IIT Delhi

Indian Institute of Technology Delhi

A PROJECT REPORT

ON

"AI/ML AND IT'S APPLICATIONS IN LICENSE PLATE DETECTION"

BY

BABRA ABBAS, ANEESA KHAN, PIRZADA AFNA,
HADIYA TANVEER NAHVI,
SHALINI SHARMA

NATIONAL INSITUTE OF TECHNOLOGY
SRINAGAR

UNDER THE GUIDANCE OF

PROF. BREJESH LALL

HEAD, BHARTI SCHOOL OF TELECOM, TECHNOLOGY MANAGEMENT

PROFESSOR, ELECTRICAL ENGINEERING DEPARTMENT

CO-ORDINATOR OF ELECTRICAL DEPARTMENT

INDIAN INSTITUTE OF TECHNOLOGY, DELHI

1ST JANUARY 2021-8TH MARCH 2021

ACKNOWLEDGEMENT

We convey our utmost gratitude to our guide **Prof. Brejesh Lall, Professor, Co-Ordinator, Electrical Engineering Department, Indian Institute of Technology, Delhi,** for giving us an opportunity to work under him and guiding us all throughout the course of this internship. It has been a great honor working under his supervision. Without his expertise and guidance, this dissertation would have not been possible.

We are deeply grateful and indebted to **Ms. Sadia Hussain, PhD Scholar,** for her expert, immeasurable and valuable guidance extended to us to make our dissertation a successful one.

We would like to thank and acknowledge our colleagues for extending their valuable assistance and sharing of opinions, which has immensely helped us in completing our dissertation more adequately.

CONTENTS

AI/ML and its Applications in License Plate Detection

❖ ABSTRACT:

The objective is to detect license plates of cars using AI/ML and its application in enhanced quality of experience in image and video communication. In this project, we are using highly accurate object detection-algorithms and methods such as YOLO version 3, OpenCV version 2, TensorFlow, Darknet framework, NumPy, Matplotlib. The advantage of using YOLO architecture over others is that it is extremely fast and compared to state of art detection systems, YOLO is less likely to predict false positives. On the downside, YOLO makes more localization errors. We were able to accomplish the task of multiple license plate detection in images and videos, where the video was depicted as an array of frames. In each frame, the bounding box successfully detected the plate. At the same time, distorted license plates and license plates with noise were left undetected. No false positives were detected during the implementation.

❖ MACHINE LEARNING:

Machine learning is programming computers to optimize a performance criterion using example data and past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data and past experience. Machine learning helps find solution to many problems in vision, speech recognition and robotics. It is not just a database problem but also a part of artificial intelligence. Machine learning uses the theory of statistics in building mathematical models, because the core task is making inferences from a sample.

❖ Machine Learning Algorithm and Types:

Machine learning algorithms are organized into taxonomy, based on the desired outcome of the algorithm. Common algorithm types include:

- **Supervised Learning:** It is a machine learning algorithm where the instances are given with known labels (the corresponding correct outputs). The goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown. **[1]** Supervised learning problems can be further grouped into Regression and Classification problems.
    - **a.** **Regression Problem:** A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight".
    - **b.** **Classification Problem:** A classification problem is when the output variable is a category, such as "malignant" or "benign" tumor.

- **Unsupervised Learning:** It is a machine learning algorithm where the instances are unlabeled. By applying unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items. It allows the model to work on its own to discover patterns and information that was previously undetected. The hope is that through mimicry, the machine is forced to build a compact internal representation of its world. **[2]** Clustering and Association are two types of unsupervised learning.
    - **a.** **Clustering Algorithm:** groups similar entities together. It finds similarities in the data points and groups them together.
    - **b.** **Association Algorithm:** discovers interesting relationships between variables in large databases.

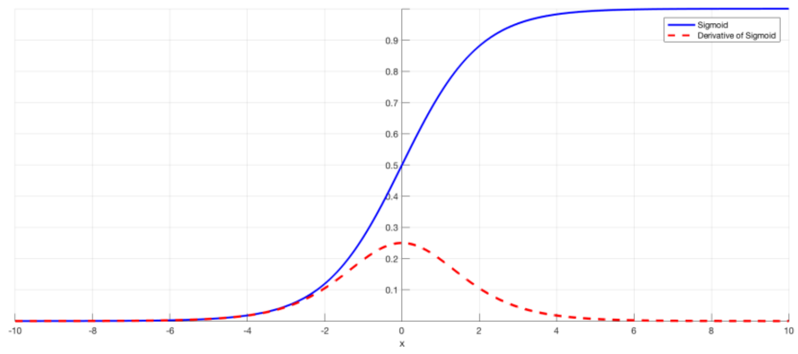
- **Semi-supervised Learning:** It is a machine learning algorithm which combines both labeled and unlabeled examples to generate an appropriate function or classifier. Semi-supervised learning falls between supervised learning (with only labeled training data) and unsupervised learning (with no labeled training data). A part of data is labeled and pseudo-labeling is done on the basis of the output of labeled data. Finally, we train our model on full training set i.e., labeled and pseudo-labeled data.

- **Reinforcement Learning:** It is a machine learning algorithm that learns a policy of how to act, given an observation of the world. Every action has some impact in the environment and the environment provides feedback that guides the learning algorithm.
  In reinforcement learning, an artificial intelligence faces a game like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

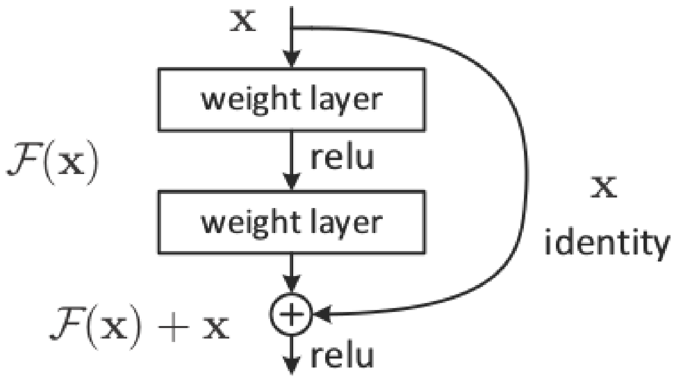
❖ Vanishing Gradient:

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approach zero, making the network hard to train. This is the vanishing gradient problem. This happens because certain activation functions, like the sigmoid function, squishes a large input space into a small output space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

As an example, in the given image the sigmoid function and its derivative are depicted. When the inputs of the sigmoid function become larger or smaller (when |x| becomes bigger), the derivative becomes close to zero.

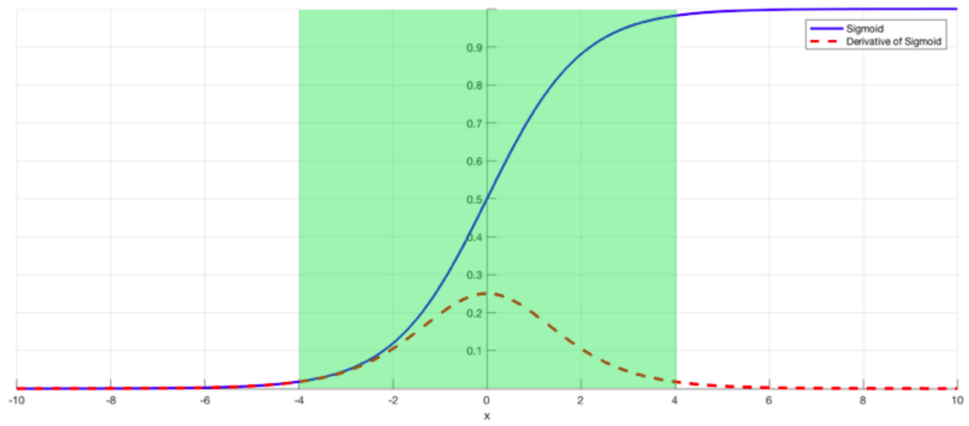


The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative. Residual networks are another solution, as they provide residual connections straight to earlier layers. The residual connection directly adds the value at the beginning of the block, x, to the end of the block (F(x) + x). This residual connection doesn't go through activation functions that squash the derivatives, resulting in a higher overall derivative of the block.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Finally, batch normalization layers can also resolve the issue. As stated before, the problem arises when a large input space is mapped to a small one, causing the derivatives to disappear. This is most clearly seen at when |x| is big. Batch normalization reduces this problem by simply normalizing the input so |x| doesn't reach the outer edges of the sigmoid function. It normalizes the input so that most of it falls in the green region, where the derivative isn't too small.



❖ Statistical Fit:

In statistics, a fit refers to how well you approximate a target function. This is good terminology to use in machine learning, because supervised machine learning algorithms seek to approximate the unknown underlying mapping function for the output variables given the input variables. Statistics often describe the goodness of fit which refers to measures used to estimate how well the approximation of the function matches the target function.

Some of these methods are useful in machine learning (e.g., calculating the residual errors), but some of these techniques assume we know the form of the target function we are approximating, which is not the case in machine learning. If we knew the form of the target function, we would use it directly to make predictions, rather than trying to learn an approximation from samples of noisy training data.

❖ Overfitting in Machine Learning:

Overfitting is a modeling error that occurs when a function is too closely fit to a limited set of data points. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model’s ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

❖ How to Limit Overfitting:

By far the most common problem in applied machine learning is overfitting. It is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation, we actually care the most about, namely how well the algorithm performs on unseen data.

There are two important techniques that can be used when evaluating machine learning algorithms to limit overfitting:

a. Using a re-sampling technique to estimate model accuracy.
b. Holding back a validation dataset.

The most popular re-sampling technique is k-fold cross validation. It allows us to train and test our model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

A validation dataset is simply a subset of our training data that we hold back from our machine learning algorithms until the very end of our project.

After we have selected and tuned our machine learning algorithms on our training dataset, we can evaluate the learned models on the validation dataset to get a final objective idea of how the models might perform on unseen data. Using cross validation is a gold standard in applied machine learning for estimating model accuracy on unseen data. If we have the data, using a validation dataset is also an excellent practice.

❖ Underfitting in Machine Learning:

Underfitting refers to a model that can neither generalize to new data nor model the training data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

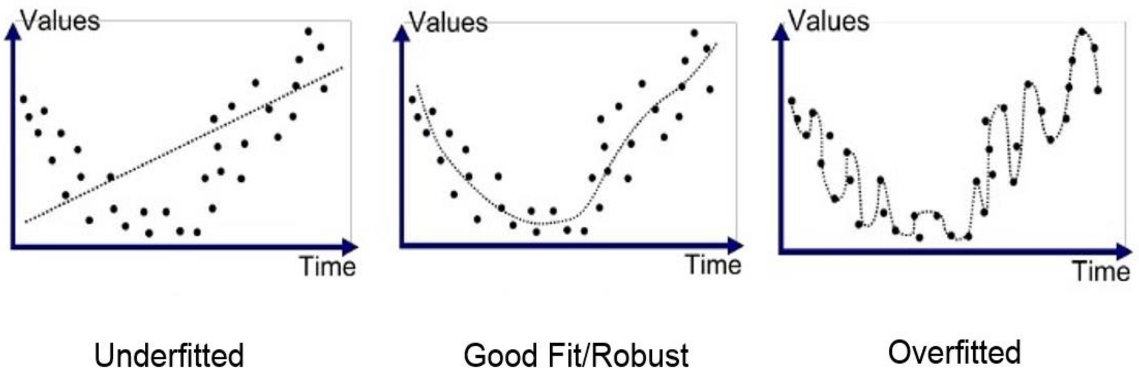
❖ A Good Fit in Machine Learning:

Ideally, we prefer to select a model at the sweet spot between underfitting and overfitting. This is the goal, but is very difficult to do in practice. To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data. We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process.

Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset. At the same time the error for the test set starts to rise again as the model's ability to generalize decreases. The sweet spot is the point just before the error on the test dataset starts to increase, where the model has good skill on both the training dataset and the unseen test dataset.

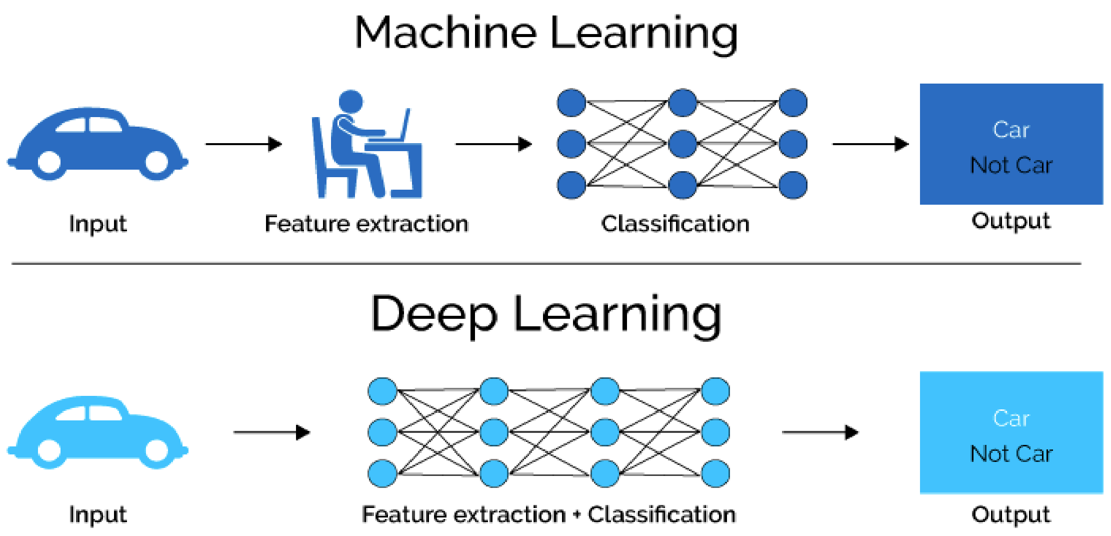
We can perform this experiment with our favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the test set is no longer unseen or a standalone objective measure.

There are two additional techniques that can be used to help find the sweet spot in practice: re-sampling methods and a validation dataset.



❖ DEEP LEARNING:

Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator formally to specify all of the knowledge needed by the computer. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones; a graph of these hierarchies would be many layers deep.

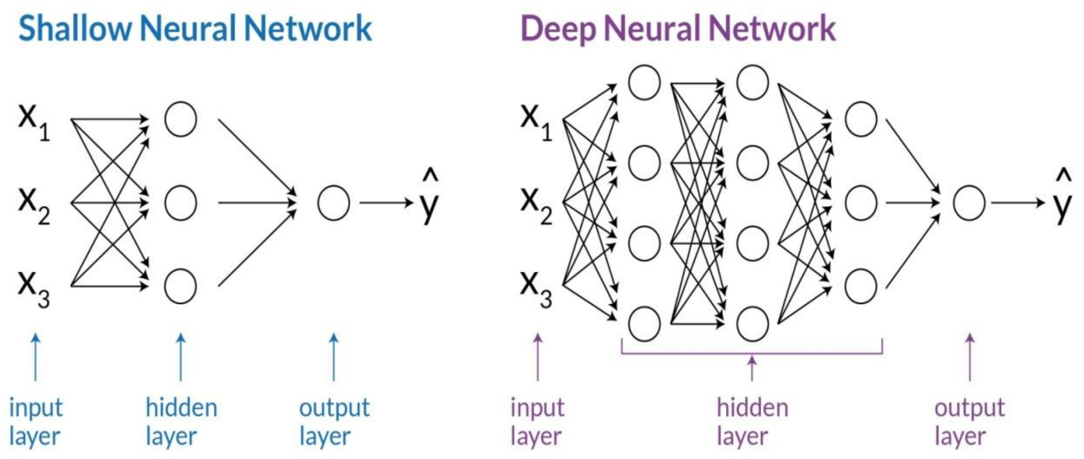


❖ NEURAL NETWORKS:

Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment.
A neuron in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear. In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers.

- **Input Layer:** collects input patterns i.e., the initial data for the neural network. It passes the data directly to the first hidden layer.
- **Output Layer:** has classification or output signals to which input patterns may map. It produces the result for given inputs.
- **Hidden Layers:** fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.



❖ Activation Functions:

In Neural Network the activation function defines if given node should be activated or not based on the weighted sum (z). Some of the most popular activation functions are:

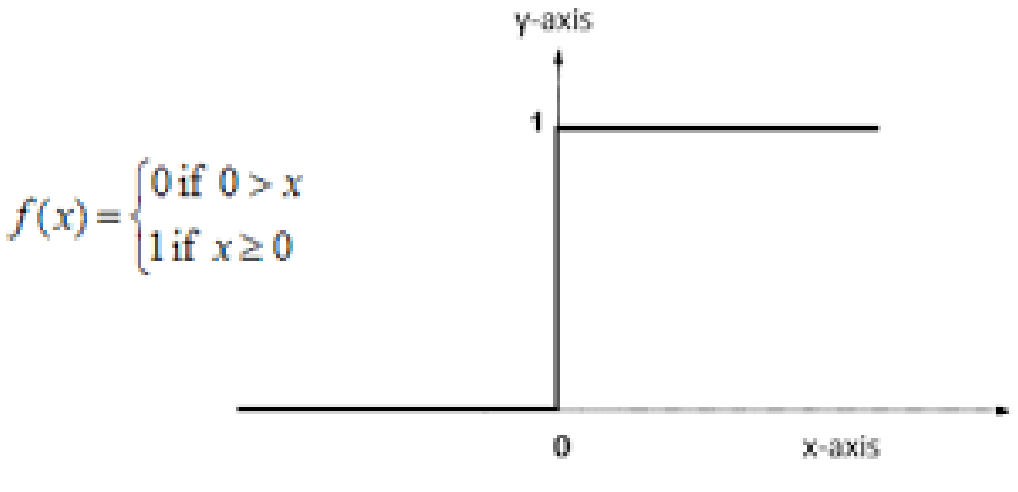
- <u>**Step Function:**</u>

Step Function is one of the simplest kinds of activation functions. In this, we consider a threshold value and if the value of net input is greater than the threshold, then the neuron is activated.

If (z > threshold) — "activate" the node (value 1)
If (z < threshold) — don't "activate" the node (value 0)

<u>Disadvantages:</u>

a. The problem with a step function is that it does not allow multi-value outputs. For example, it cannot support classifying the inputs into one of several categories.



- **<u>Linear Function:</u>**

A linear activation function takes the form:

$$A=cx$$

It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.

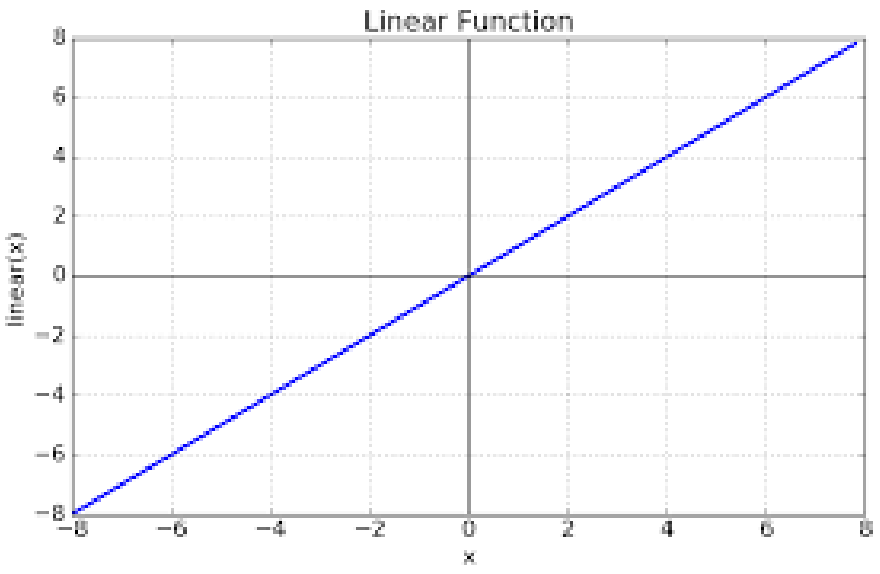
<u>Advantages:</u>

a. Linear function is better than a step function because it allows multiple outputs, not just two.

<u>Disadvantages:</u>

a. It is not possible to use back propagation (gradient descent) to train the model. The derivative of the function is a constant, and has no relation to the input, x. So, it's not possible to go back and understand which weights in the input neurons can provide a better prediction**.**
b. All layers of the neural network collapse into one, with linear activation functions no matter how many layers in the neural network, the last layer will be a linear function of the first layer. So, a linear activation function turns the neural network into just one layer.

A neural network with a linear activation function is simply a linear regression model. It has limited power and ability to handle complexity varying parameters of input data.



- **<u>Sigmoid Function</u>:**

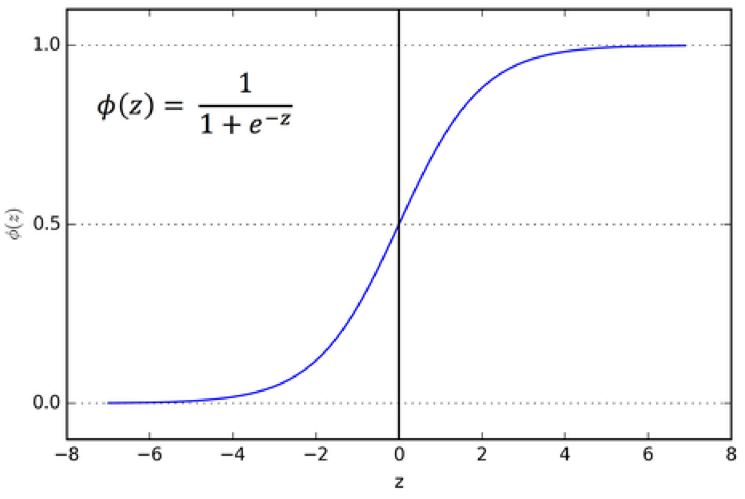
It is a mathematical function having a characteristic that can take any real value and map it to a value between 0 and 1. The sigmoid function is also called logistic function.

$$\mathbf{Y = 1 / 1{+}e^{-z}}$$

So, if the value of z goes to positive infinity, then the predicted value of y will become 1 and if it goes to negative infinity, then the predicted value of y will become 0. And if the outcome of the sigmoid function is more than 0.5, then we classify that label as class 1 or positive class and if it is less than 0.5, then we can classify it to negative class or label as class 0.

<u>Advantages:</u>

a. Smooth gradient prevents jumps in output values.
b. Output values are bound between 0 and 1, normalizing the output of each neuron.



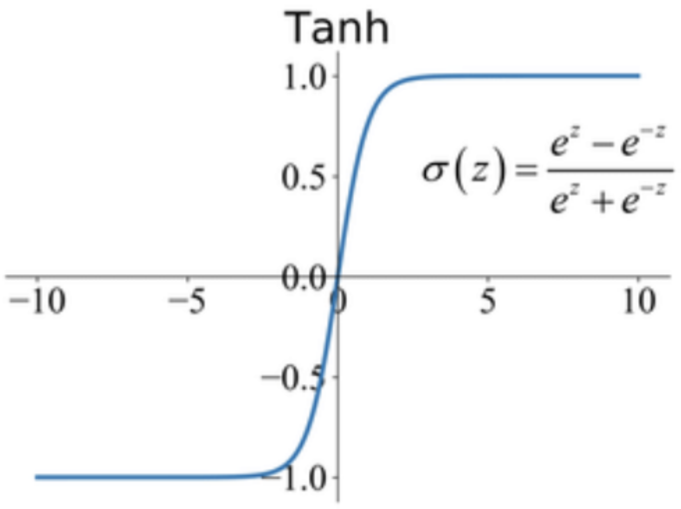
<u>Disadvantages:</u>

a. For very high or very low values of z, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
b. Outputs are not zero centered**.**

- **<u>Tanh Function:</u>**

In neural networks, as an alternative to sigmoid function, hyperbolic tangent function could be used as activation function. When we back propagate, derivative of activation function would be involved in calculation for error effects on weights. Derivative of hyperbolic tangent function has a simple form just like sigmoid function. This explains why hyperbolic tangent is common in neural networks.
Hyperbolic Tangent Function is given as:

$$\mathbf{tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})}$$



<u>Advantages:</u>

a. The advantage of tanh function over the sigmoid function is that its derivative is steeper, which means it can get more values. This means that it will be more efficient because it has a wider range for faster learning and grading.
b. Tanh function has all the advantages of sigmoid function. It is a zero centric function i.e., output of values which are far away from centroid is close to zero.

<u>Disadvantages:</u>

a. It suffers from vanishing gradient problem.

- **<u>ReLU Function:</u>**

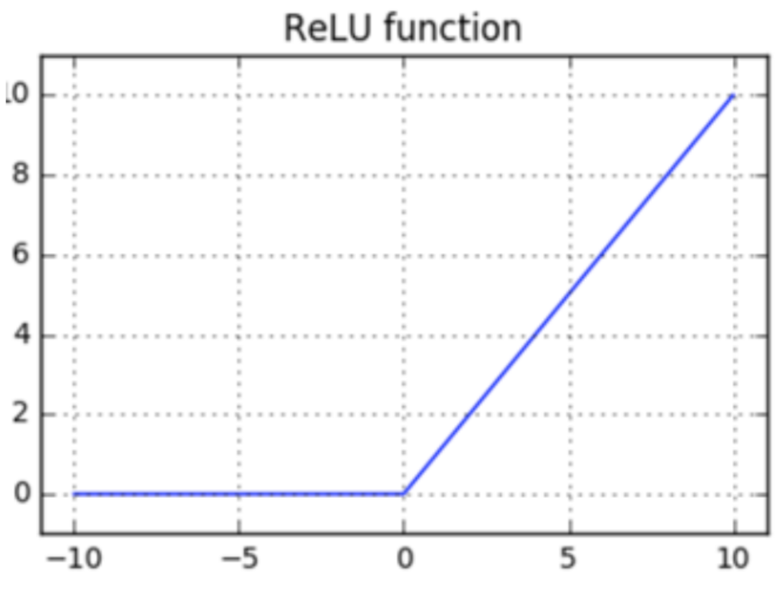
The rectified linear activation function or ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer perceptron and convolutional neural networks.

<u>Advantages:</u>

a. It is computationally efficient. It allows the network to converge very quickly.
b. It is non-linear. Although it looks like a linear function, ReLU has a derivative function and allows for back propagation.

<u>Disadvantages:</u>

a. The Dying ReLU problem—when input approach zero, or are negative, the gradient of the function becomes zero. The network cannot perform back propagation and cannot learn.



- **Leaky ReLU:**

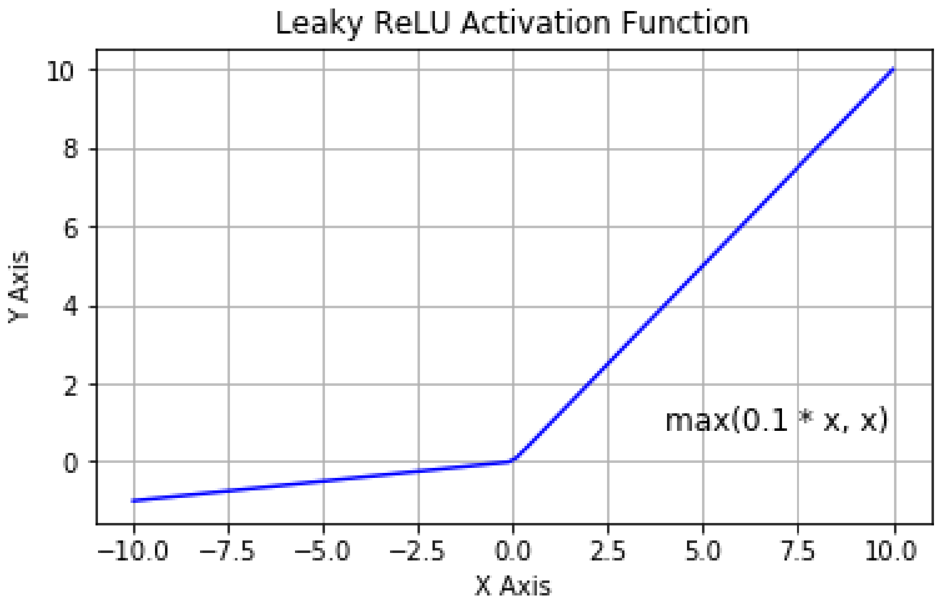
Leaky ReLU function is an improved version of the ReLU activation function. As for the ReLU activation function, the gradient is zero for all the values of inputs that are less than zero, which would deactivate the neurons in that region and may cause dying ReLU problem. Leaky ReLU is defined to address this problem.

Advantages:

a. It prevents dying ReLU problem. This variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values.

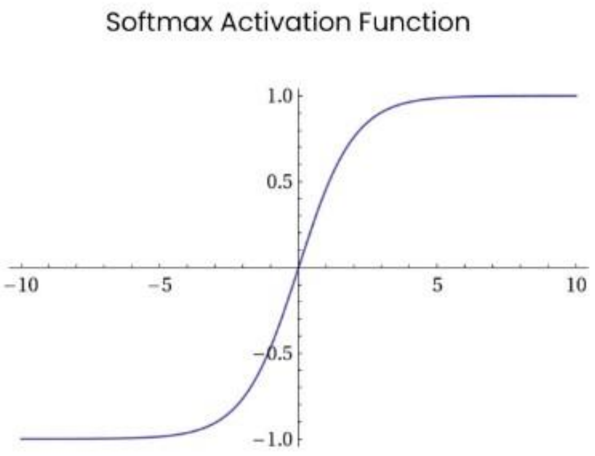
Disadvantages:

a. The results in this case are not consistent. The leaky ReLU does not provide consistent predictions for negative input values.



- **SoftMax Function:**

SoftMax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. Specifically, the network is configured to output N values, one for each class in the classification task.
SoftMax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the SoftMax function is interpreted as the probability of membership for each class.



Advantages:

a. It is able to handle multiple classes. It normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.

b. It is useful for output neurons. Typically, SoftMax is used only for the output layer in neural networks that need to classify inputs into multiple categories.

- **Cost Function:**

A cost function is a mechanism utilized in supervised machine learning. The cost function returns the error between predicted outcomes compared with the actual outcomes. The aim of supervised machine learning is to minimize the overall cost, thus optimizing the correlation of the model to the system that it is attempting to represent.

In the case of Linear Regression, the cost function is given as

$$\mathrm{J}(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}[h_\Theta(x^{(i)}) - y^{(i)}]^2$$

In case of Logistic Regression, the cost function is given as

$$\mathrm{h}_\Theta(x) = g(\Theta^T x)$$

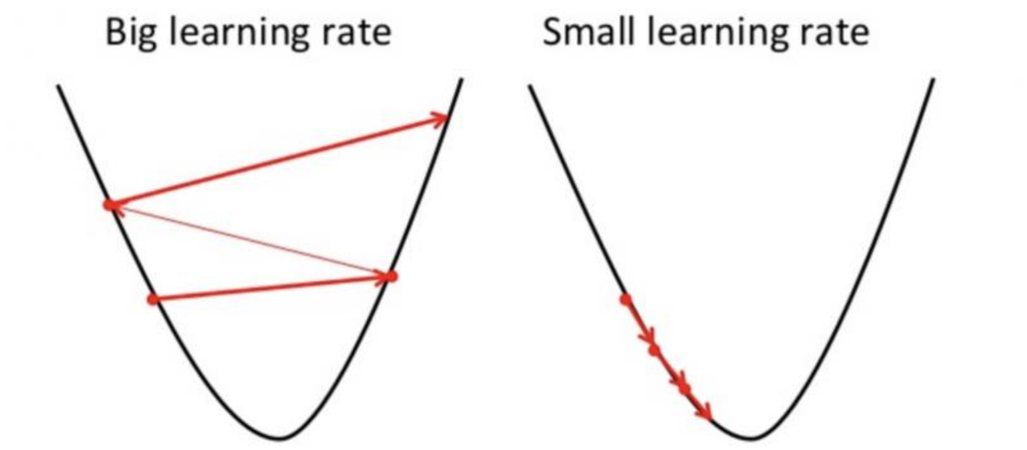
It will result in a non-convex cost function with local optima's which is a very big problem for gradient descent to compute the global optima.

$$\mathrm{Cost}(\mathrm{h}_\Theta(x), y) = \begin{cases} -log(h_\Theta(x)) & if \quad y = 1 \\ -log(1 - h_\Theta(x)) & if \quad y = 0 \end{cases}$$

❖ Gradient Descent:

Gradient descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible. How big the steps that the gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.

For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent. If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while. So, the learning rate should never be too high or too low for this reason. We can check if the learning rate is doing well by plotting it on a graph.



❖ Types of Neural Networks:

a. <u>Artificial Neural Network (ANN):</u>

Artificial Neural Networks are a technology based on studies of the brain and nervous system. These networks emulate a biological neural network but they use a reduced set of concepts from biological neural systems. Specifically, ANN models simulate the electrical activity of the brain and nervous system.

Processing elements (also known as either a neurodes or perceptron are connected to other processing elements. Typically, the neurodes are arranged in a layer or vector, with the output of one layer serving as the input to the next layer and possibly other layers. A neurode may be connected to all or a subset of the neurodes in the subsequent layer, with these connections simulating the synaptic connections of the brain.

Weighted data signals entering a neurode simulate the electrical excitation of a nerve cell and consequently the transference of information within the network or brain. The input values to a processing element are multiplied by a connection weight that simulates the strengthening of neural pathways in the brain. It is through the adjustment of the connection strengths or weights that learning is emulated in ANNs.

b. <u>Recurrent Neural Network (RNN):</u>

Recurrent Neural Networks (RNN) are a type of neural network where the output from previous step is fed as input to the current step**.** In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a hidden layer. The main and most important feature of RNN is hidden state, which remembers some information about a sequence.

RNN has memory which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

c. Convolutional Neural Network (CNN):

CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns.

- **Building blocks of CNN Architecture:**

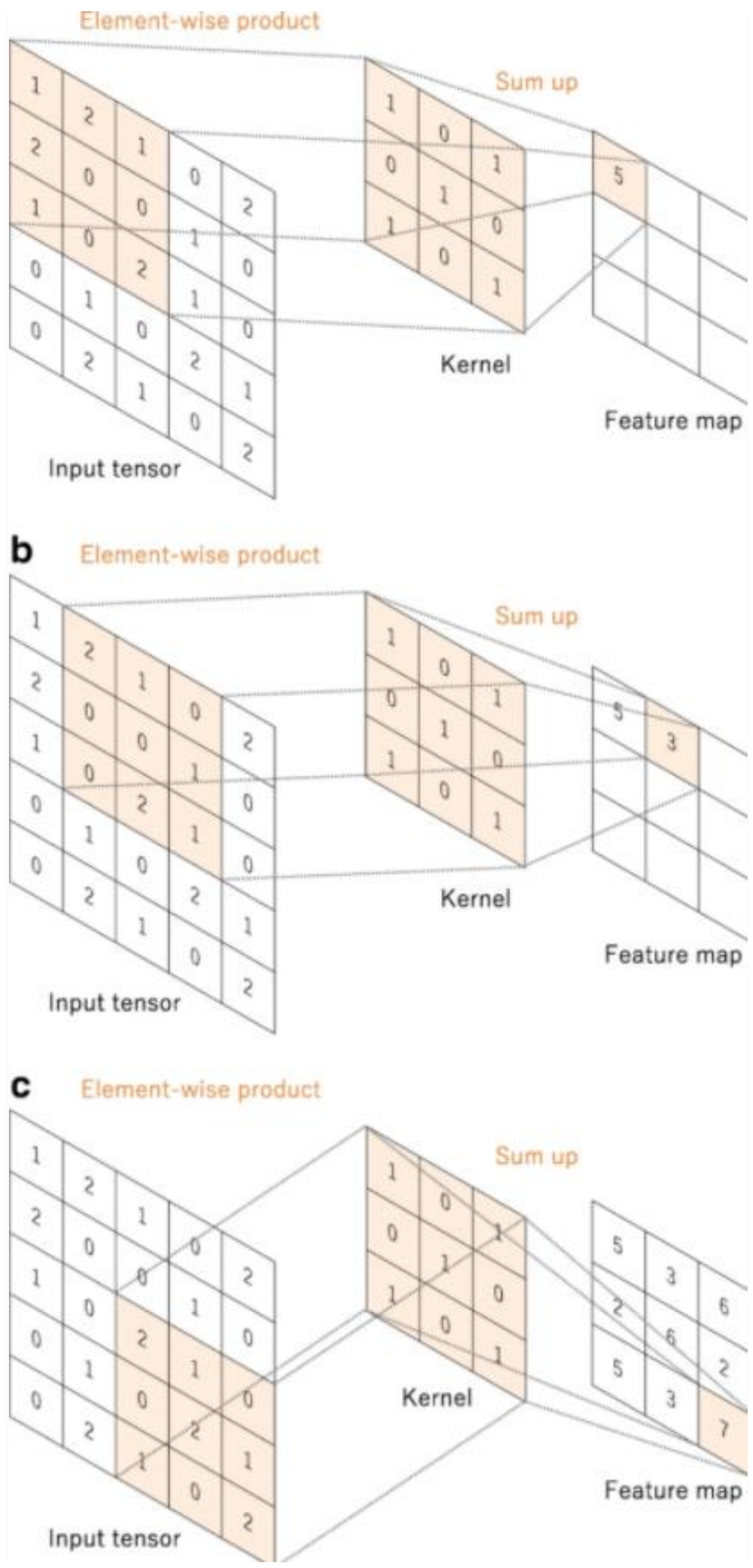
The CNN architecture includes several building blocks, such as convolution layers, pooling layers, and fully connected layers. A typical architecture consists of repetitions of a stack of several convolution layers and a pooling layer, followed by one or more fully connected layers. The step where input data are transformed into output through these layers is called forward propagation.

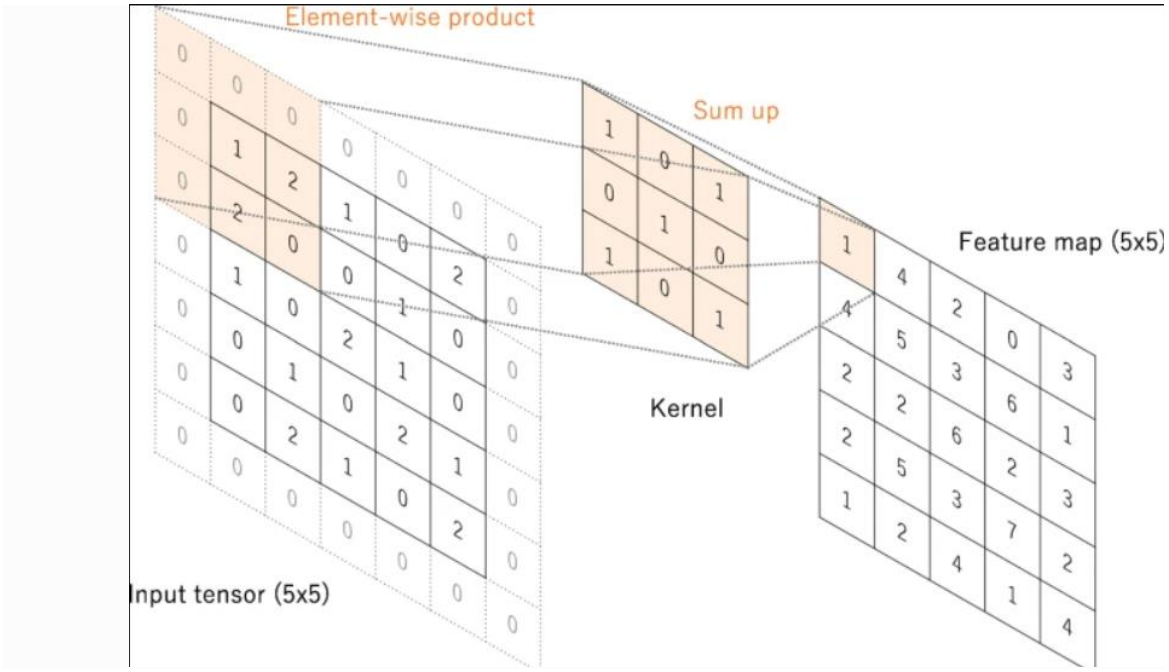
a. Convolutional Layer:

i. A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function.

ii. Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a tensor. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a feature map. This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors. Two key hyper parameters that define the convolution operation are size and number of kernels. The former is typically 3×3, but sometimes 5×5 or 7×7. The latter is arbitrary, and determines the depth of output feature maps.

The convolution operation does not allow the center of each kernel to overlap the outermost element of the input tensor, and reduces the height and width of the output feature map compared to the input tensor. Padding, typically zero padding, is a technique to address this issue, where rows and columns of zeros are added on each side of the input tensor, so as to fit the center of a kernel on the outermost element and keep the same in-plane dimension through the convolution operation. Modern CNN architectures usually employ zero padding to retain in-plane dimensions in order to apply more layers. Without zero padding, each successive feature map would get smaller after the convolution operation.


Element-wise product
Sum up
Kernel
Feature map
Input tensor
b
Element-wise product
Sum up
Kernel
Feature map
Input tensor
c
Element-wise product
Sum up
Kernel
Feature map
Input tensor



The distance between two successive kernel positions is called a stride, which also defines the convolution operation. The common choice of a stride is 1; however, a stride larger than 1 is sometimes used in order to achieve down sampling of the feature maps.

The key feature of a convolution operation is weight sharing: kernels are shared across all the image positions. Weight sharing creates the following characteristics of convolution operations: letting the local feature patterns extracted by kernels translation be invariant as kernels travel across all the image positions and detect learned local patterns, learning spatial hierarchies of feature patterns by down sampling in conjunction with a pooling operation, resulting in capturing an increasingly larger field of view, and increasing model efficiency by reducing the number of parameters to learn in comparison with fully connected neural networks.

b. Pooling Layer:

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarizing the features lying within the region covered by filter. For a feature map having dimensions n_h **x** n_w **x** n_c, the dimensions of output obtained after a pooling layer is

$$(n_h - f + 1) / s \times (n_w - f + 1)/s \times n_c$$

where,

n_h **-** height of feature map
n_w **-** width of feature map
n_c **-** number of channels in the feature map
f - size of filter
s - stride length

A common CNN model architecture is to have a number of convolution and pooling layers stacked one after the other. Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

- **Types of Pooling Layers:**

i. Max Pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

ii. Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

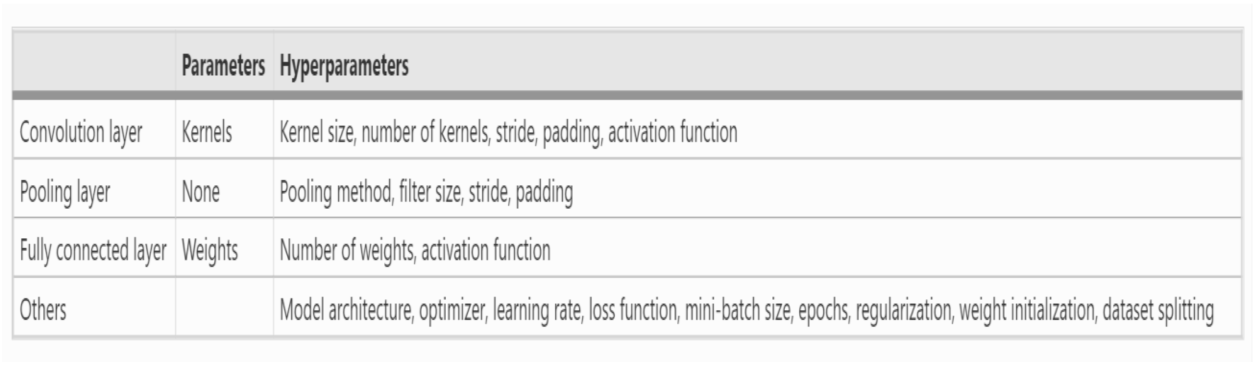
iii. Global Pooling

Global pooling reduces each channel in the feature map to a single value. Thus,
a $n_h \times n_w \times n_c$ feature map is reduced to $1 \times 1 \times n_c$ feature map. This is equivalent to using a filter of dimensions $n_h \times n_w$ i.e., the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

c. <u>Fully Connected Layer:</u>

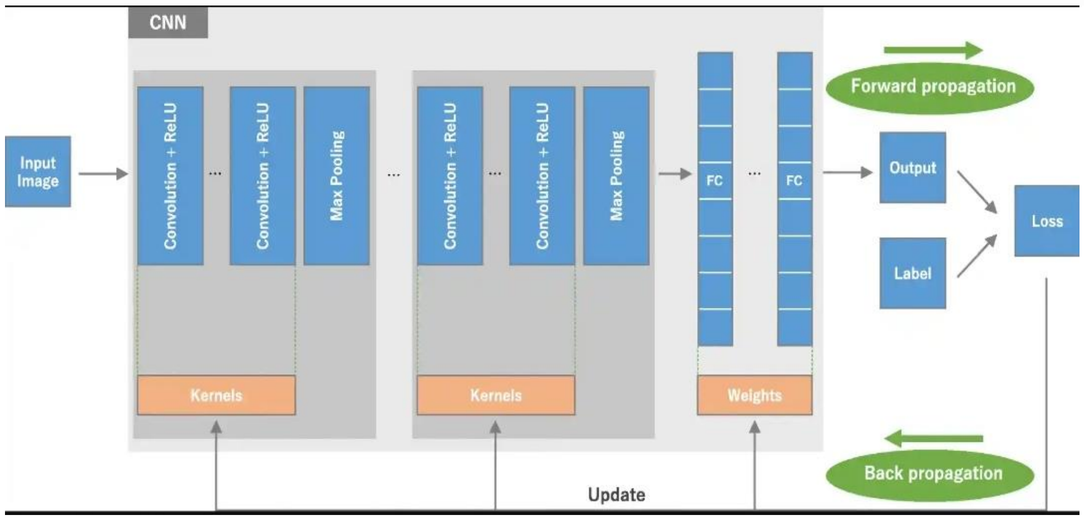
Fully connected layers are an essential component of Convolutional Neural Networks (CNNs), which have been proven very successful in recognizing and classifying images for computer vision.

The CNN process begins with convolution and pooling, breaking down the image into features, and analyzing them independently. The result of this process feeds into a fully connected neural network structure that drives the final classification decision.



	Parameters	Hyperparameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding, activation function
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function
Others		Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regularization, weight initialization, dataset splitting

The process of training a CNN model with regard to the convolution layer is to identify the kernels that work best for a given task based on a given training dataset. Kernels are the only parameters automatically learned during the training process in the convolution layer; on the other hand, the size of the kernels, number of kernels, padding, and stride are hyper parameters that need to be set before the training process starts.



❖ ARCHITECTURES

- **AlexNet:**

A large deep convolutional neural network was trained to classify the 1.2 million high resolution images in the ImageNet ILSVRC-2012 contest into the 1000 different classes. On the test data top-1 and top-5 error rates of 37.5% and 17.0% were achieved which is considerably better than the previous state of the art. The neural network has 60 million parameters and 650,000 neurons and it consists of five convolutional layers, some of which are followed by max pooling layers, and three fully connected layers with a final 1000-way SoftMax. Non-saturating neurons and a very efficient GPU implementation of the convolution operation is used to make the training faster.**[3]**

The dataset used is ImageNet which is made of more than 15 million high resolution images, labeled with 20 thousand classes. There are about 1.2 million training images,50 thousand validation images and 150 thousand testing images.**[4]**

The network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution. The network contains eight layers with weights; the first five are convolutional and the remaining three are fully connected.

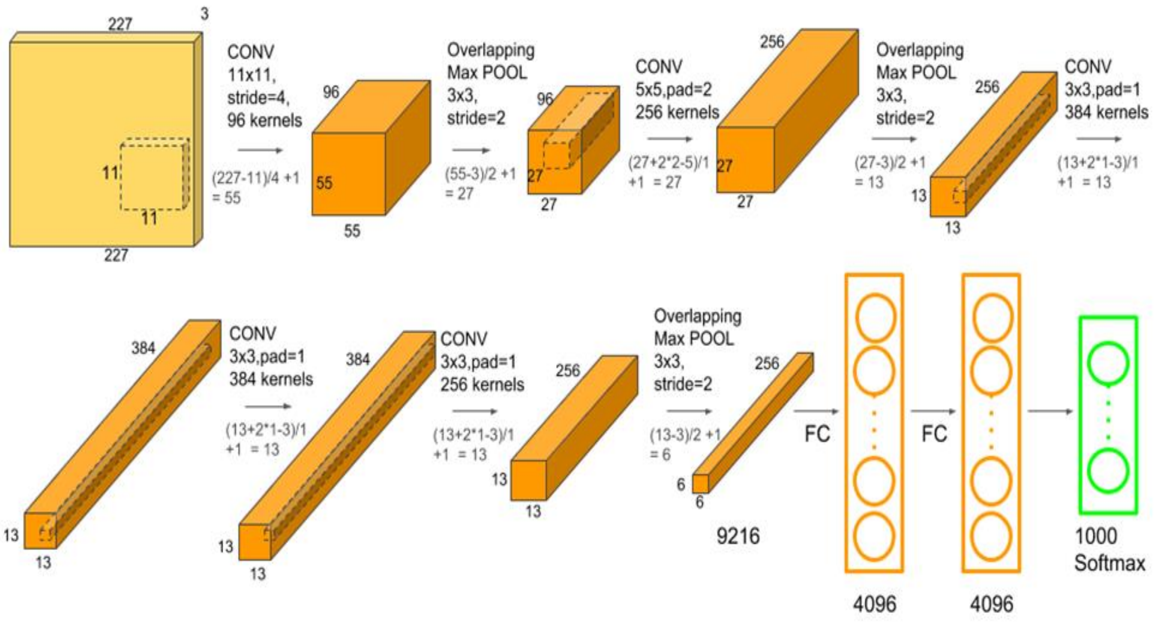
The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU. The kernels of the third convolutional layer are connected to all kernel maps in the second layer.
The neurons in the fully connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers.
Max-pooling layers follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

• A Walkthrough AlexNet architecture:

AlexNet was classified as a deep convolutional network and it was built to accommodate colored images of size (224x224x3). It boasted a total of over 62 million trainable parameters.**[5]**

The 11 layers of AlexNet were:

a. **Layer C1:** Convolution Layer (96, 11×11)
b. **Layer S2:** Max Pooling Layer (3×3)
c. **Layer C3:** Convolution Layer (256, 5×5)
d. **Layer S4:** Max Pooling Layer (3×3)
e. **Layer C5:** Convolution Layer (384, 3×3)
f. **Layer C6:** Convolution Layer (384, 3×3)
g. **Layer C7:** Convolution Layer (256, 3×3)
h. **Layer S8:** Max Pooling Layer (3x3)
i. **Layer F9:** Fully Connected Layer (4096)
j. **Layer F10:** Fully Connected Layer (4096)
k. **Layer F11:** Fully Connected Layer (1000)


227
3
11
11
227
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

C1: First Convolution Layer:

The first layer of AlexNet was a convolutional layer that accepted a (224×224×3) image tensor as its input. It performed a convolution operation using 96 (11×11) kernels with a stride of four and a padding of two. This produced a (55×55×96) output tensor that was then passed through a ReLU activation function then on to the next layer. The layer contained 34,944 trainable parameters.

S2: First Max Pooling Layer:

The second layer of AlexNet was a max-pooling layer that accepted output from the layer C1, a (55×55×96) tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two. This produced a (27×27×96) output tensor that was passed on to the next layer.

C3: Second Convolution Layer:

The third layer of AlexNet was another convolutional layer that accepted output from the layer S2, a (27×27×96) tensor, as its input. It performed a convolution operation using 256 (5×5) kernels with a stride of one, and a padding of two. This operation produced a (27×27×256) output tensor that was then passed through a ReLU activation function, and then on to the next layer. The layer contained 614,656 trainable parameters, which summed to a total of 649,600 trainable parameters so far.

S4: Second Max Pooling Layer:

The fourth layer of AlexNet was a max-pooling layer that accepted output from the layer C3, a (27×27×256) tensor, as its input. It performed a zero-padded sub-sampling operation using a (3×3) kernel with a stride of two, similar to layer S2. This produced a (13×13×256) output tensor that was then passed through a ReLU activation function, and then on to the next layer.

C5: Third Convolution Layer:

The fifth layer of AlexNet was another convolutional layer that accepted output from the layer C5, a (13×13×256) tensor, as its input. It performed a convolution operation using 384 (3×3) kernels with a stride and padding of one. This produced a (13×13×384) output tensor that was then passed through a ReLU activation function, and then on to the next layer. The layer contained 885,120 trainable parameters, which summed to a total of 1,534,720 trainable parameters so far.

C6: Fourth Convolution Layer:

The sixth layer of AlexNet was another convolutional layer that accepted output from the layer C5, a (13×13×384) tensor, as its input. It performed the same convolution operation as layer C5, which lead to the same output size. The output was also passed through a ReLU activation function. The layer contained 1,327,488 trainable parameters, which summed to a total of 2,862,208 trainable parameters so far.

C7: Fifth Convolution Layer:

The seventh layer of AlexNet was another convolutional layer that accepted a (13×13×384) tensor from the layer C6 as its input. It performed a convolution operation using 256 (3×3) kernels with a stride and padding of 1. This produced a (13×13×256) output tensor. The output was also passed through a ReLU activation function. The layer contained 884,992 trainable parameters, which summed to a total of 3,747,200 trainable parameters so far.

S8: Third Max Pooling Layer:

The eighth layer of AlexNet was a max pooling layer that accepted a (13×13×256) tensor from the layer C7 as its input. It performed a zero-padded sub-sampling operation using a (3×3) window region with a stride of two, similar to layer S2 and S4. This produced a (6×6×256) output tensor that was then passed through a ReLU activation function, and then on to the next layer.

F9: First Fully Connected Layer:

The ninth layer of AlexNet was a fully connected layer that accepted a flattened (6×6×256) tensor from the layer S8 as its input. It performed a weighted sum operation with an added bias term. This produced a (4096×1) output tensor that was then passed through a ReLU activation function, and on to the next layer. The layer contained 37,752,832 trainable parameters, which summed to a total of 41,500,032 trainable parameters so far.

F10: Second Fully Connected Layer:

The tenth layer of AlexNet was another fully connected layer that accepted a (4096×1) tensor from the layer F9 as its input. It performed the same operation as layer F9 and produced the same (4096×1) output tensor that was then passed through a ReLU activation function, and then on to the next layer. The layer contained 16,781,312 trainable parameters, which summed to a total of 58,281,344 trainable parameters so far.

F11: Third Fully Connected Layer:

The eleventh and final layer of the network was also a fully connected layer that accepted a (4096×1) tensor from the layer F10 as its input. It performed the same operation as layer F9 and F10 and produced a (1000×1) output tensor that was then passed through a SoftMax activation function. The layer contained 4,097,000 trainable parameters, which summed to a total of 62,378,344 trainable parameters overall. The output of the SoftMax activation function contained the predictions of the network.

AlexNet architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting.
The two primary methods of overcoming the problems of overfitting are described below:

- ### Data Augmentation:

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations. Two distinct forms of data augmentation are employed, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224x224 patches (and their horizontal reflections) from the 256x256 images and training our network on these extracted patches4. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224x224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's SoftMax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, with magnitudes proportional to the corresponding eigen values times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1.

- **<u>Dropout:</u>**

Dropout consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation.

This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

A large, deep convolutional neural network is capable of achieving record-breaking results on a highly challenging dataset using purely supervised learning. It is notable that this network’s performance degrades if a single convolutional layer is removed. For example, removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So, the depth really is important for achieving better result.

- **VGG-16:**

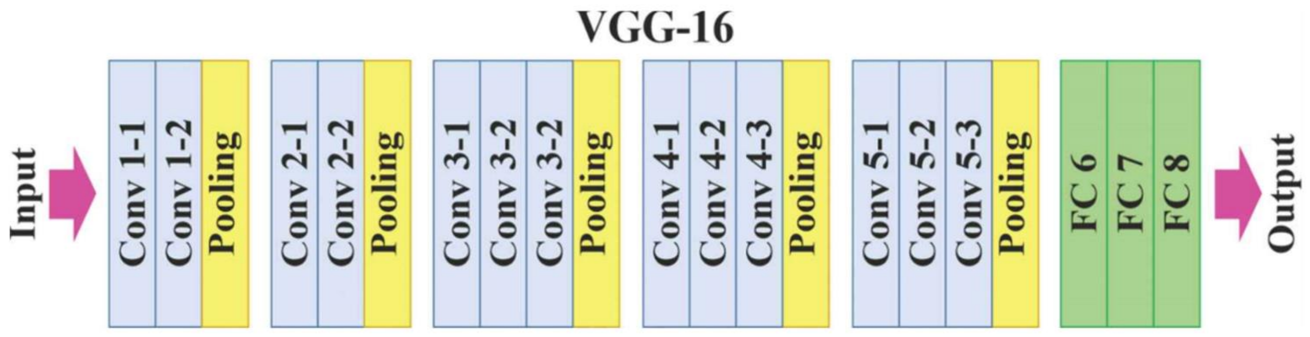
VGG is an acronym for the Visual Geometric Group from Oxford University and VGG-16 is a network with 16 layers proposed by the Visual Geometric Group. These 16 layers contain the trainable parameters and there are other layers also like the Max pooling layer but those do not contain any trainable parameters. This architecture was the 1st runner up of the Visual Recognition Challenge of 2014 i.e., ILSVRC-2014.

The VGG research group released a series of the convolution network model starting from VGG11 to VGG19.The main intention of the VGG group on depth was to understand how the depth of convolutional networks affects the accuracy of the models of large-scale image classification and recognition. The minimum VGG11 has 8 convolutional layers and 3 fully connected layers as compared to the maximum VGG19 which has 16 convolutional layers and the 3 fully connected layers.

The different variations of VGGs are exactly the same in the last three fully connected layers. The overall structure includes 5 sets of convolutional layers, followed by a Max-Pooling Layer. But the difference is that as the depth increases, that is, as we move from VGG11 to VGG19 more and more cascaded convolutional layers are added in the five sets of convolutional layers.

There are 13 convolution layers along with the non-linear activation function which is a rectified linear unit or (ReLU), and five MaxPooling layers. Along with these, there are three fully connected layers. At the output, we have a SoftMax layer having thousand outputs per image category in the ImageNet dataset. This architecture begins with a very low channel size of 64 and then gradually increased by a factor of two after each MaxPooling layer, until it reaches 512. The architecture is very simple.

It has got two contiguous blocks of two convolution layers followed by MaxPooling, then it has three contiguous blocks of three convolution layers followed by MaxPooling and at last we have three dense layers. The important thing to analyze here is that after every MaxPooling the size is getting reduced by half.



- **A Walkthrough VGG-16 Architecture:**

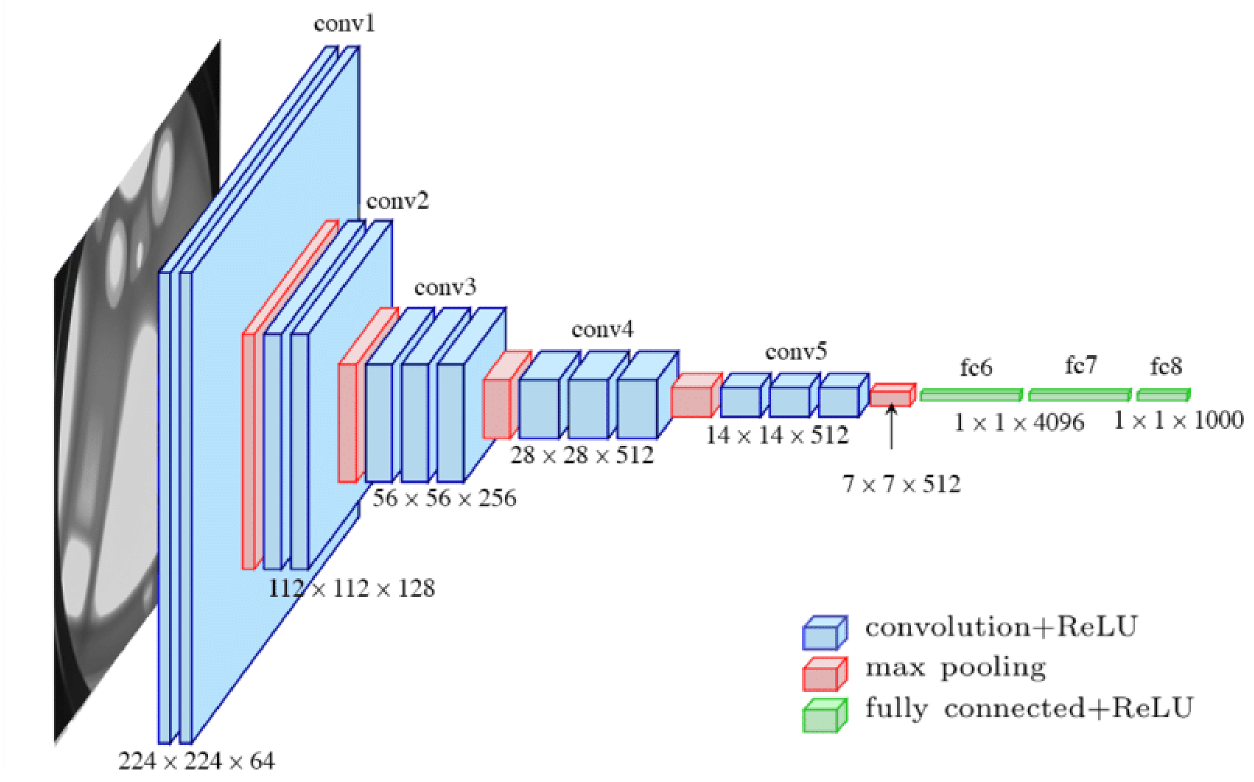
a. **Input Layer:** It accepts a color image as an input with the size 224 x 224 and 3 channels i.e., Red, Green, and Blue.

b. **Convolution Layer:** The image passes through a stack of convolution layers where every convolution filter has a very small receptive field of 3 x 3 and stride of 1. Every convolution kernel uses row and column padding so that the size of input as well as the output feature maps remains the same or in other words, the resolution after the convolution is performed remains the same.

c. **Max Pooling Layer:** It is performed over a max-pool window of size 2 x 2 with stride equals to 2, which means here max pool windows are non-overlapping windows. Not every convolution layer is followed by a max pooling layer as at some places a convolution layer is following another convolution layer without the max-pooling layer in between. **[6]**

d. **Fully Connected Layer:** The first two fully connected layers have 4096 channels each and the third fully connected layer which is also the output layer has 1000 channels, one for each category of images in the ImageNet database.

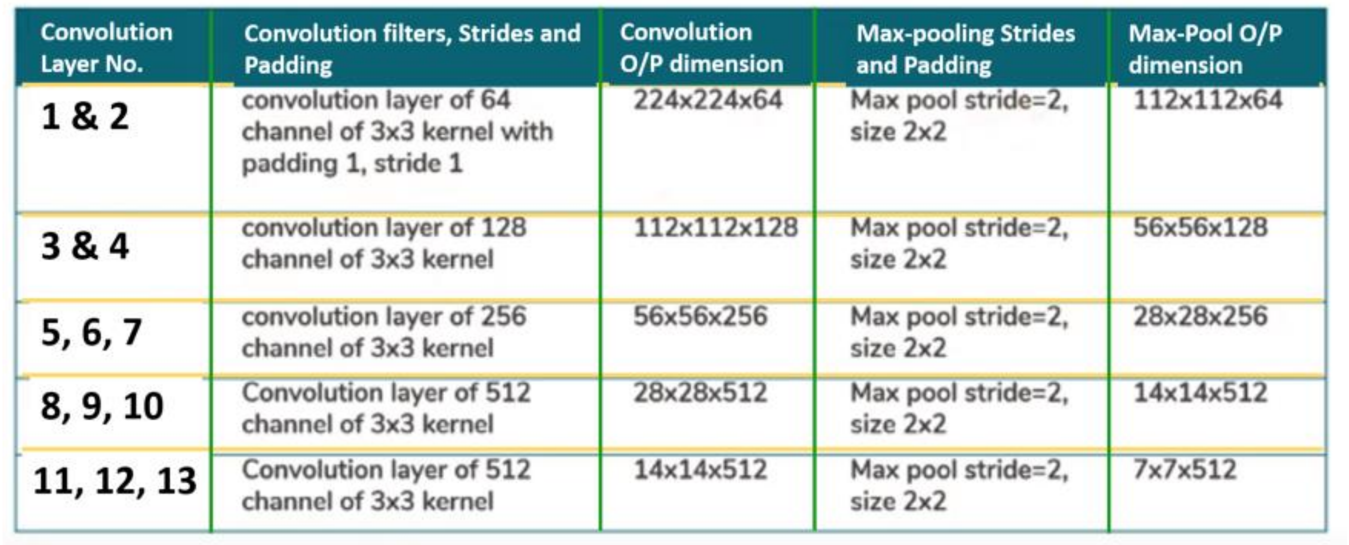
e. **Hidden Layer:** They have ReLU as their activation function.



VGG-16 Network Architectu

As we know more the layers of convolution, more sharply the features will be extracted from our input as compared to when we have fewer layers. So, having 3 x 3 kernel size would lead to much better feature extraction than 7 x 7 kernel size. When we take 3 x 3 kernel size the number of trainable parameters will be 27K² as compared to 7 x 7 kernel size when taken gives 49K² trainable parameters which is 81% more.

The complete architecture of the VGG-16 has been summed up in the table shown below:



Convolution Layer No.	Convolution filters, Strides and Padding	Convolution O/P dimension	Max-pooling Strides and Padding	Max-Pool O/P dimension
1 & 2	convolution layer of 64 channel of 3x3 kernel with padding 1, stride 1	224x224x64	Max pool stride=2, size 2x2	112x112x64
3 & 4	convolution layer of 128 channel of 3x3 kernel	112x112x128	Max pool stride=2, size 2x2	56x56x128
5, 6, 7	convolution layer of 256 channel of 3x3 kernel	56x56x256	Max pool stride=2, size 2x2	28x28x256
8, 9, 10	Convolution layer of 512 channel of 3x3 kernel	28x28x512	Max pool stride=2, size 2x2	14x14x512
11, 12, 13	Convolution layer of 512 channel of 3x3 kernel	14x14x512	Max pool stride=2, size 2x2	7x7x512

- <u>**Differences Between AlexNet and VGG-16:**</u>

Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative. There are also fewer parameters (27 times the number of channels instead of AlexNet, which has 49 times the number of channels).
VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields. The small-size convolution filters allow VGG to have a large number of weight layers; of course, more layers lead to improved performance.

On a single test scale, VGG achieved a top-1 error of 25.5% and a top-5 error of 8.0%. At multiple test scales, VGG got a top-1 error of 24.8% and a top-5 error of 7.5%. VGG also achieved second place in the 2014 ImageNet competition with its top-5 error of 7.3%, which they decreased to 6.8% after the submission.

- **YOLO:**

YOLO is refreshingly simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

YOLO is extremely fast. Since detection is framed as a regression problem, a complex pipeline is not required. Neural network is simply run on a new image at test time to predict detections. The base network runs at 45 frames per second with no batch processing on a Titan X.GPU and a fast version runs at more than 150 fps. This means streaming of video can be processed in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

Yolo reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. YOLO makes less than half the number of background errors compared to Fast R-CNN.

YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. **[7]** While it can quickly identify objects in images, it struggles to precisely localize some objects, especially small ones.

- **Object Detection in YOLO:**

The separate components of object detection are unified into a single neural network. The network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously.
The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision. The system divides the input image into an S x S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the predicted box is. Formally we define confidence score as

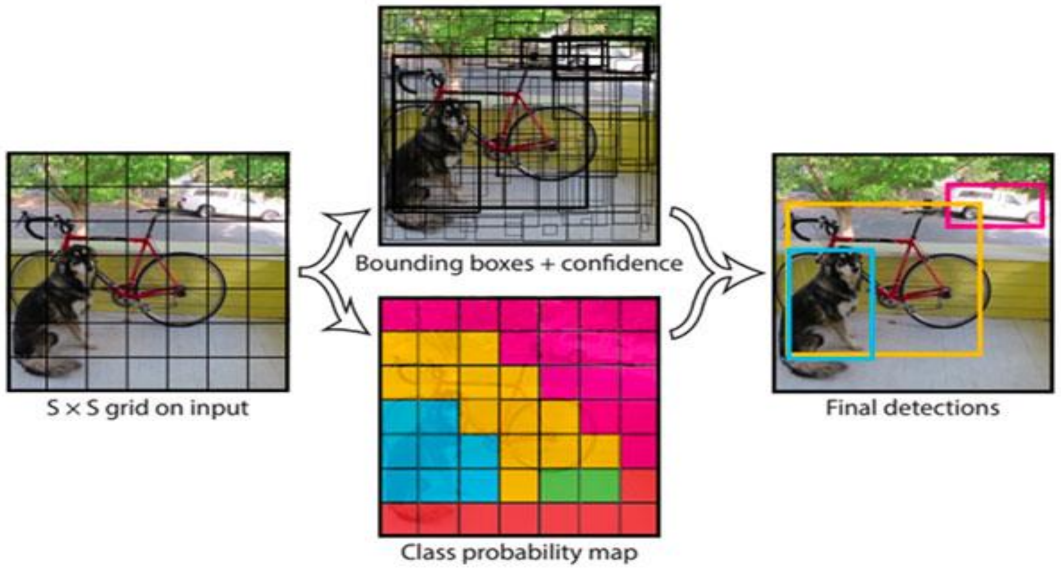
$$\mathbf{P(Object) * IOU^{truth}_{pred}}$$

If no object exists in that cell, the confidence scores should be zero. Each bounding box consists of 5 predictions: x, y, w, h, and confidence score. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell.
The width and height are predicted relative to the whole image. Finally, the confidence score represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, P(Class$_i$IObject). These probabilities are conditioned on the grid cell containing an object. One set of class probabilities is predicted per grid cell, regardless of the number of boxes B.

At test time the conditional class probabilities and the individual box confidence predictions are multiplied.

$$\mathbf{P(Class_i IObject)*P(Object)*IOU^{truth}_{Pred}=P(Class_i)*IOU^{truth}_{pred}}$$

This gives us the class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicated box fits the object.



The YOLO Model

The network architecture is inspired by the Google Net model for image classification. The network has 24 convolutional layers followed by 2 fully connected layers. Simple 1x1 reduction layers are followed by 3x3 convolutional layers. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. The final output of our network is the 7x7x30 tensor of predictions.

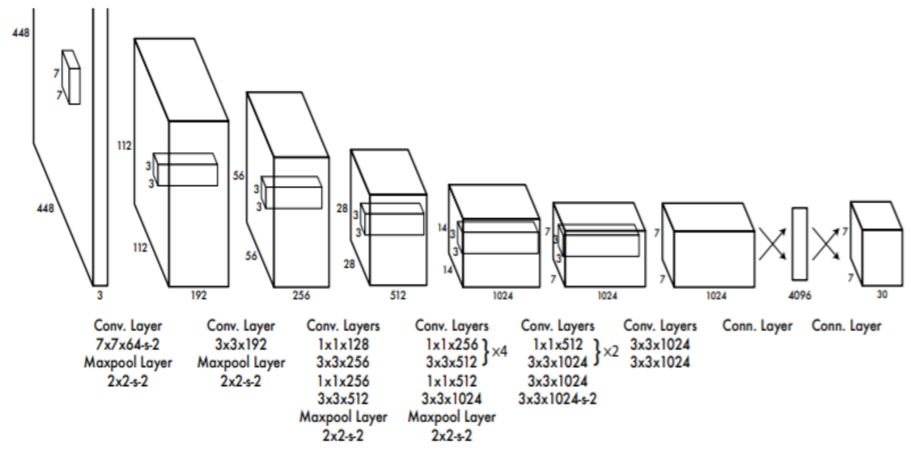
- **A Walkthrough YOLO Architecture:**

Convolutional layers were pretrained on the ImageNet 1000-class competition dataset. For pretraining 20 convolutional layers were used followed by an average-pooling layer and a fully connected layer. The network was trained for approximately a week and a top-5 accuracy of 88% was achieved on the 2012 validation set.

The model was then converted to perform detection. Four convolutional layers and two fully connected layers with randomly initialized weights were added. The input resolution of the network was increased from 224x224 to 448x448 because detection often requires the fine-grained visual information.

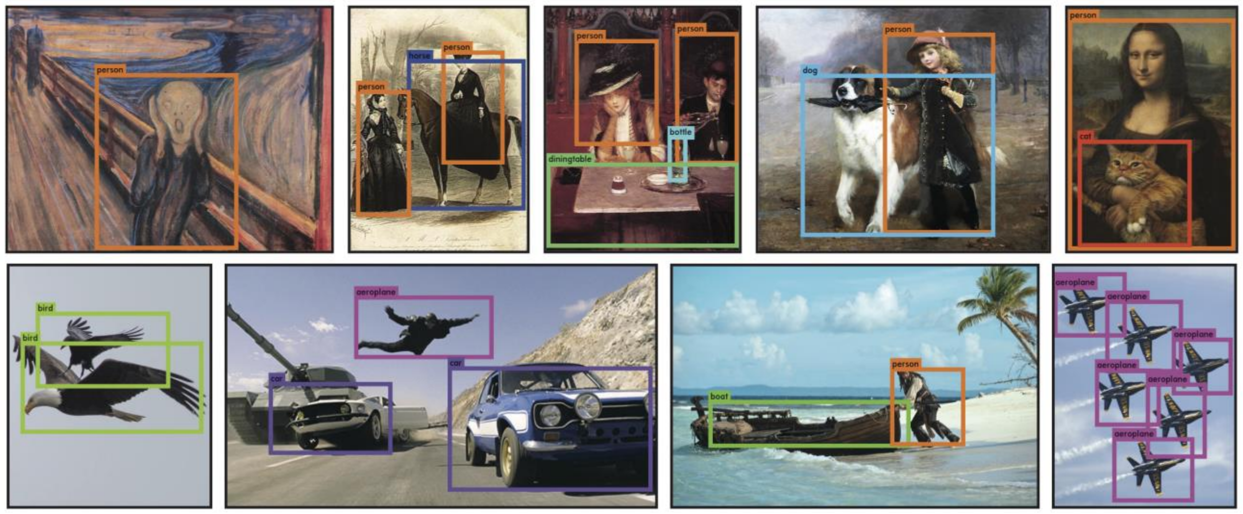
The final layer predicted both class probabilities and bounding box coordinates. The bounding box width and height was normalized by the image width and height so that they fell between 0 and 1. The bounding box x and y coordinates were parameterized to be offsets of a particular grid cell location so they were also bounded between 0 and 1.A linear activation function for the final layer was used and it was followed by using leaky rectified linear activation function

$$\boldsymbol{\varphi(x) = \{x;\ \text{if } x > 0 \\ 0.1x;\ \text{otherwise}\}}$$


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

The detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 x 1convolutional layers reduce the features space from preceding layers. Convolutional layers are pretrained on the ImageNet classification task at half the resolution (224x224 input image) and then double the resolution for detection.

YOLO predicts multiple bounding boxes per grid cell. At training time only one bounding box predictor is wanted to be responsible for each object.
One predictor is assigned which is responsible for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or class of object, improving overall recall.



- **<u>Qualitative Results:</u>**

YOLO is mostly accurate although as clear from the above image, it does predict one person as airplane. During training, a multipart loss function is optimized. The loss function only penalizes classification error if an object is present in that grid cell. It also only penalizes bounding box coordinate error if that predictor is responsible for the ground truth box (i.e., has the highest IOU of any predictor in that grid cell).

The network is trained for about 135 epochs on the training and validation sets from PASCAL VOC 2007 and 2012. Throughout the training, a batch size of 64, a momentum of 0.9 and a decay of 0.0005 is used.

To avoid overfitting, dropout and extensive data augmentation is used. A dropout layer with rate 0.5 after the first connected layer prevents co-adaptation between layers. For data augmentation, random-scaling and translations of up to 20% of the original image size is introduced, and the adjustment of exposure and saturation of image by up to a factor of 1.5 in HSV color space also takes place.

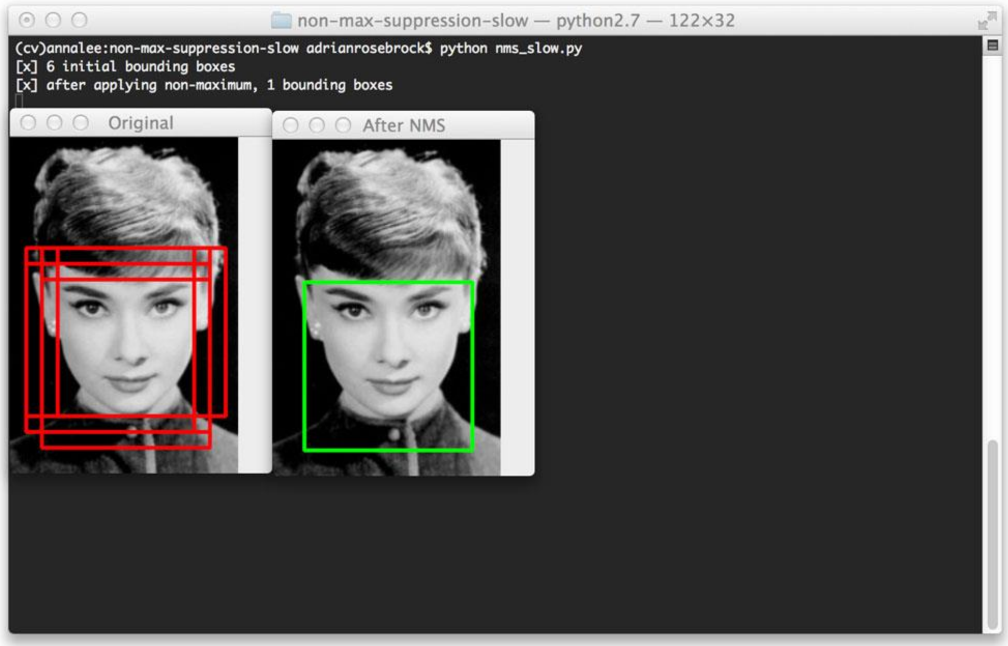
- **<u>Limitations of YOLO:</u>**

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class.
This spatial constraint limits the number of nearby objects that the model can predict. The model struggles with small objects that appear in groups, such as flocks of birds.

Since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. The model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple down sampling layers from the input image.

The model is trained on a loss function that approximates detection performance; the loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. The main source of error is incorrect localizations.

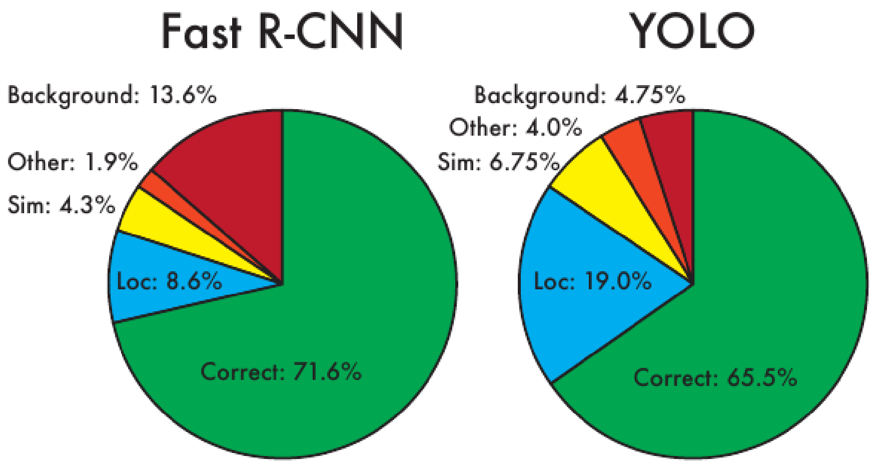
Another limitation of YOLO is the detection of multiple bounding boxes for a single object. Non-Maximum Suppression (NMS) is a technique used in many computer vision algorithms**.** It is a class of algorithms to select one entity (i.e., bounding boxes) out of many overlapping entities. The selection criteria can be chosen to arrive at particular results.



```
(cv)annalee:non-max-suppression-slow adrianrosebrock$ python nms_slow.py
[x] 6 initial bounding boxes
[x] after applying non-maximum, 1 bounding boxes
```

- **<u>Comparison with R-CNN:</u>**

YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of its top detections are false positives that don't contain any objects. Fast R-CNN is almost 3 times more likely to predict background detections than YOLO**.**



Error Analysis: Fast R-CNN vs. YOLO

These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

YOLO model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for various applications.

❖ IMPLEMENTATION:

• TENSORFLOW:

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. It is used to build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging. We've used TensorFlow for easy model building.

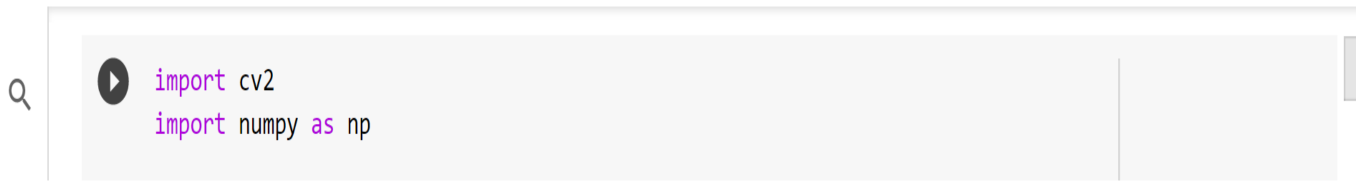
• OPENCV:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. The library is used extensively in companies, research groups and by governmental bodies.

• NUMPY:

NumPy is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be utilized to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines. NumPy is an extension of Numeric and NumArray.



```
import cv2
import numpy as np
```

- **DARKNET:**

Darknet is an open-source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. Once we train the network, we do not need Darknet itself for the inference. OpenCV has built in support for Darknet formats so both model and trained weights are directly usable anywhere where OpenCV is in use, also from Python.

The positive side of this network is that there is somewhat normal documentation on how to train the own data set and how to run the inference on the own input. Other popular frameworks are sometimes so heavily optimized for training and validation against various existing data sets that it gets surprisingly difficult to break out of this golden cage and build a usable product.

```
yolo = cv2.dnn.readNet("model.weights", "darknet-yolov3.cfg")
classes = []
```

- **IPYTHON:**

IPython is a command shell for interactive computing in multiple languages, originally developed for the python programming language, that offers introspection, rich media, shell syntax, tab completion and history.

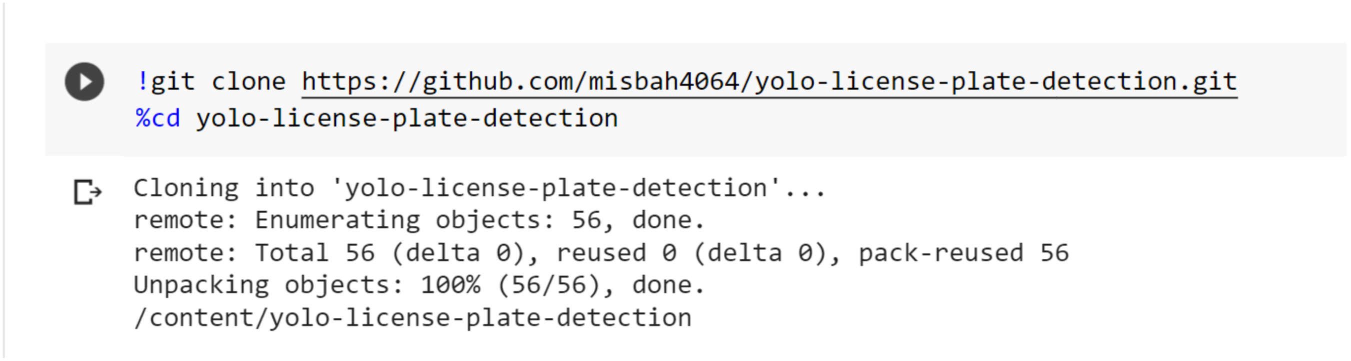
- **MATPLOTLIB:**

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

- **Requirements:**

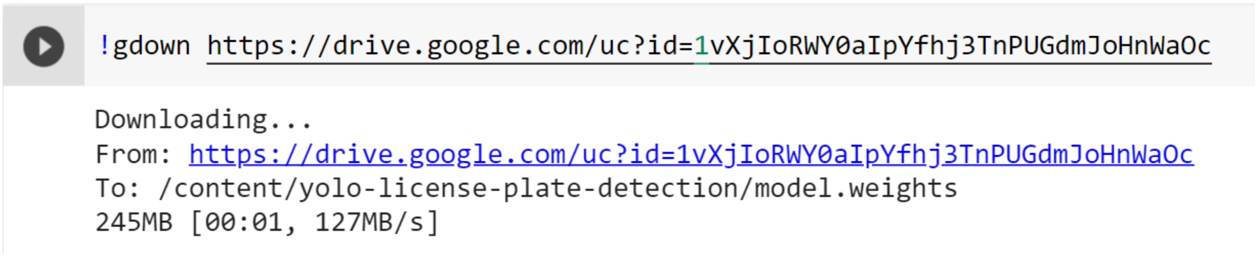
The objective is to detect the location of license plates in cars. The first step to be taken is data gathering and preparation. This involves dataset collection, followed by labelling. The next step is training the dataset which usually takes a few hours, in case the dataset is reasonably large. However, both these steps are not required in our case, since we're cloning a repository from GitHub which contains the configuration file and a class file containing a single class i.e., License Plate. We also change the directory containing these files.



```
!git clone https://github.com/misbah4064/yolo-license-plate-detection.git
%cd yolo-license-plate-detection
```

```
Cloning into 'yolo-license-plate-detection'...
remote: Enumerating objects: 56, done.
remote: Total 56 (delta 0), reused 0 (delta 0), pack-reused 56
Unpacking objects: 100% (56/56), done.
/content/yolo-license-plate-detection
```

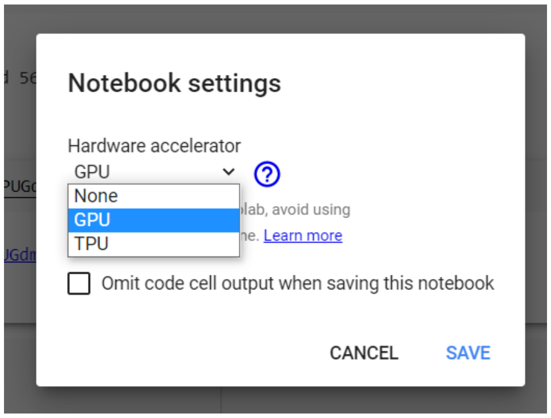
Also, we separately downloaded the pre-trained weight file in order to increase the accuracy and to save time needed for training manually. Since the weight file is very large, it is uploaded separately.



```
!gdown https://drive.google.com/uc?id=1vXjIoRWY0aIpYfhj3TnPUGdmJoHnWaOc
```

```
Downloading...
From: https://drive.google.com/uc?id=1vXjIoRWY0aIpYfhj3TnPUGdmJoHnWaOc
To: /content/yolo-license-plate-detection/model.weights
245MB [00:01, 127MB/s]
```

We can also change runtime type to GPU as it is supposed to increase speed, however while using Google Colab, this factor is of little help.



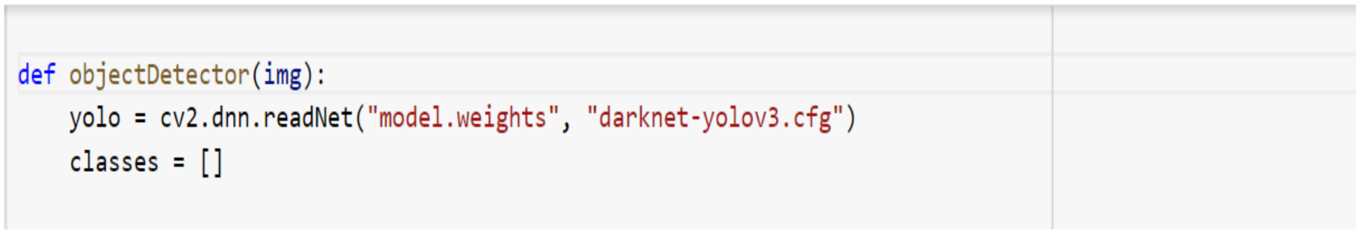
Notebook settings

Hardware accelerator: GPU

☐ Omit code cell output when saving this notebook

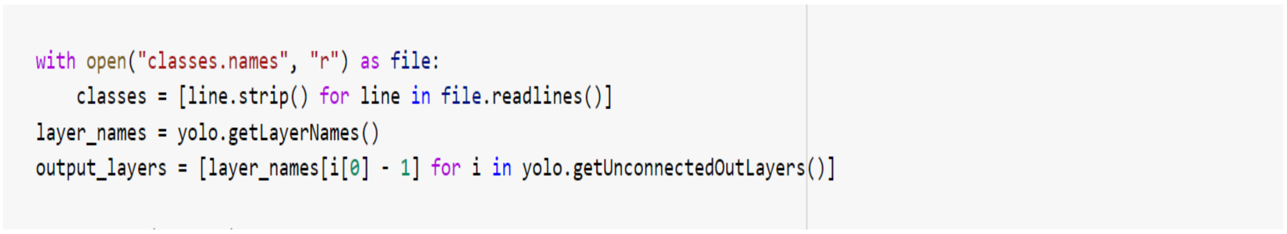
- **Object Detector Function:**

It is a user defined function which takes image as its argument and return image with bounding box around the license plate. Now, we will take a deeper look inside this function. Firstly, we are loading the YOLO network model into memory with cv2.dnn.readNet() function. This function requires both a configuration Path and weights Path. It automatically detects configuration and framework.



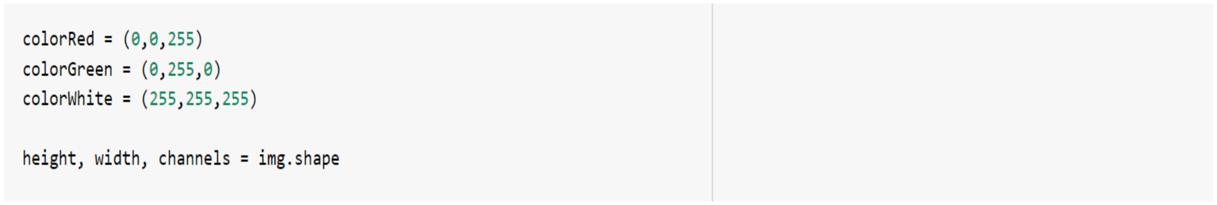
```
def objectDetector(img):
    yolo = cv2.dnn.readNet("model.weights", "darknet-yolov3.cfg")
    classes = []
```

Then we are reading the text file "classes.names" containing class names in human readable form and extracting the class names to the list classes. Generally, in a sequential CNN network there will be only one output layer at the end. In the YOLO v3 architecture that we are using, there are multiple output layers giving out predictions. So, to get these output layer names, we are using "getUnconnectedOutLayers" function.



```
with open("classes.names", "r") as file:
    classes = [line.strip() for line in file.readlines()]
layer_names = yolo.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in yolo.getUnconnectedOutLayers()]
```

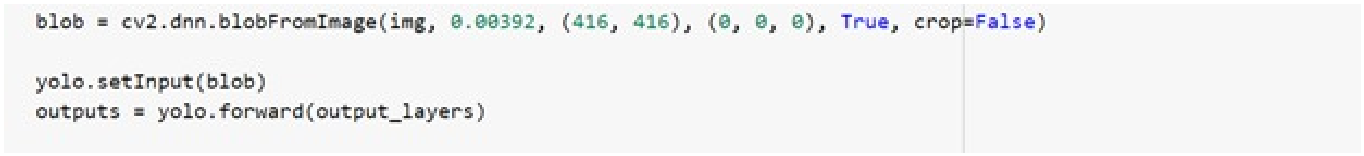
We are generating red, green and white color to use it to draw bounding box later with different color for different classes. We are saving height, width and channels of image so that we can use them to predict the coordinates of bounding box.



```
colorRed = (0,0,255)
colorGreen = (0,255,0)
colorWhite = (255,255,255)

height, width, channels = img.shape
```

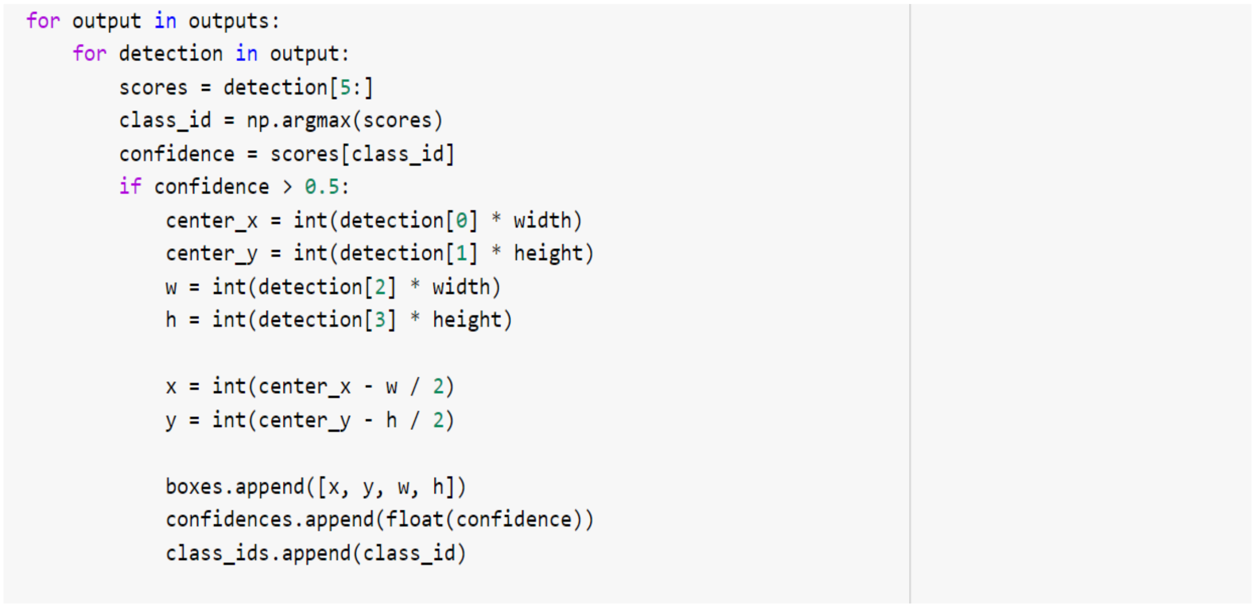
The input to the network is a so-called blob object. Blob is a 4D NumPy array object (images, channels, width, height). The function "cv2.dnn.blobFromImage" transforms the given image into a blob. It has the following parameters:

a. Image to transform (img)
b. Scale factor to scale the pixel value (0.00392)
c. Size (416,416)
d. Mean value(default=0)
e. SwapBR=True (since OpenCV uses BGR)



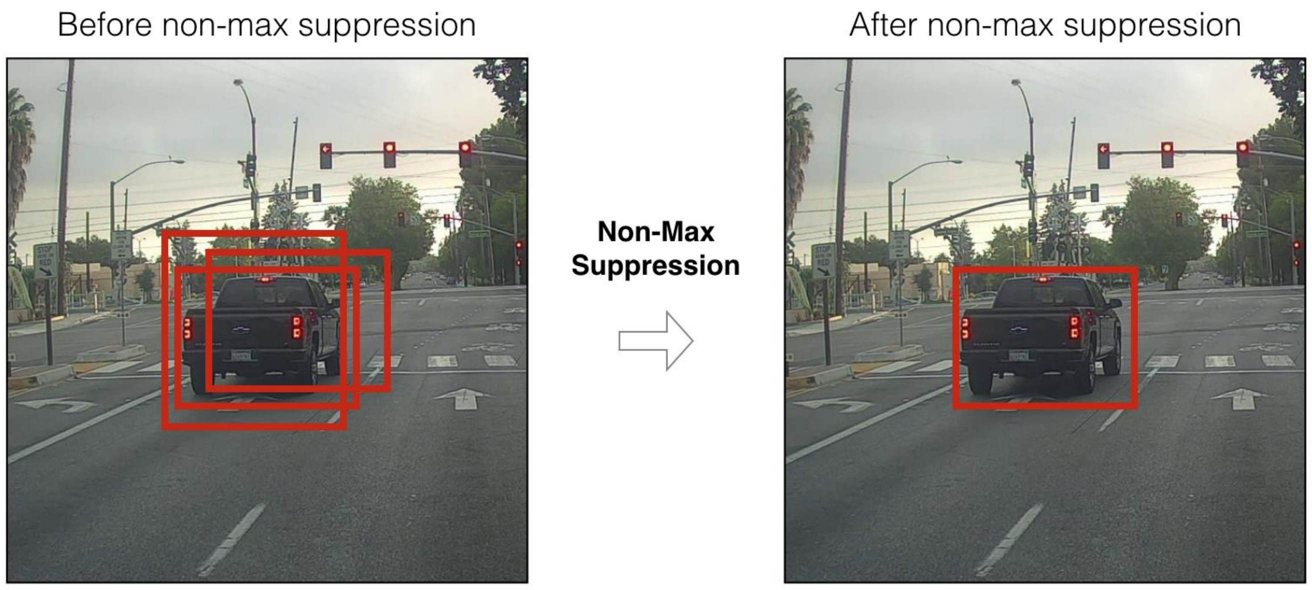
```
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

yolo.setInput(blob)
outputs = yolo.forward(output_layers)
```

SetInput function is preparing the image to run through the network. The exact feed forward through the network happens when we use forward function. If we don't specify the output layer names, by default, it will return the predictions only from final output layer. All intermediate output layers will be ignored.

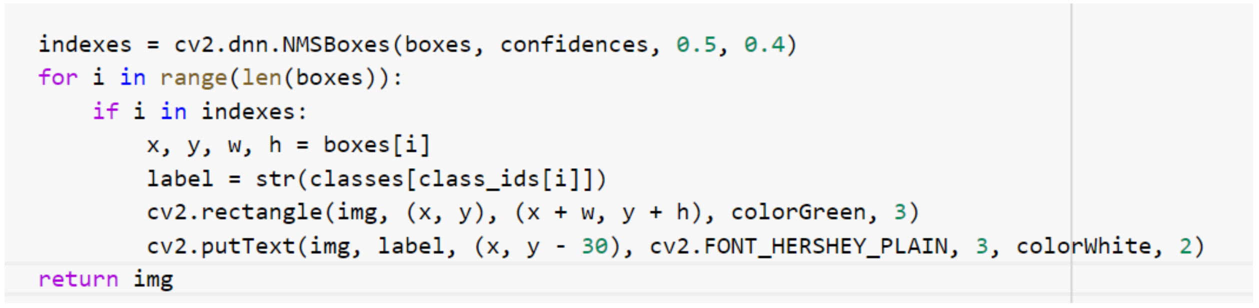


```
for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)
```

We need to go through each detection from each output layer to get the class id, confidence and bounding box corners and most importantly ignore the weak detections (confidence<=0.5). Even though we have ignored weak detections, there will be a lot of duplicate detections with overlapping bounding boxes. Non-maxima suppression removes boxes with overlapping.



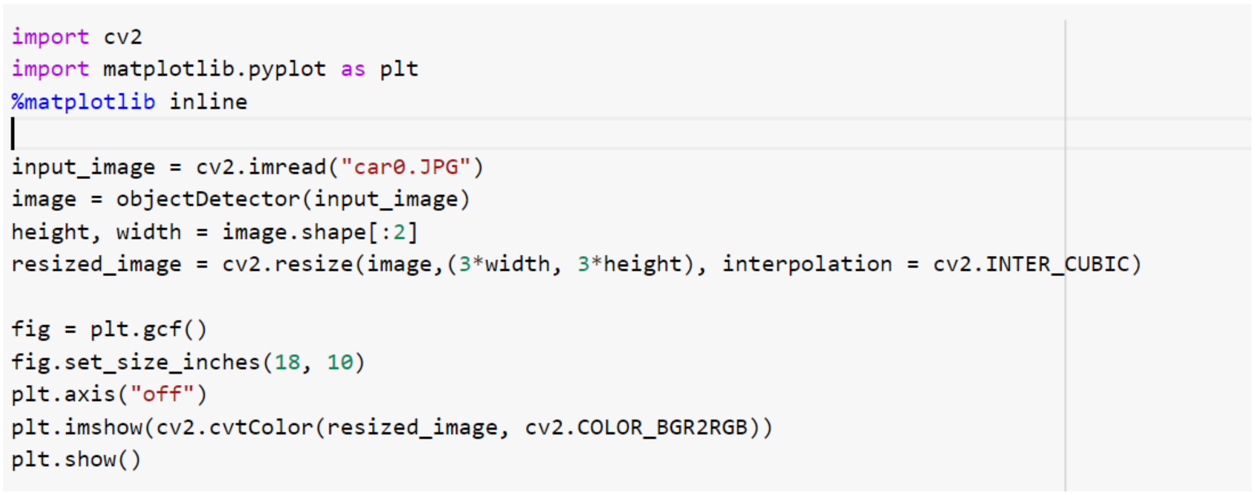
YOLO does not apply non-maxima suppression for us, so we need to explicitly apply it using NMSBoxes function of OpenCV. Applying non-maxima suppression suppresses significantly overlapping bounding boxes, keeping only the most confident ones. It also ensures that we do not have any redundant or extraneous bounding box. To apply this function all we required is we submit our bounding boxes, confidences, as well as both our confidence threshold (0.5) and NMS threshold (0.4). Finally, we are drawing bounding boxes around the license plate detected on image and returning that resulted image from this function.



```
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        cv2.rectangle(img, (x, y), (x + w, y + h), colorGreen, 3)
        cv2.putText(img, label, (x, y - 30), cv2.FONT_HERSHEY_PLAIN, 3, colorWhite, 2)
return img
```

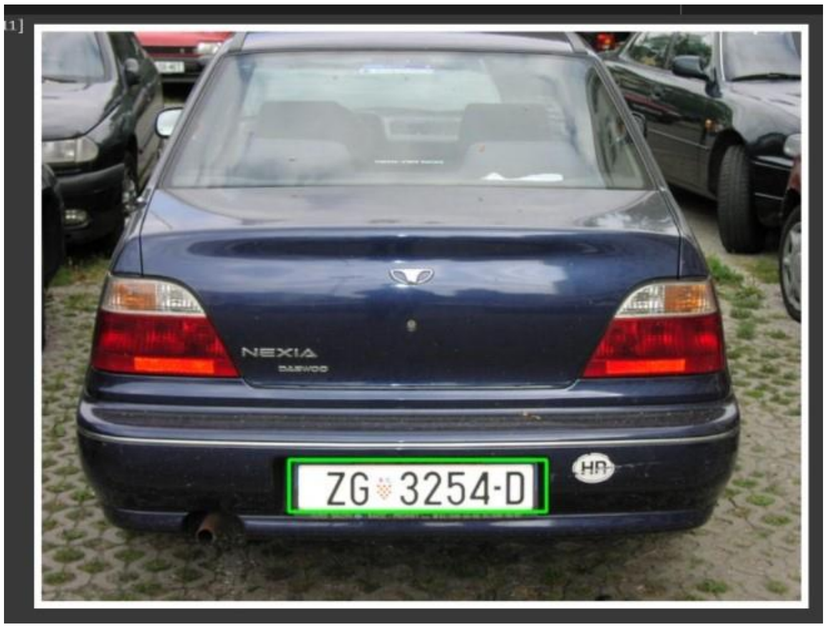
- **Testing on Images:**

Now, we will test images in different cases by giving that image as argument to "ObjectDetector" function which returns image with bounding boxes around license plates detected. then after resizing we are plotting this image to be shown as output. since, Google Colab don't support "imshow" function of Open CV so, we are showing output with the help of matplotlib library "imshow" function. Now, we will see output in different cases and will analyze the results.



```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

input_image = cv2.imread("car0.JPG")
image = objectDetector(input_image)
height, width = image.shape[:2]
resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)

fig = plt.gcf()
fig.set_size_inches(18, 10)
plt.axis("off")
plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
plt.show()
```

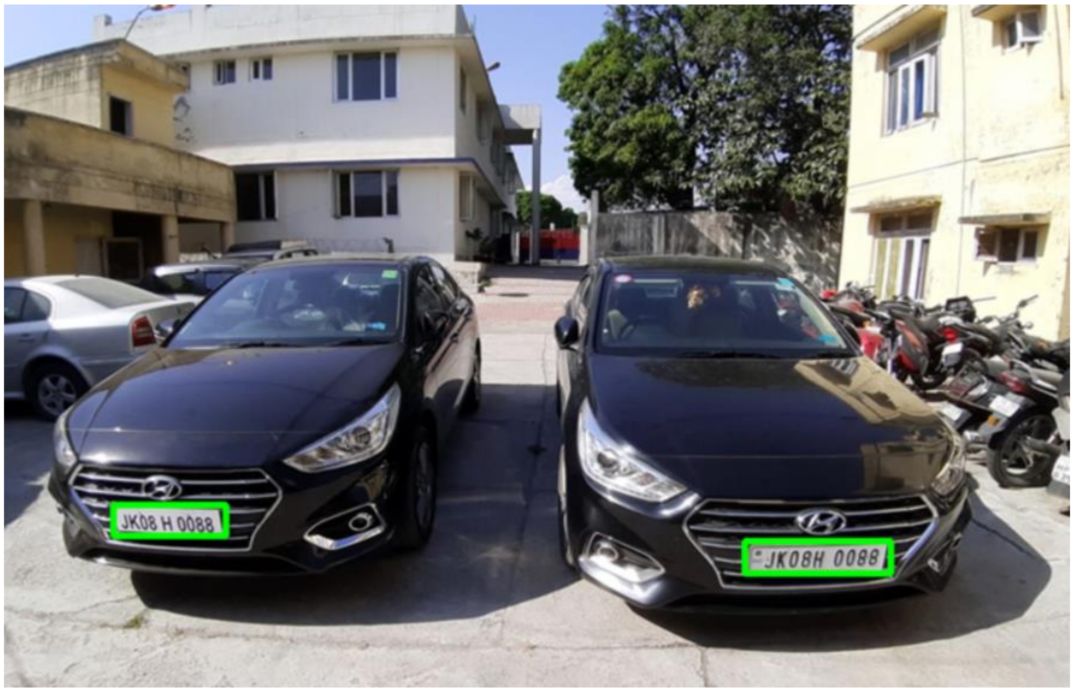
- **CASE I: Single Car**



(a)

(b)

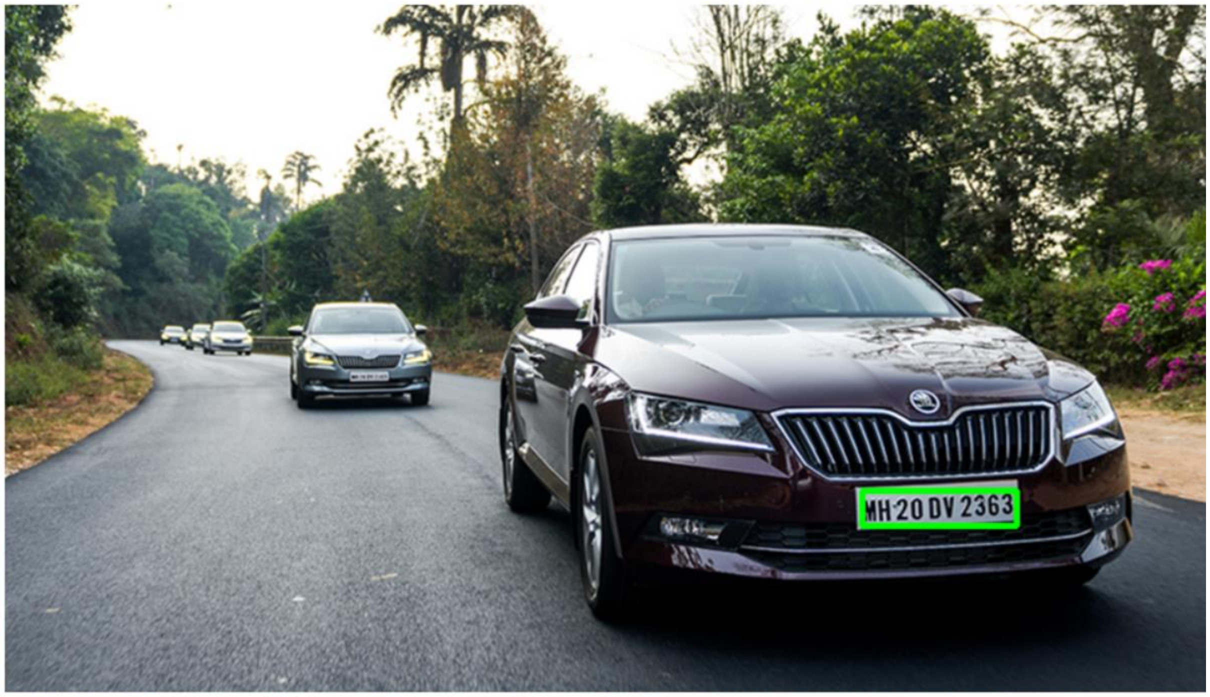
- **CASE II: Two Cars**



(a)

(b)

- **<u>CASE III: Multiple Cars</u>**



- **CASE IV: Blur Image**

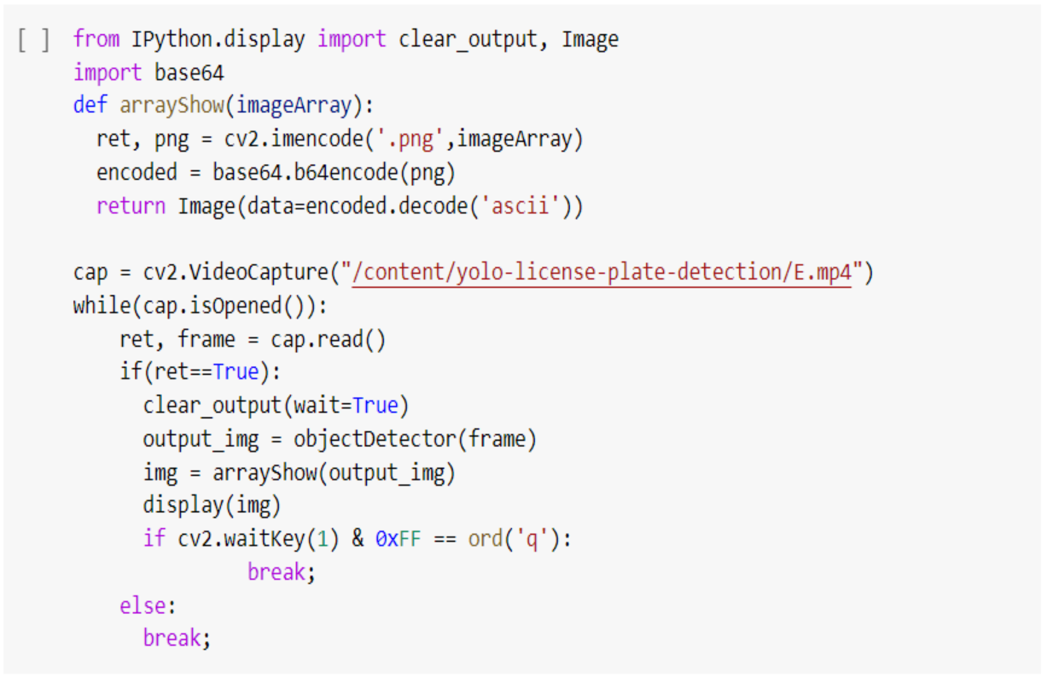
We can clearly see that, license plate is detected in only first, third and fifth image. Also, in the third image, the license plates of the cars behind the Ist one are left undetected.

- **Observations:**

a. Our model behaves inefficiently when car is tilted at an angle and cannot detect license plate in that case.
b. Image needs to be very clear, only then we can use this model otherwise license plate will not be detected as we can see in case IV.

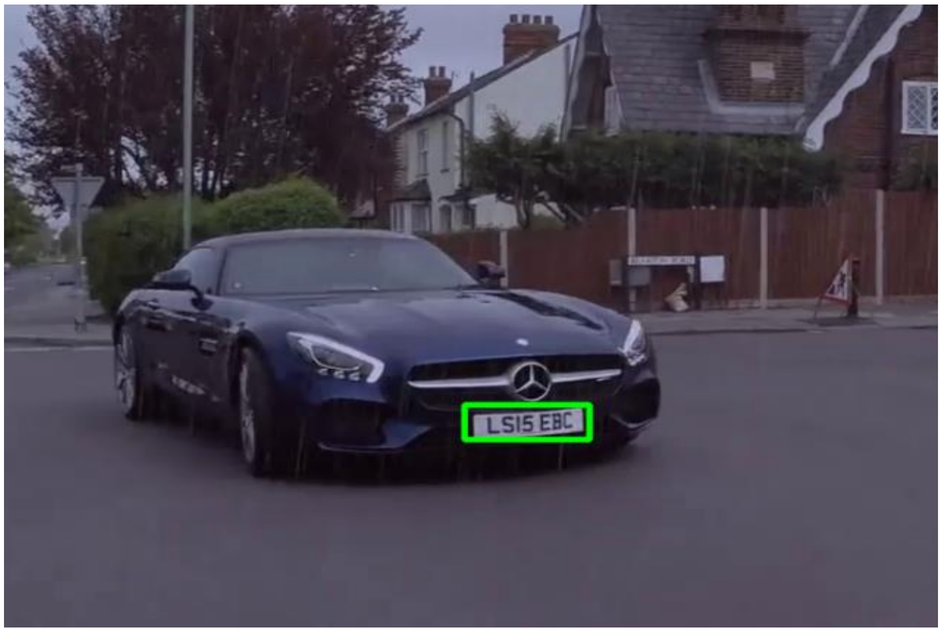
• <u>**Testing on Videos:**</u>

Now, we will test videos containing cars with number plates in different cases by giving a frame as an argument to "ObjectDetector" function which returns frames with bounding boxes around the license plates. When we will plot the output for this video, it will display an array of frames which seems like a video. Now, we will see output in different cases and will analyze the results.

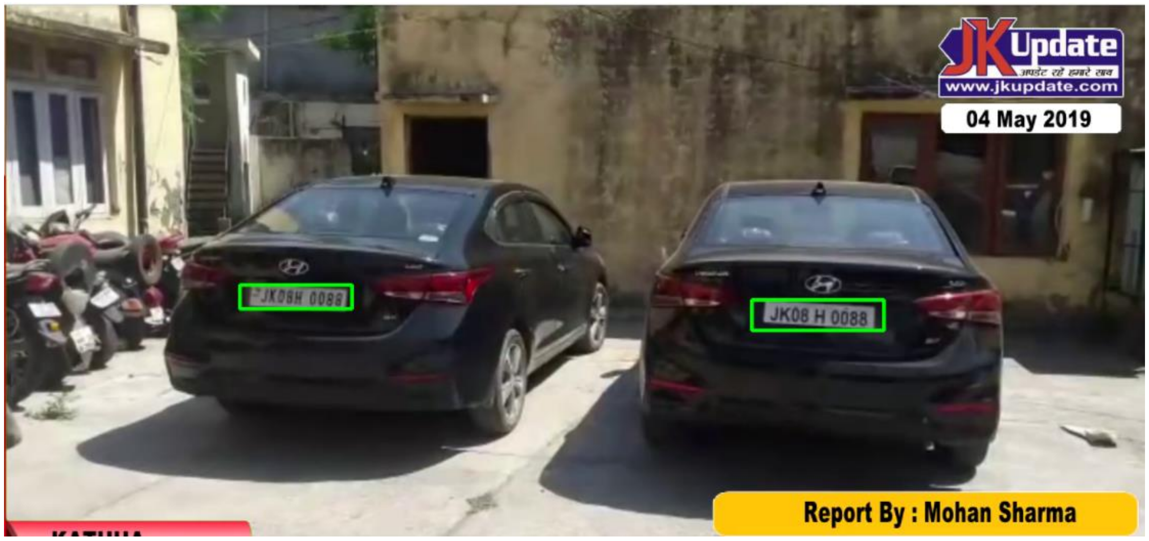


```
[ ] from IPython.display import clear_output, Image
    import base64
    def arrayShow(imageArray):
      ret, png = cv2.imencode('.png',imageArray)
      encoded = base64.b64encode(png)
      return Image(data=encoded.decode('ascii'))

    cap = cv2.VideoCapture("/content/yolo-license-plate-detection/E.mp4")
    while(cap.isOpened()):
        ret, frame = cap.read()
        if(ret==True):
          clear_output(wait=True)
          output_img = objectDetector(frame)
          img = arrayShow(output_img)
          display(img)
          if cv2.waitKey(1) & 0xFF == ord('q'):
                   break;
        else:
          break;
```

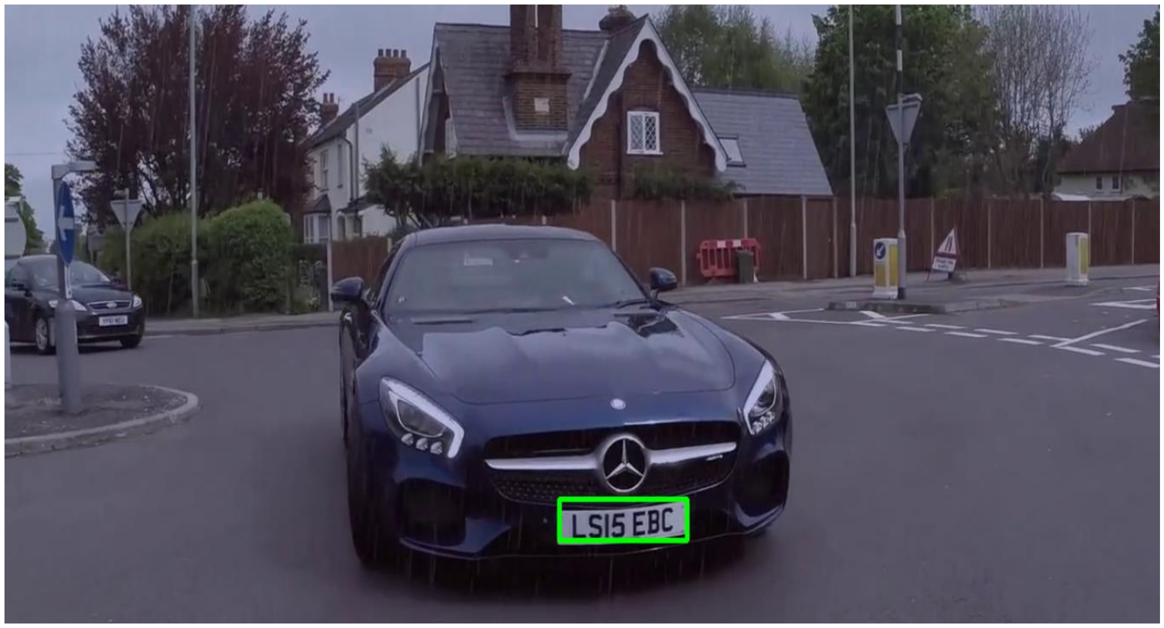
- **<u>CASE I: Single Car</u>**



- **CASE II: Two Cars**

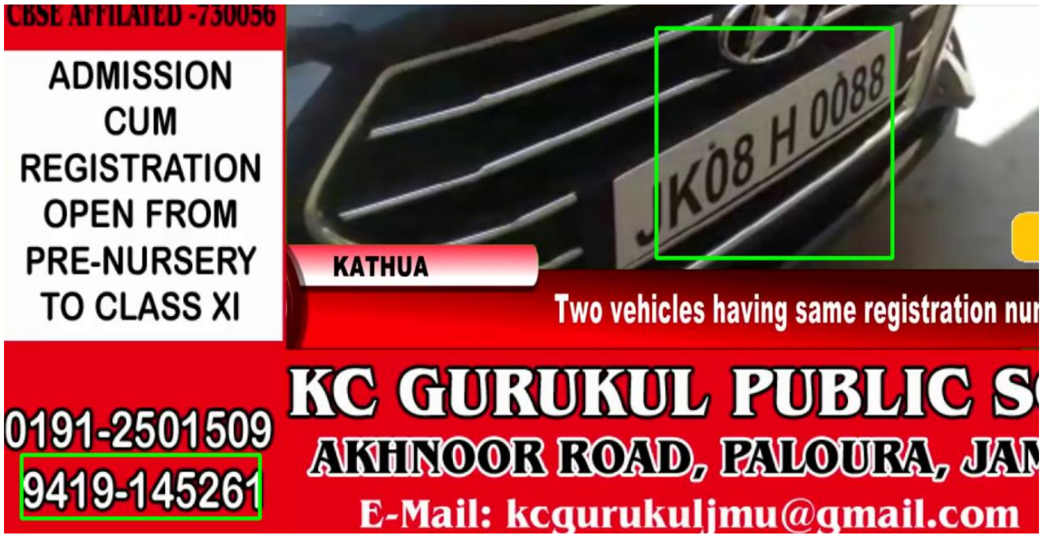


(a)



(b)

- **<u>CASE III: False Positive Detected</u>**


CBSE AFFILATED -730056
ADMISSION CUM REGISTRATION OPEN FROM PRE-NURSERY TO CLASS XI
JK08 H 0088
KATHUA
Two vehicles having same registration nu
KC GURUKUL PUBLIC S
0191-2501509
9419-145261
AKHNOOR ROAD, PALOURA, JA
E-Mail: kcgurukuljmu@gmail.com

- **<u>Observations:</u>**

a. In I and II (a) case, license plates are detected perfectly.
b. However, in case II (b) the car behind the first one is blurred, hence left undetected.
c. In our observation, only one false positive was detected as clear from case III.

❖ CONCLUSION:

This report presents our work on license plate detection systems based on the Yolo architecture.
We used Google Colab platform for our project and from our personal experience we realized that multimedia processing in Google Colab is very slow even after using a GPU. We were able to accomplish the task of license plate detection in images and videos using the Yolo version 3. Evaluation proved that the model was able to detect single as well as multiple license plates in case of both, images and videos. During implementation the video was depicted as an array of frames instead of a continuous video. In each frame, the bounding box successfully detected the plate. At the same time, distorted license plates and license plates with noise were left undetected which showed that the set of blurry and skewed snapshots give worse recognition rates than a set of snapshots which are captured clearly. False positives were detected during the implementation of the model on videos containing objects similar to a license plate. With further optimizations on this model, it can become more feasible and accurate to be used on surveillance videos of crossroads or urban roads without electronic police.

❖ REFERENCES:

[1]
https://books.google.co.in/books?hl=en&lr=&id=vLiTXDHr_sYC&oi=fnd&pg=PA3&dq=supervised+machine+learning&ots=CZlwuz-Fhl&sig=gXYb8mitT60pq24Qy7A08-GxJno&redir_esc=y#v=onepage&q=supervised%20machine%20learning&f=false

[2]
https://link.springer.com/chapter/10.1007/978-0-387-77240-0_10

[3]
http://kr.nvidia.com/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf

[4]
https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

[5]
https://towardsdatascience.com/understanding-alexnet-a-detailed-walkthrough-20cd68a490aa

[6]
https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c

[7]
https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0