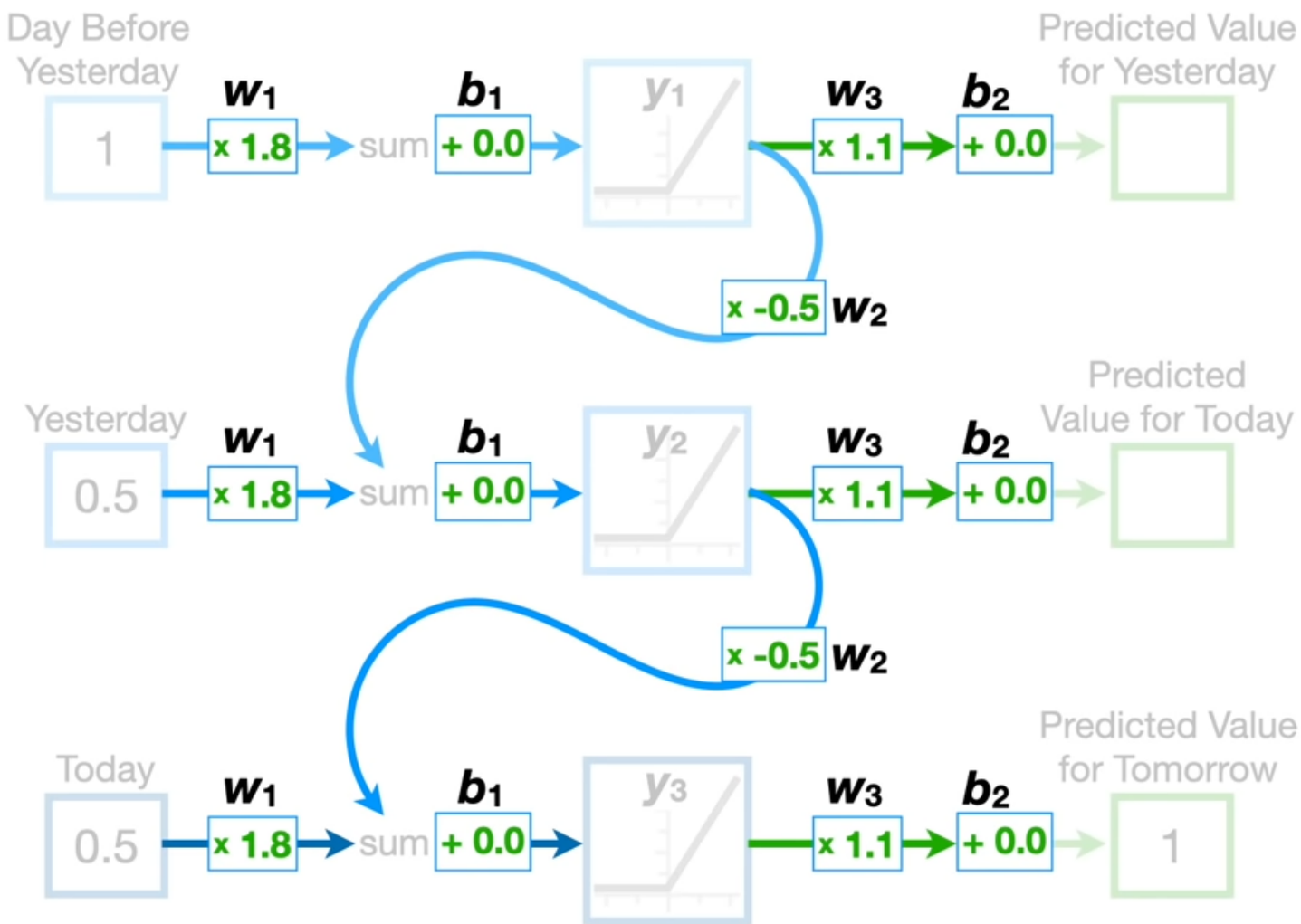
input vector at some time step

some function with parameters W



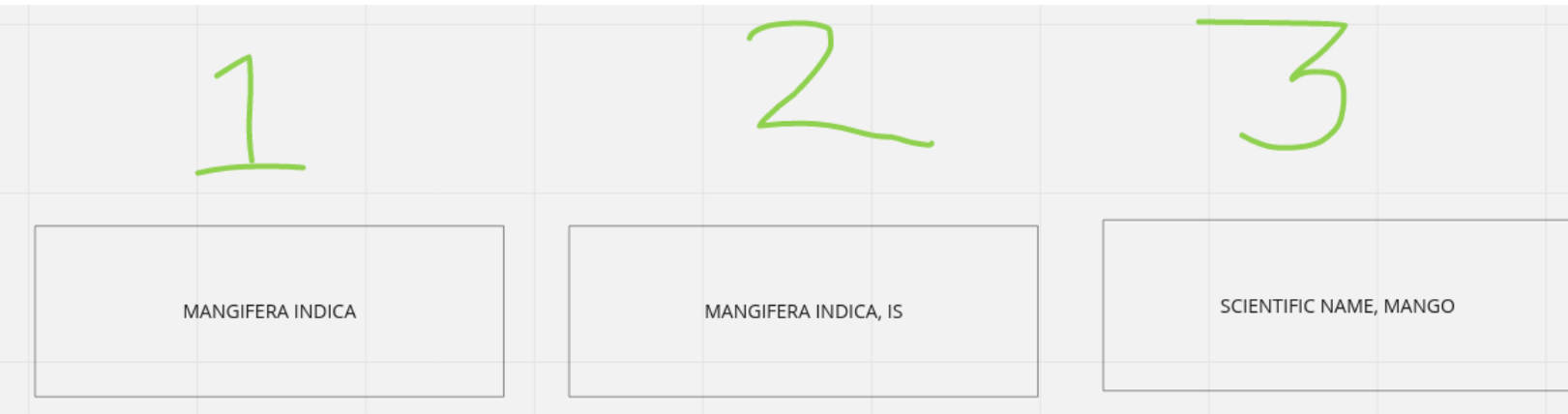


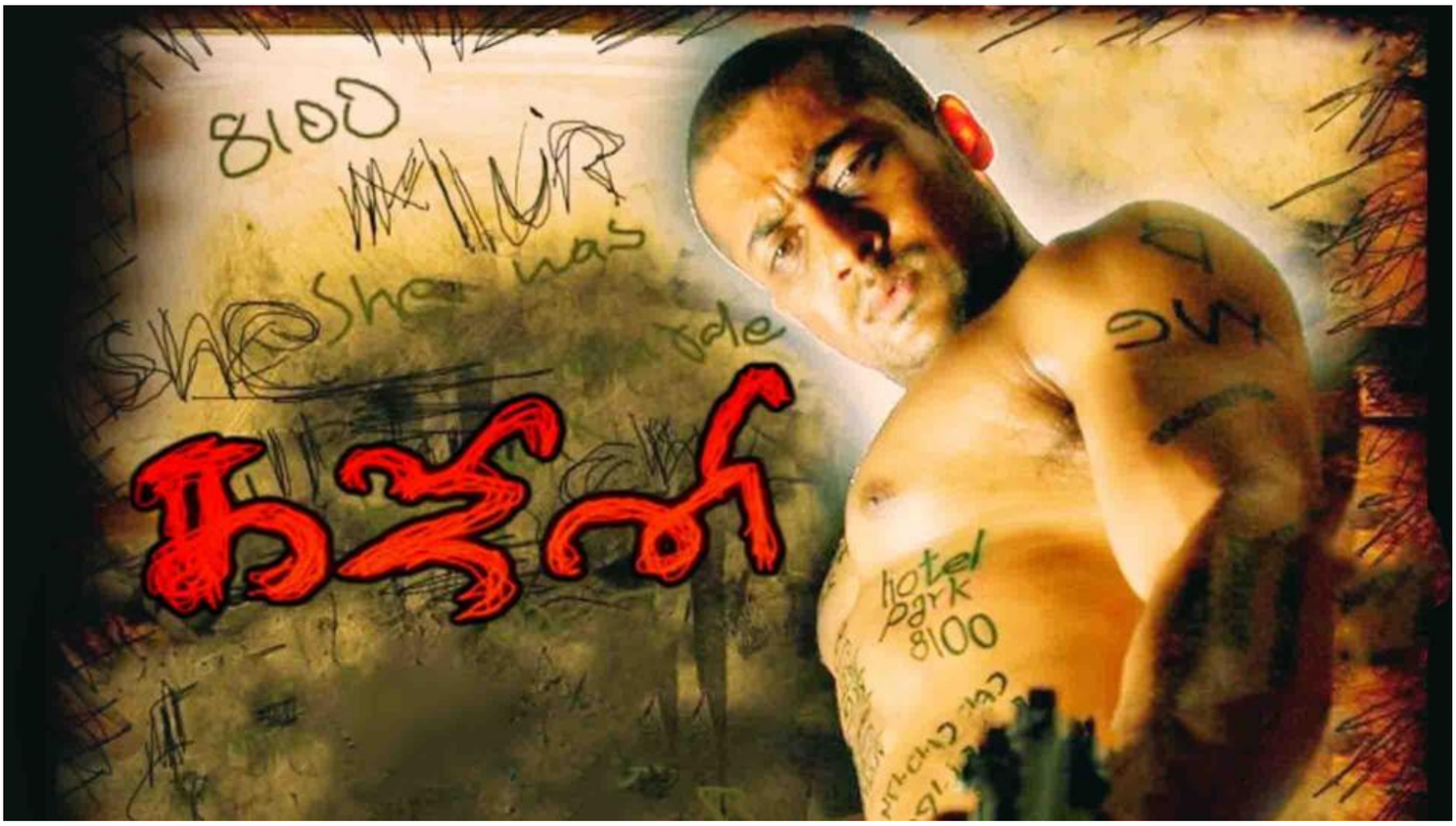
The more the network is unrolled, the more the possibilities for the problem like Exploding Gradient and Vanishing Gradient.



LONG SHORT TERM MEMORY - LSTM

A General problem in RNN is the so-called Vanishing Gradient Problem. Using tanh will make the gradient last for a considerable amount of time. The issue occurs when we are taking the derivative and derivative and so on and hence when we multiply a lot of derivatives together according to the chain rule, we end up with a vanishing value such that we can't use them. Let's consider one example. The word indica in the scientific name of Mango indicates that it is an Indian Fruit. Let's look at our RNN how it will try to predict the correct output for the following sentence. "Mangifera Indica is the scientific name of Mango so obviously it belongs to the _____" (Indian Subcontinent) The one below is just for an understanding purpose. The RNN forgets after a specific time period.





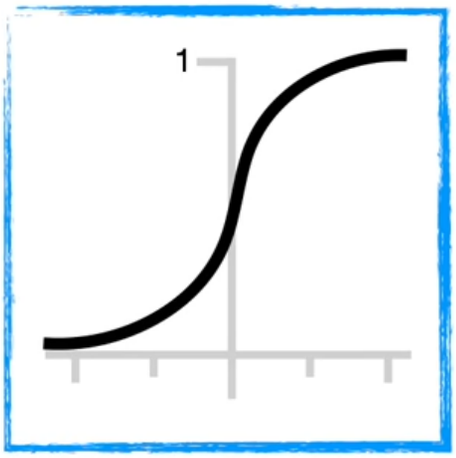
LSTM is a recurrent neural network (RNN) architecture that **REMEMBERS** values over arbitrary intervals. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. One of the advantage with LSTM is insensitivity to gap length.

WHAT'S SPECIAL WITH LSTM ?

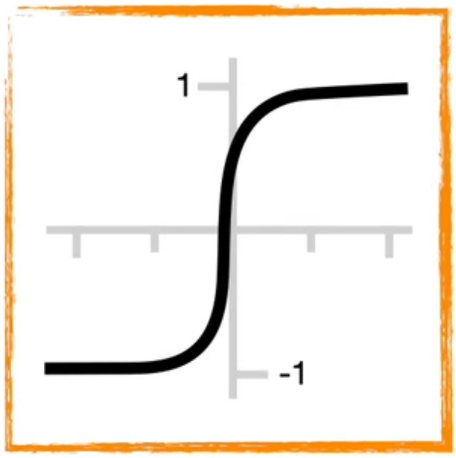
LSTM follows an approach of incorporating paths within. One path is for the Long Term Memory and another is for the Short Term Memory. The basic unit of the LSTM is a bit complicated but easily understandable.

ACTIVATION FUNCTIONS :

So, now that we know that the **Sigmoid Activation Function** turns any input into a number between **0** and **1**...

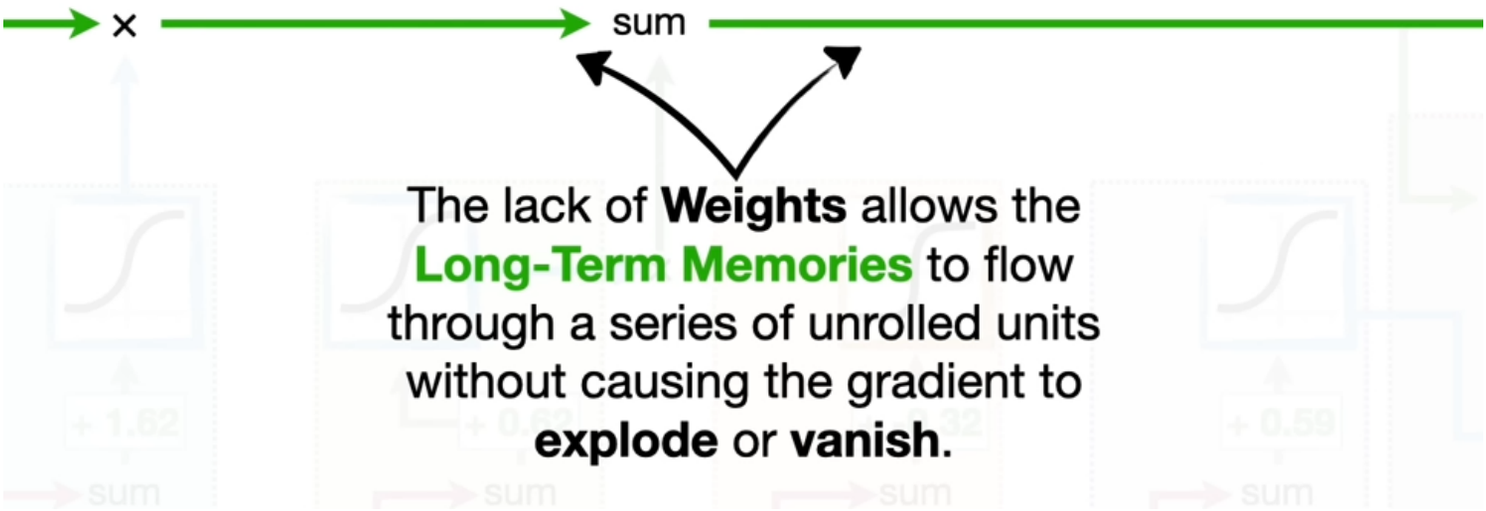


...and the **Tanh Activation Function** turns any input into a number between **-1** and **1**...

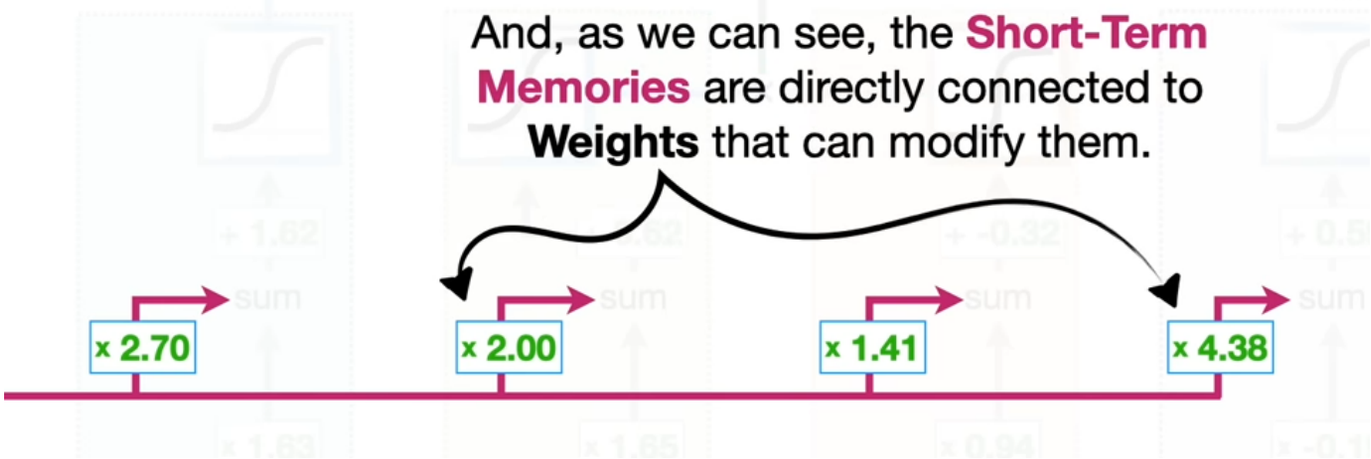


LSTM has two paths of which one is for Long Term Memory and the another one is for Short Term Memory.

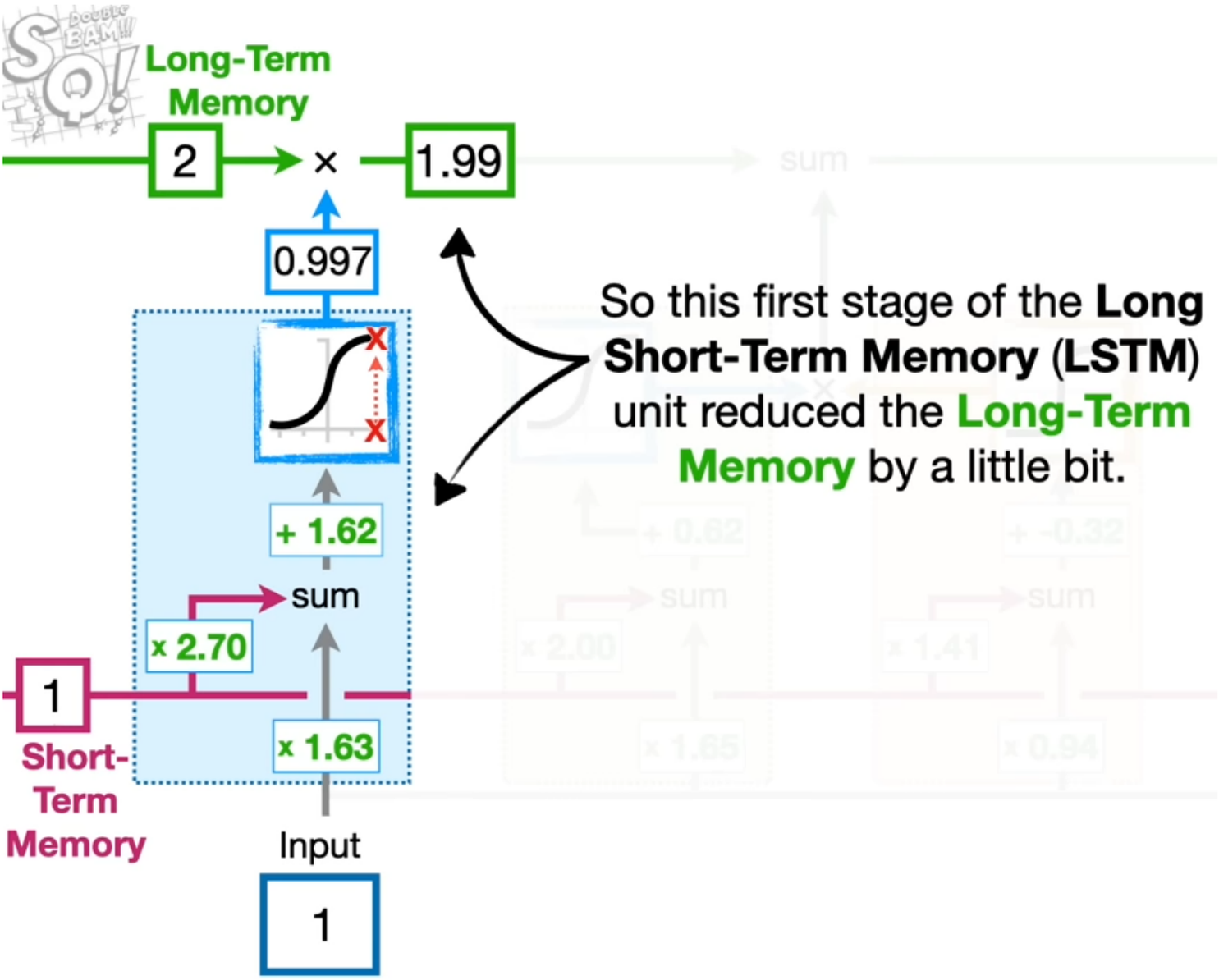
THE LONG TERM MEMORY PATH :



THE SHORT TERM MEMORY PATH :

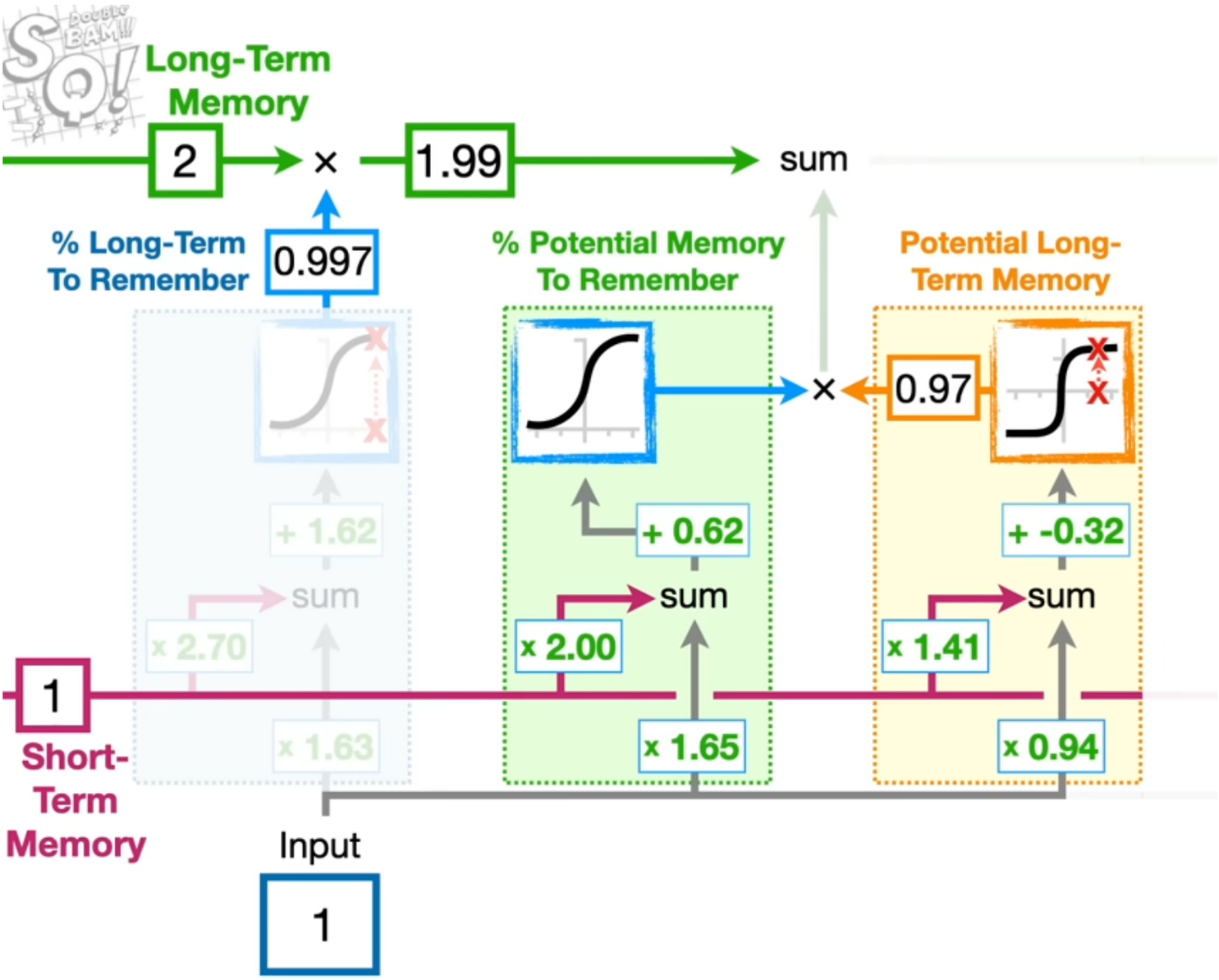


THE FORGET GATE



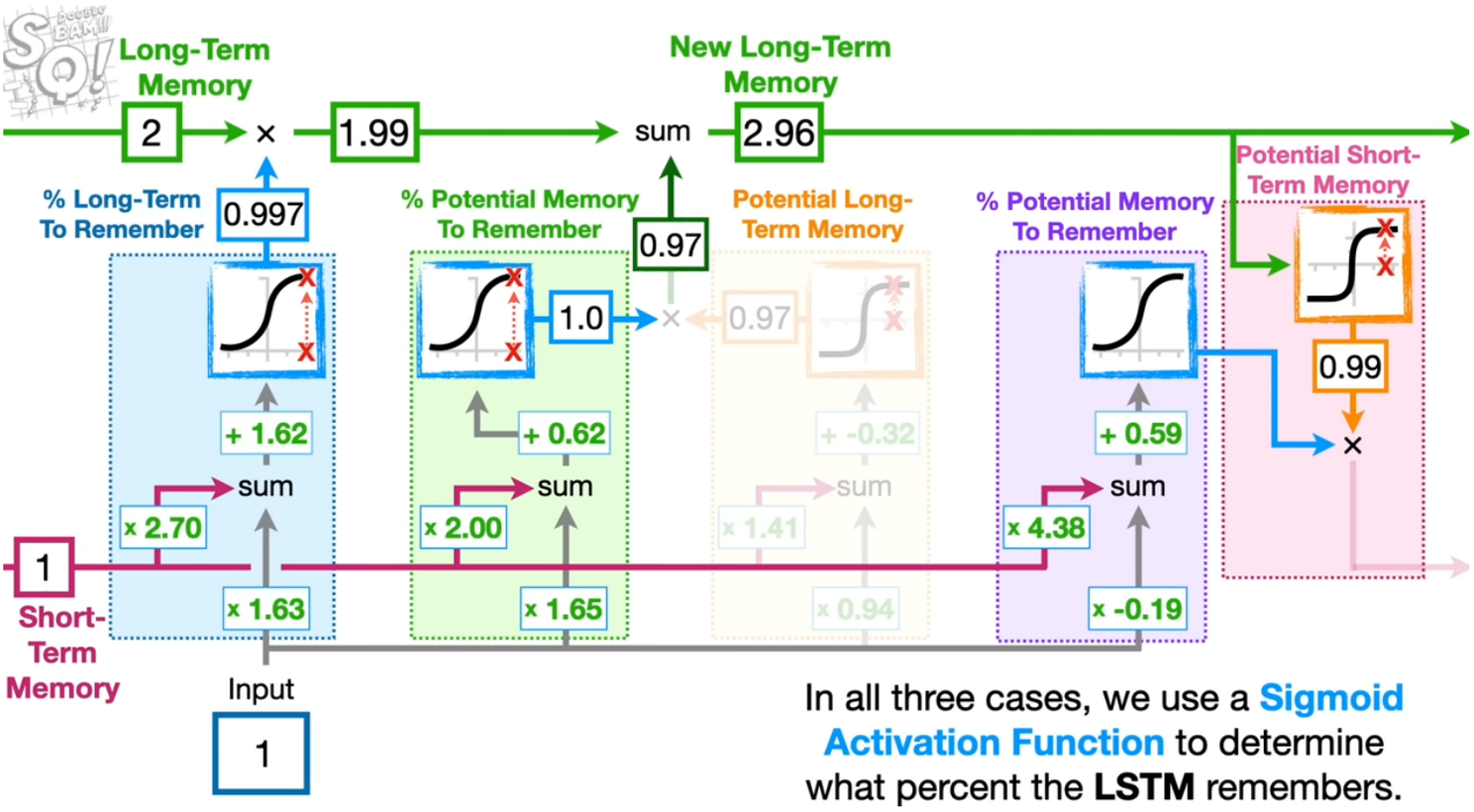
The short term memory is multiplied by it's weight and the input is also multiplied by it's weight which is later added and squished using sigmoid function. It determines what percentage of the long term memory should be remembered.

THE INPUT GATE



Here, The Potential Long Term memory to be remembered is computed using tanh activation function and the exact same thing performed in the previous step.

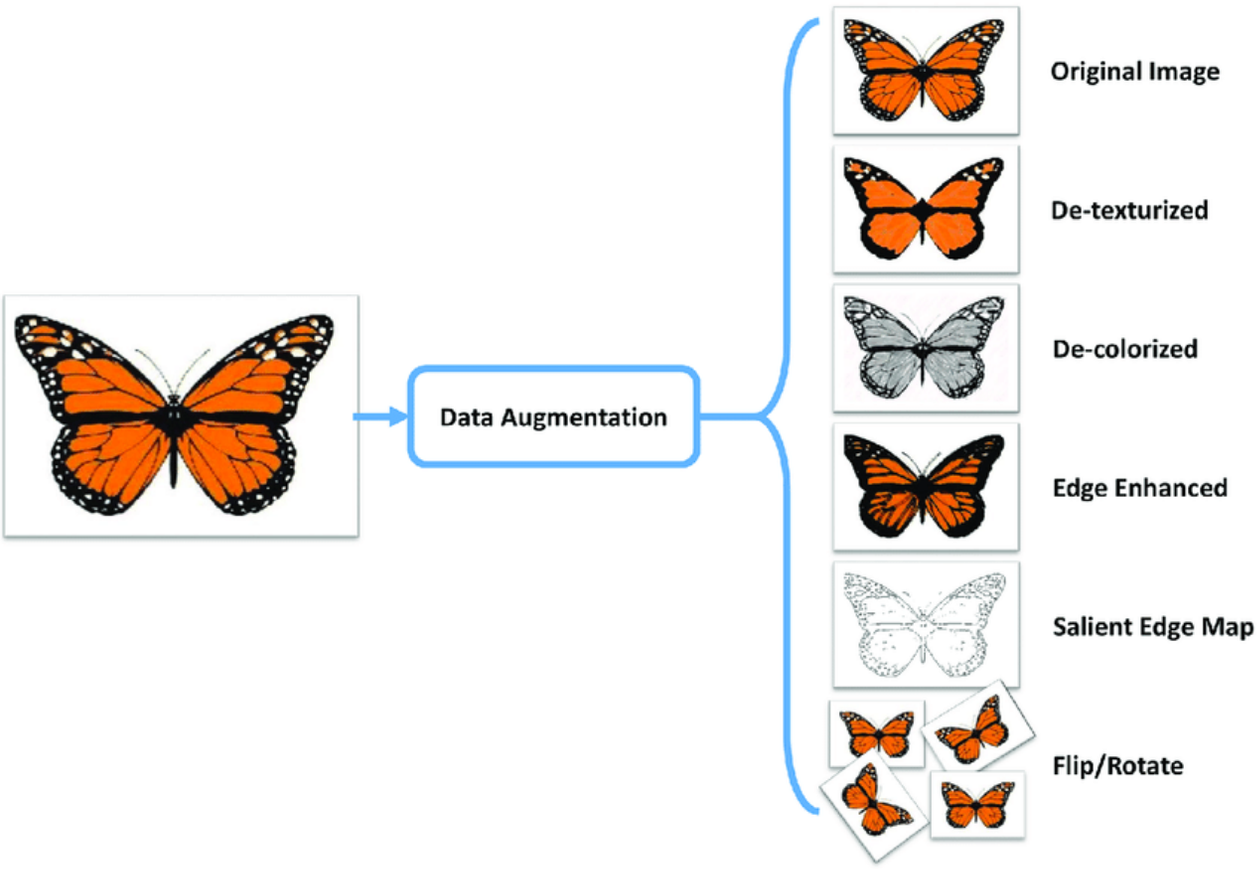
THE OUTPUT GATE :



The last arrangements determines what percentage of the potential short term memory should be remembered. LSTM is great tool for anything that has a sequence . Since the meaning of a word depends on the ones that preceded it. This paved the way for NLP and narrative analysis to leverage Neural Networks.

DATA AUGMENTATION

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points. **Augmented data** is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set. It can be used as a remedical measure for the problems of overfitting, lack of enough size in the training dataset, accuracy and so on.



SOME PERFORMANCE METRICS

Let's assume the following as predictions.

There are four metrics combinations we can list out

- True Positive : This combination tells us how many times a model correctly classifies a positive sample as Positive?
- False Negative : This combination tells us how many times a model incorrectly classifies a positive sample as Negative?
- False Positive : This combination tells us how many times a model incorrectly classifies a negative sample as Positive?
- True Negative : This combination tells us how many times a model correctly classifies a negative sample as Negative?

@ PRECISION : Ratio of correctly classified True Positive Samples to a total number of classified Positive samples *[If 75 out of 100 mixed parakeets are classified as Ringed and that is true, our precision here is 100%]*

@ RECALL : Ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples *[If 75 out of 75 ringed parakeets are classified as Ringed, our recall here is 100%]*

@ ACCURACY : If I predict for 100 times and I get correct prediction for 75 times, our accuracy is 75%.

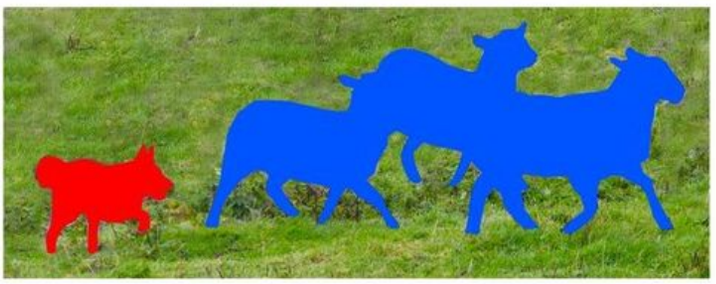
THE SEGMENTATION

A segment here is a group of pixels from an image which is separated out on the basis of the class which they belong to.

SEMANTIC SEGMENTATION

Essentially, the task of Semantic Segmentation can be referred to as **classifying** a certain class of image and **separating** it from the rest of the image classes by overlaying it with a segmentation mask. It can also be thought of as the classification of images at a pixel level.

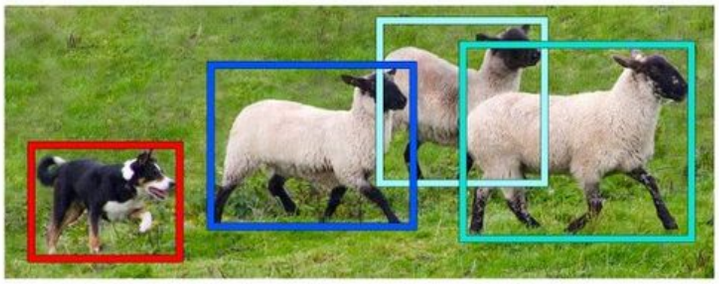
Image Recognition



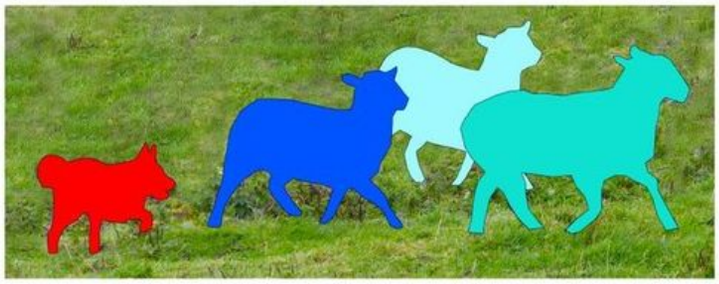
Semantic Segmentation

INSTANCE SEGMENTATION

For each pixel in the given image, it identifies the object instance it belongs to. It dives deeper than semantic segmentation and differentiates two objects with the same labels.



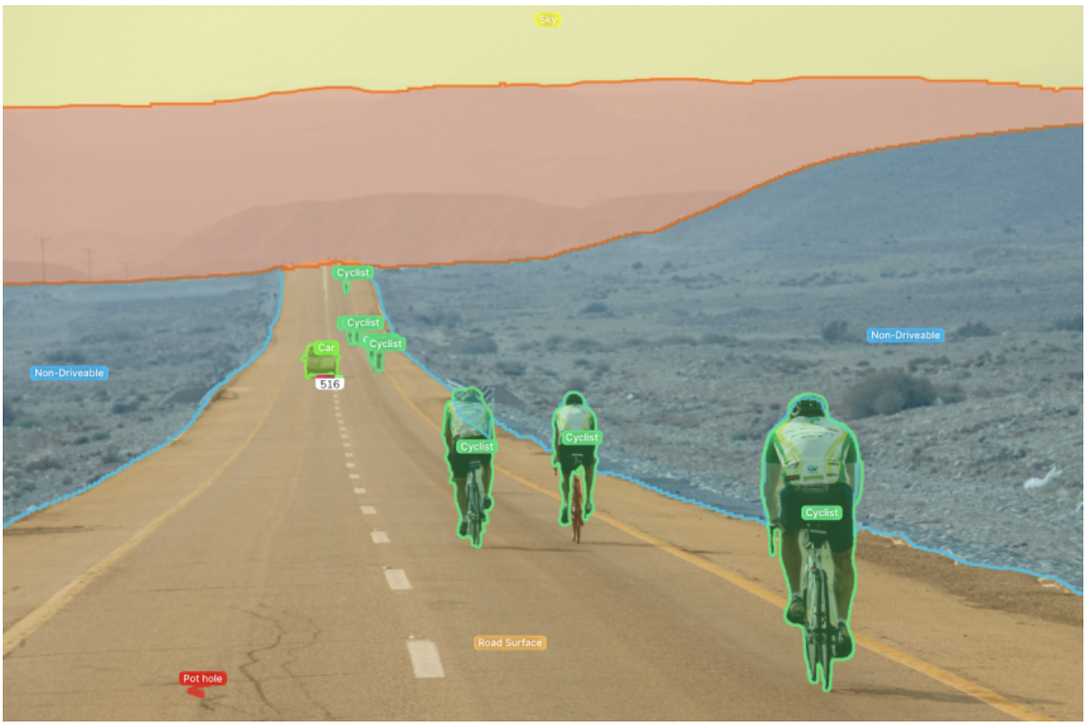
Object Detection



Instance Segmentation

PANOPTIC SEGMENTATION

The word panoptic is derived from two words: pan and optic. Pan means “all” and optic means “vision”. Panoptic segmentation, therefore, roughly means “everything visible in a given visual field”.



SOME DEEP LEARNING LIBRARIES

Caffe , PyTorch, Scikit-Learning, Tensorflow, SciPy, Keras, Theano and so on.

FINE