Chapter 1

Background

1.1 Machine Learning

1.1.1 Introduction

Machine Learning (ML) is a field which is raised out of Artificial Intelligence (AI). Applying AI, we wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges. There was a realisation that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realise it while using it.

Finding patterns in data on planet earth is possible only for human brains. The data being very massive, the time taken to compute is increased, and this is where Machine Learning comes into action, to help people with large data in minimum time.
There are three kinds of Machine Learning Algorithms :

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning

A majority of practical machine learning uses supervised learning. In supervised learning, the system tries to learn from the previous examples that are given. Speaking mathematically, supervised learning is where you have both input variables (x) and output variables (Y) and can use an algorithm to derive

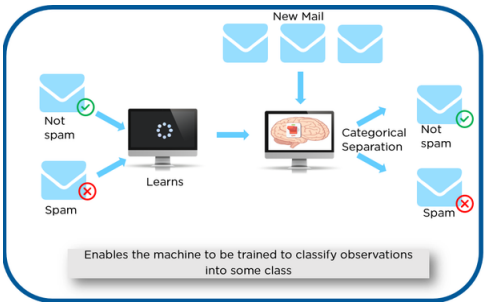


Figure 1.1: Example of supervised Learning

the mapping function from the input to the output. The mapping function is expressed as $Y = f(x)$.

As shown in Figure 1.1, we have initially taken some data and marked them as 'Spam' or 'Not Spam'. This labeled data is used by the training supervised model, in order to train the model. Once it is trained, we can test our model by testing it with some test new mails and checking of the model is able to predict the right output.

Supervised learning problems can be further divided into two parts, namely **classification**, and **regression**.

Classification : A classification problem is when the output variable is a category or a group, such as "black" or "white" or "spam" and "no spam".

Regression : A regression problem is when the output variable is a real value, such as "Rupees" or "height."

Some Supervised learning algorithms include:

- Decision trees
- Support-vector machine
- Naive Bayes classifier
- k-nearest neighbors
- linear regression

Unsupervised Learning

In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data. Mathematically, unsupervised learning is when you only have input data (X) and no corresponding output variables. This is called unsupervised learning because unlike supervised learning above, there are no given correct answers and the machine itself finds the answers. In Figure 1.2, we have given some characters to our model which are 'Ducks' and 'Not

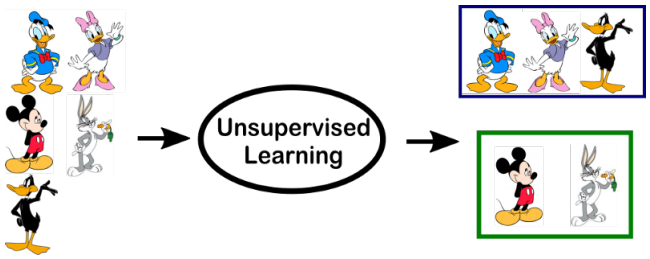


Figure 1.2: Example of unsupervised Learning

Ducks'. In our training data, we don't provide any label to the corresponding data. The unsupervised model is able to separate both the characters by looking at the type of data and models the underlying structure or distribution in the data in order to learn more about it. Unsupervised learning problems can be further divided into **association** and **clustering** problems.

Association : An association rule learning problem is where you want to discover rules that describe large portions of your data, such as "people that buy X also tend to buy Y".

Clustering : A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.

Reinforcement Learning

A computer program will interact with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car). The program is provided feedback in terms of rewards and punishments as it navigates its problem space. Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it continuously trains itself using trial and error method. In Figure 1.3, we can see that the agent is given 2 options i.e. a path with water or a path with fire. A reinforcement algorithm works on reward a system i.e. if the agent uses the fire path then the rewards are subtracted and agent tries to learn that it should avoid the fire path. If it had chosen the water path or the safe path then some points would have been added to the reward points, the agent then would try to learn what path is safe and what path isn't

1.1.2 Neural Networks

Neural Networks are a class of models within the general machine learning literature. Neural networks are a specific set of algorithms that have revolutionized the field of machine learning. They are inspired by biological neural networks and the current so called deep neural networks have proven to work

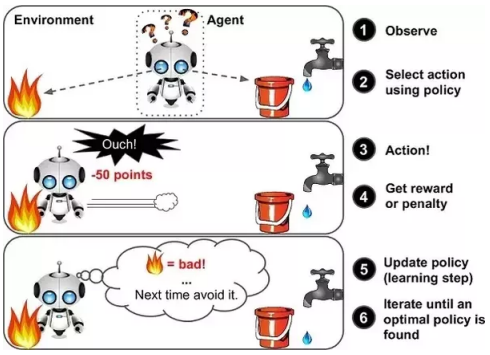


Figure 1.3: Example of Reinforcement Learning

quite very well. Neural Networks are themselves general function approximations, that is why they can be applied to literally almost any machine learning problem where the problem is about learning a complex mapping from the input to the output space.

1.1.3 A single Neuron

The basic unit of computation in a neural network is the neuron, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. In purely mathematical terms, a neuron in the machine learning world is a placeholder for a mathematical function, and its only job is to provide an output by applying the function on the inputs provided. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function *f (defined below)* to the weighted sum of its inputs as shown in Figure 1.4. The network takes numerical inputs *X1* and *X2* and has weights *w1* and *w2*

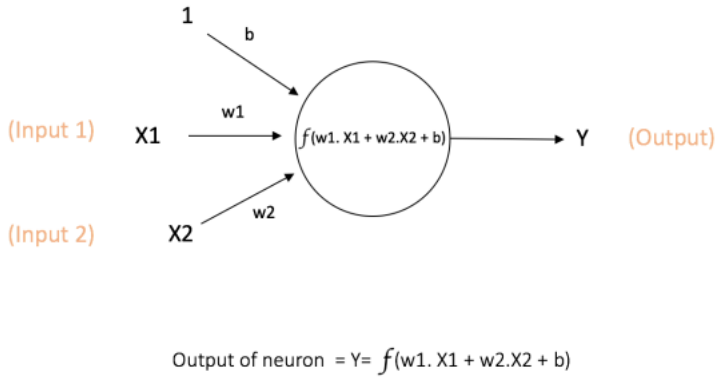


Figure 1.4: An example of a single Neuron

associated with those inputs. Additionally, there is another *input 1* with weight b (called *Bias*) associated with it. The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives). The output Y from the neuron is computed as shown in the Figure 1.4. The function f is non-linear and is called **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data are non linear and we want neurons to learn these non-linear representations.

Activation Functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions:

Sigmoid : takes a real-valued input and squashes it to range between 0 and 1. Its formula is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It is easy to understand and apply but it has major reasons which have made it fall out of popularity:

- Vanishing gradient problem
- Its output isn't zero centered. It makes the gradient updates go too far in different directions.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

Tanh : takes a real-valued input and squashes it to the range [-1, 1]. Its formula is:

$$tanh(x) = 2\sigma(2x) - 1$$

Now it's output is zero centered because its range in between -1 to 1. Hence optimization is easier in this method and in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.

ReLU : ReLU stands for *Rectified Linear Unit*. It takes a real-valued input and thresholds it at zero (replaces negative values with zero). So its formula is:

$$f(x) = max(0, x)$$

It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. Seeing the mathamatical form of this function we can see that it is very simple and efficinent . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies

vanishing gradient problem . Almost all deep learning Models use ReLu nowadays.

Figure 1.5 show each of the above activation functions.

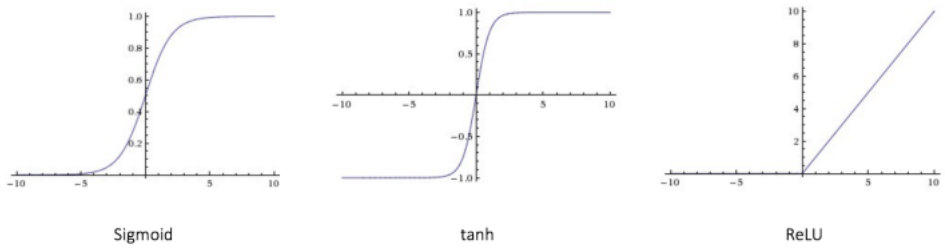


Figure 1.5: Plots of Activation functions

Feedforward Neural Network

Till now we have covered neuron and activation functions which together for the basic building blocks of any neural network. The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in layers. A layer is nothing but a collection of neurons which take in an input and provide an output. Inputs to each of these neurons are processed through the activation functions assigned to the neurons. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them. An example of a feedforward neural network is shown in Figure 1.6. A feedforward neural network can consist of three types of nodes:

Input Nodes The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.

Hidden Nodes The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

Output Nodes The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

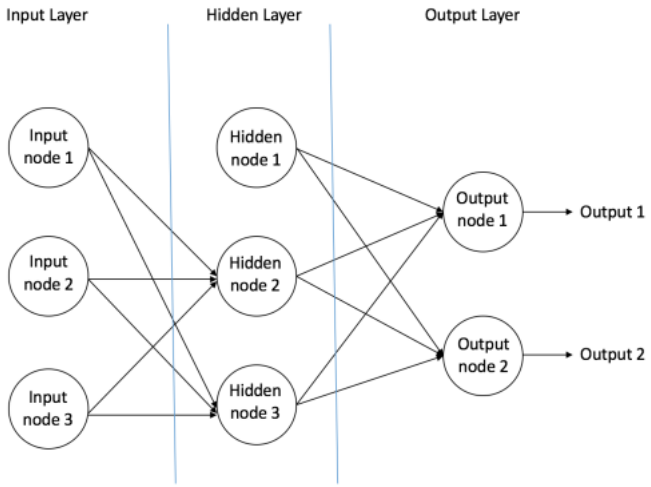


Figure 1.6: An example of a Feedforward Neural Network

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle). Another important point to note here is that each of the hidden layers can have a different activation function, for instance, hidden layer1 may use a sigmoid function and hidden layer2 may use a ReLU, followed by a Tanh in hidden layer3 all in the same neural network. Choice of the activation function to be used again depends on the problem in question and the type of data being used.

1.1.4 2D Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is one of the variants of neural networks used heavily in the field of Computer Vision. It derives its name from the type of hidden layers it consists of. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers. Here it simply means that instead of using the normal activation functions defined above, convolution and pooling functions are used as activation functions. It can take in an input image, assing importance (learning weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to the other classification algorithms. While in primitive method filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the structure of

the Visual Cortex. However, most ConvNets costist mainly in 2 parts:

- **Feature extractor** :
  This part of the network takes as input the image and extract features that are meaningful for its classification. It amplifies aspects of the input that are important for discrimination and suppresses irrelevant variations. Usually, the feature extractor cosists of several layers. For instance, an image which could be seen as an array of pixel values. The first layer often learns reprensations that represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardeless of small variations in the edge positions. Finally, the third may assemble motifs into larger combinations that correspond to paths of familiar objects, and subsequent layers would detect objects as combinations of these parts.

- **Classifier** :
  This part of the network takes as input the previously computed features and use them to predict the correct label.

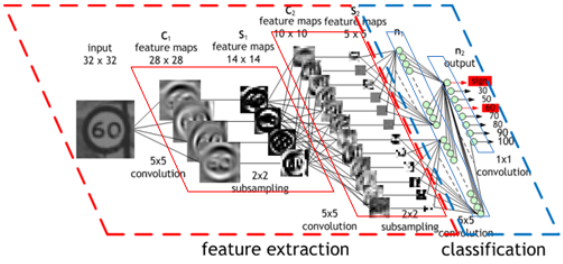


Figure 1.7: Typical structure of a ConvNet

Convolutional Layers In order to extract such features, ConvNets use 2D convolution operations. These operations take place in convolutional layers. Convolutional layers consist of a set of learnable filters. Every filter is small spatially (along widht and height), but extends through the full depth of input. During forword pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position (as Figure 1.8 shows). The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

Figure 1.8: Convolution with kernel of 3, stride of 2 and padding of 1

Pooling Layers Pooling Layers are also refered as downsampling layers and are used to reduce the spatial dimensions, but not depth, on a convolution neural network. The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. The main advantages of pooling layer are:

- We gain computation performance since the amount of parameters is reduce.
- Less parameters also means We deal with overfitting situations.

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

Average Pooling Calculate the average value for each patch on the feature map.

Maximum Pooling (or Max Pooling) Calculate the maximum value for each patch of the feature map.

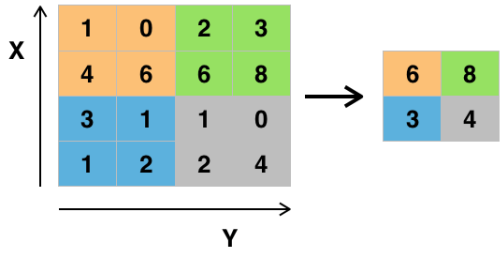


Figure 1.9: Example of Max pooling operation with a 2x2 filter and stride of 2

1.1.5 3D Convolutional Neural Network

Traditionally, ConvNets are targeting RGB images (3 channels). The goal of 3D CNN is to take as input a video and extract features from it. When ConvNets extract the graphical characteristics of a single image and put them in a vector (a low-level representation), 3D ConvNets extract the graphical characteristics of a set of images. 3D CNNs takes in to account a temporal dimension (the order of the images in the video). From a set of images, 3D CNNs find a low-level representation of a set of images, and this representation is useful to find the right label of the video (a given action is performed).
In order to extract such features, 3D ConvNets use 3D convolution operations.

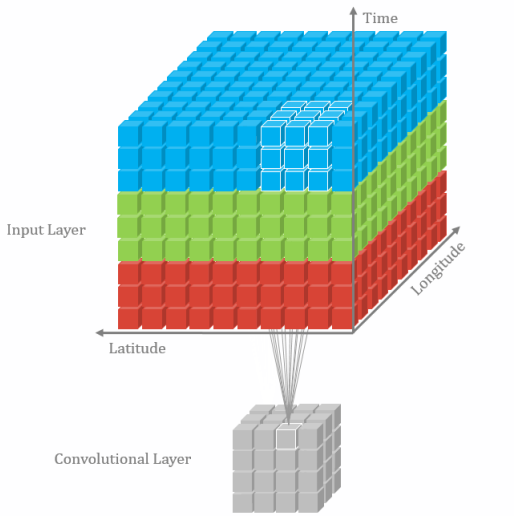


Figure 1.10: 3D Convolution operation

There are several existing approaches to tackle the video classification. This is a nonexaustive list of existing approaches:

- **ConvNets + LSTM cell** : Extract features from each frame with a ConvNet, passing the sequence to an RNN
- **Temporal Relation Networks** : Extract features from each frame with a ConvNet and pass the sequence to an MLP
- **Two-Stream Convolutional Networks** : Use 2 CNN, 1 spatial stream ConvNet which process one single frame at a time, and 1 Temporal stream ConvNet which process multi-frame optical flow

1.2 Object Detection

Within the field of Deep Learning, the sub-discipline called "Object Detection" involves processes such as identifying the objects through a picture, video or a webcam feed. Object Detection is used almost everywhere these days. The use cases are endless such as Tracking objects, Video surveillance, Pedestrian detection etc. An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a label that specifies the class of fruit they represent (e.g. an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

The main process followed by most of CNN for Object Detection is:

1. Fistly, we do feature extraction using as backbone network, the first Convolutional Layers of a known pre-trained CNN such as AlexNet, VGG, ResNet etc.

2. Then, we propose regions of interest (ROI) in the image. These regions contain possibly an object, which we are looking for.

3. Finally, we classify each proposed ROI.

1.2.1 Region Proposal Network

From the 3 above steps, the 2nd step is considered to be very important. That is because, in this step, we should choose regions of the image, which will be classified. Poor choice of ROIs means that the CNN will pass by some object that are located in the image, because, they were not be proposed to be classified.

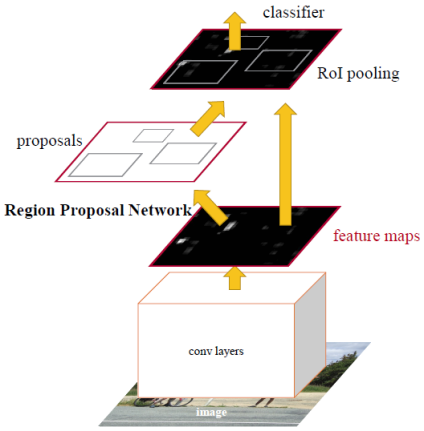


Figure 1.11: Region Proposal Network's structure

The first Object-Detection CNNs use several algorithms for proposing ROIs. For example, R-CNN[**?**], and Fast R-CNN[**?**] used Selective Search Algorithm for extracting ROIs. One of noverlties introduced by the Faster R-CNN[**?**] is **Region Proposal Network** (RPN). Its Function is to propose ROIs and its structure can be shown in 1.11. As we can see, RPN is consisted of:

- 1 2D Convolutional Layer
- 1 score layer
- 1 regression layer

Another basic element of RPN is the **anchors**. Anchors are predefined boxes used for extracting ROIs. In figure 1.12 is depicted an exaple of some anchors

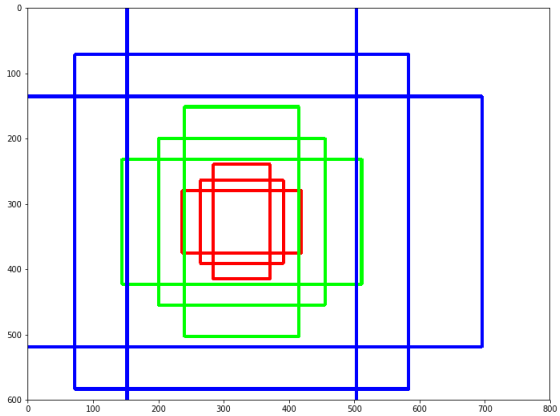


Figure 1.12: Anchors for pixel (320,320) of an image (600,800)

For each feature map's pixel corresponds **k (k=9)** anchors (3 different scales and 3 different ratios 1:1, 1:2, 2:1).

As a result, RPN gets as input feature maps extracted from the backbone CNN. Then performs 2D convolution over this input and passes the output to its scoring layer and regression layer. Those scores represent the confidence of existing an object in this specific position. On top of that, regression layer outputs 4k displacements, 4 for each anchor. Finally, we keep as output only the *n-best scoring* anchors.

1.2.2 Roi Align

The biggest problem facing Object Detection Networks is the need for fixed input size. Classification networks require a fixed input size, which is easy for image classification because it is handle by resizing the input image. However, in object recognition, each proposal has a different size and shape. This creates

the need for converting all proposals to a fixed shape. At Fast-RCNN([?]) and Faster-RCNN([?]), this operation happens by applying Roi Pooling. However, this wrapping is digitalized because the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps as shown in Figure 1.13a at the top left diagram. As a result, each target cells may not be in the same size (Figure 1.13c).

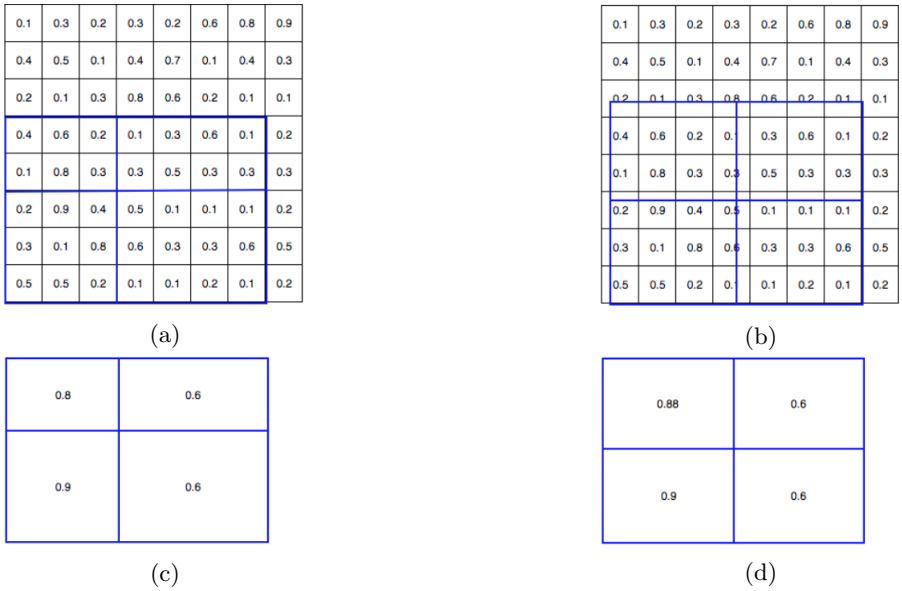


Figure 1.13

On the other hand, Mask-RCNN ([?]) introduced Roi Align operation. Roi Align avoids digitalizing the boundary of the cells as shown in Figure 1.13b, and achieves to make every target cell to have the same size according to Figure 1.13d. In order to calculate feature maps values, Roi Align uses bilinear interpolation as shown in Figure 1.14. This means that we calculate the value of the desired bins according to their neighbours'.

1.2.3 Non-maximum suppression (NMS) algorithm

That's because the neighbour windows have similar scores to some extent. This Another proble that object detection networks face is that neighbourd bounding boxes have similar scores to some extent. Most object detection systems employ a sliding window or a Region Proposal Network for proposing areas in the image that is likely to contain an object. These techniques,which return several areas in the images, achieve high recall performance. However, in these approaches, more that 1 proposal may be related with only one groundtruth object coordinates. This situation creates the need for choosing the best proposals, because, alternatively, hundreds of unnecesssary proposals will be classified.

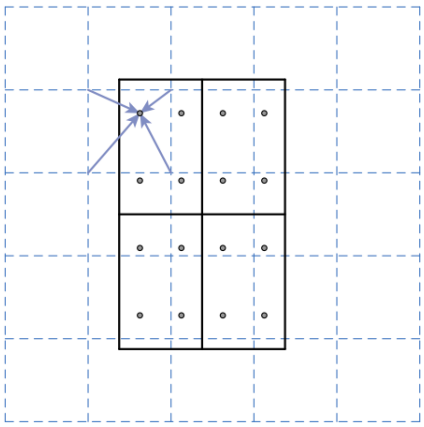


Figure 1.14

For that reason, Non-Maximum Suppression (NMS) algorithm was proposed for filtering these proposals base on some criteria. NMS gets as input a list of proposal bounding boxes B, their corresponding confidence score S and an overlap threshold N and return as output a list of the final filtered proposals D. NMS algorith's steps are:

1. Initialize an empty list D. Select the proposal with the highest confidence score, remove it from B and add it to D.

2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, remove from B.

3. From the remaining proposals, picked again the one with the highest score and remove it from B.

4. Repeat steps 2 and 3 until no more proposals are left in list B.

The aforementioned algorithm shows that the whole process depends mostly on a sigle threshold value. So that makes the selection of threshold a crucial factor for the perfomance of the model. In some situations, bad choice of the threshold may make the network to remove bounding boxes with good confidence score, if there are side by side. Figure **??** shows a situation like this, where red and blue boxes will be removed because of the presence of the black box.

Soft NMS

A simple and efficinet way to deal the aforementioned situation is to use Soft-NMS algorithm, which is presented in [**?**]. Soft-NMS algorithm is based on the idea of reducing condifence score of the proposals propotional to their overlap

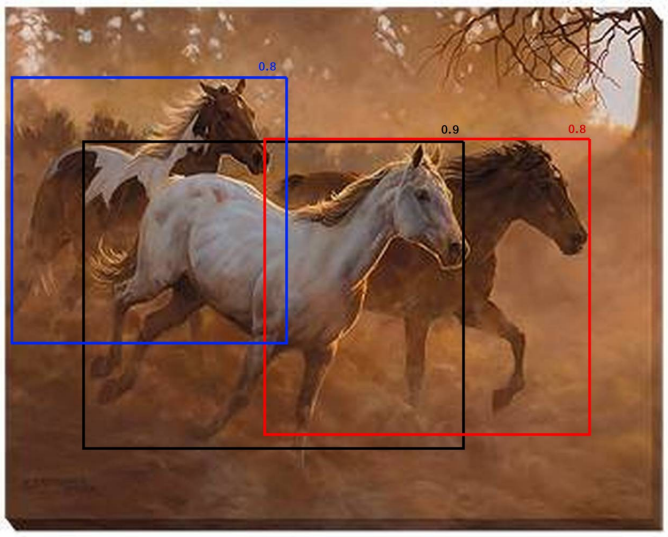


Figure 1.15: Example of a situation where NMS algorithm will remove good proposals

score, instead of completely removing them. The score calculation follows the formula

$$s_i = \begin{cases} s_i, & overlapscore(M, b_i) < N_t \\ s_i(1 - overlapscore(M, b_i)), & overlapscore(M, b_i) \geq N_t \end{cases}$$

where s_i is the score of proposal i, b_i is the box coordinates of proposal i, M is the coordinates of the box with the maximum confidence and N_t is the overlap threshold. So steps of soft-NMS algorithm are:

1. Select the proposal with the highest confidence score, remove it from B and add it to D.
2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, recalculate their confidence score according to previous formula.
3. From the remaining proposals, picked again the one with the highest score and remove it from B.
4. Repeat steps 2 and 3 until no more proposals are left in list B.

1.3 Losses and Metrics

In order to train our model and check its performance, we use some known Loss functions and Metrics used in Object Detection systems.

1.3.1 Losses

For training our network, we use **Cross Entropy Loss** for classification layers and **smooth L1-loss** for bounding box regression in each frame and their diagram is show at Figure 1.16.

Cross Entropy Loss

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

Entropy is the measure of uncertainty assosiated with a given distribution *q(y)* and its formula is:

$$H = -\sum_{i=1}^{n} p_i \cdot \log p_i$$

Intuitively, entropy tells us how "surprised" we are when some event E happened. When we are sure about an event E to happened ($p_E = 1$) we have 0 entropy (we are not surprised) and vise versa.

On top of that, let's assume that we have 2 distributions, one known (our network's distribution) *p(y)* and one unknown (the actual data's distribution) *q(y)*. Cross-entropy tells us how accurate is our known distribution in predicting the unknown distribution's results. Respectively, Cross-entropy measures how accurate is our model in predicting the test data. Its formula is:

$$H_p(q) = -\sum_{c=1}^{C} q(y_c) \cdot log(p(y_c))$$

Smooth L1-loss

Smooth L1-loss can be interpreted as a combination of L1-loss and L2-loss. It behaves as L1-loss when the absolute value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero. It is usually used for doing box regression on some object detection systems like Fast-RCNN([**?**]), Faster-RCNN([**?**]) and it is less sensitive to outliers according according to [**?**]. As showm in [**?**], its formula is:

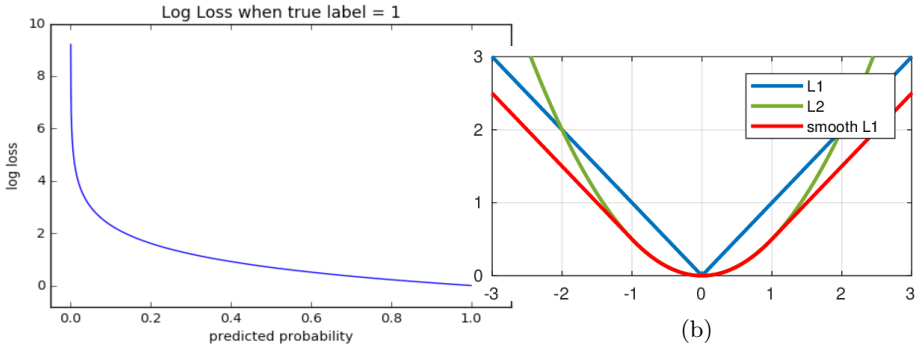
$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } x < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

It is similar to Huber loss whose formula is:

$$L_\delta(x) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

if we set δ parameter equal 1.

Smooth L1-loss combines the advantages of L1-loss (steady gradients for large values of x) and L2-loss (less oscillations during updates when x is small). Figure 1.16 b shows a comparison between L1-norm, L2-norm and smooth-L1 .



(a) (b)

Figure 1.16: (a) and (b) show the behavior of cross-entropy loss and smooth-L1 respectively.

1.3.2 Metrics

Evaluating our machine learning algorithm is an essential part of any project. The way we choose our metrics influences how the performance of machine learning algorithms is measured and compared. They influence how to weight the importance of different characteristics in the results and finally, the ultimate choice of which algorithm to choose. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model.

At first, we introduce some basic evaluation metrics in order,then, to present those we use for our assesment.

Intersection over Union

The first and most important metric that we use is Intersection over Union (IoU). IoU measures the overlap between 2 boundaries. It is usually used in Object Recognition Networks in order to define how good overlap a predicted bounding box with the actuall bounding box as shown in Figure 1.17. We predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

Intersection over Union is defined as:

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

In Figure 1.17, spatial IoU between 2 bounding boxes, (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4),is presented, which means IoU metric is implemented for x-y dimensions. Area of

Figure 1.17: Example of IoU scoring policy

overlap and area of union can be defined as:

$$\text{Area of overlap } = \|(min(x_2, x_4) - max(x_1, x_3), min(y_2, y_4) - max(y_1, y_3))\|$$

$$\text{Area of union } = \|(max(x_2, x_4) - min(x_1, x_3), max(y_2, y_4) - min(y_1, y_3))\|$$

On top of that, we can implement IoU for 1 dimension and for 3 dimensions.

1D IoU We can name 1D IoU as temporal overlap. Let's consider 2 temporal segments (t_1, t_2) and (t_3, t_4), between which we want to estimate their overlap score. Their IoU can be described as:

$$\text{Length of overlap } = \|(min(t_2, t_4) - max(t_1, t_3))\|$$

$$\text{Length of union } = \|(max(t_2, t_4) - min(t_1, t_3))\|$$

3D IoU 3-dimentional Intersection over Union which can, also, be named as spatiotemporal IoU, can be defined by 2 ways:

3D boxes are cuboids In this case, 3D boxes can be written as (x, y, z, x', y', z'). So, the IoU overlap between 2 boxes, $(x_1, y_1, z_1, x_2, y_2, z_2)$ and $(x_3, y_3, z_3, x_4, y_4, z_4)$, is defined as:

$$\begin{aligned}\text{Volume of overlap } &= \|(min(x_2, x_4) - max(x_1, x_3),\\ min(y_2, y_4) - max(y_1, y_3), &min(z_2, z_4) - max(z_1, z_3))\|\\ \text{Volume of union } &= \|(max(x_2, x_4) - min(x_1, x_3),\\ max(y_2, y_4) - min(y_1, y_3), &max(z_2, z_4) - min(z_1, z_3)\end{aligned}$$

x-y are continuous and z discrete In this case 3D boxes is defined as a sequence of 2D boxes (x, y, x', y'). For this definition, z-dimension is discrete, and IoU can be defined with 2 ways, which both result in the same overlapping score. Let's cosider 2 sequences of boxes, with temporal limits (t_1, t_2) and (t_3, t_4). We calculate their IoU following one of the following methods:

1. IoU is the product between temporal-IoU and the average spatial-IoU between 2D boxes in the overlapping temporal area and it is described as:

$$IoU = IoU((t_1, t_2), (t_3, t_4)) \cdot \frac{1}{K_2 - K_1} \sum_{i=K_1}^{K_2} IoU(X_1^i, X_2^i)$$

where

- $K_1 = min(t_2, t_4)$
- $K_2 = max(t_1, t_3)$
- $X_1^i = (x_1^i, y_1^i, x_2^i, y_2^i)$ and $X_2^i = (x_3^i, y_3^i, x_4^i, y_4^i)$

2. IoU is the average spatial-IoU if we consider 2D boxes as $(0, 0, 0, 0)$ if $t \notin [t_{start}, t_{finish}]$ and it is written as:

$$IoU = \frac{1}{K} \sum_{i=min(t_1,t_3}^{max(t_2,t_4)} IoU(X_1^i, X_2^i)$$

- $K = max(t_2, t_4) - min(t_1, t_3)$
- $X_1 = (x_1, y_1, x_2, y_2)$ if $i \in [t_1, t_2]$ or $(0, 0, 0, 0)$ if $i \notin [t_1, t_2]$
- $X_2 = (x_3, y_3, x_4, y_4)$ if $i \in [t_3, t_4]$ or $(0, 0, 0, 0)$ if $i \notin [t_3, t_4]$

From above implementations, we are involve mostly with temporal and spatiotemporal IoU.

Precision & Recall

In order to describe **precision** and **recall** metrics, we will use an example. Let's consider a group of people in which, some of them are sick and the others are not. We use a network which is able to predict if a person is sick or not if we give it some data as input.

Precision measures how accurate are our model's predictions. i.e. the percentage of predictions that are correct. In our case, how accurate is our model when it predicts that a person is sick.

Recall measures how good we found all the sick people. In our case, how many of the acctual sick people we managed to find.

Their definisions are:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where

- TP = True positive, which means that we predict a person to be sick and he is actually sick.
- TN = True negative, we predict that a person isn't sick and he isn't.
- FP = False positive, we predict a person to be sick but he isn't actually.
- FN = False negative, we predict a person not to be sick but he actually is.

From these 2 metrics we use recall metric in order to evaluate our networks performance, and more specifically, its performance on finding good action tube proposals. We consider a groundtruth action as true positive when there is at least 1 proposed action tube that its IoU overlap score is bigger that a predifed threshold. If there is no such action tube, then we consider this groundtruth action tube as false negative.

mAP

Precision and recall are single-value metrics based on the whole list of predictions. By looking their formulas, we can see that there is a trade-off between precision and recall performance. This trade-off can be adjusted by the softmax threshold, used in model's final layer. In order to have high precision performance, we need to decrease the number of FP. But this will lead to decrease recall performance and vice-versa.

As a result, these metrics fail to determine if a model is performing well in object detection tasks as well as action detection tasks. For that reason, we use mean Average Precision (mAP) metric.

AP (Average precision) Before defining mAP metric, we will define Average Precision metric (AP). AP is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. As mention before, during classification stage, our prediction result in True positive(TP), False positive(FP), True Negative(TN) and False Negative(FN). For object recognition and action localization networks, we don't care about TN. We consider a prediction as True positive when our prediction (an bounding box for object detection network or a sequence of bounding boxes in action localization networks) overlaps with the groundtruth bounding box/action tube over a predefined threshold, and our predicted class is the same as groundtruth's. In addition, we consider a False Negative when either no detection overlaps with the groundtruth bounding box/action tube or our prediction was incorrect. We consider a prediction as False positive, when more that one predictions overlap with the groundtruth. In this situation, we consider the prediction with the biggest confidence score as TP and the rest as FP.

For a class, we need to calculate, first, its precision and recall scores in order to calculate its AP score. We sort our predictions according to their confidence

score and for each new prediction we calculate precision and recall values. An example, for a class containing 4 TP and 8 predictions is shown in Table 1.1. Precision and recall are calculated according to the number of elements that are above in the order. So, for rank #3, Precision is calculated as the proposion of TP = 2/3 = 0.67 and Recall as the proportion of TP out of all the possible TP = 2/4 = 0.5.

Rank	Prediction	**Precision**	**Recall**
1	Correct	1.0	0.25
2	Correct	1.0	0.5
3	False	0.67	0.5
4	False	0.5	0.5
5	False	0.4	0.5
6	Correct	0.5	0.75
7	False	0.42	0.75
8	Correct	0.5	1

Table 1.1: Ordered by confidence predictions and their precision and recall values

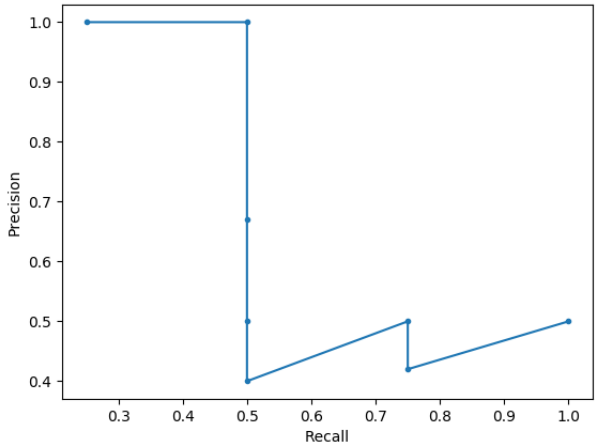


Figure 1.18: Precision/Recall curve

We plot Precision against Recall and their curve is shown in Figure 1.18. The general definition for Average Precision(AP) is finding the area under the precision-recall curve and its formula is:

$$AP = \int_0^1 p(r)dr$$

Precision and Recall values $\in [0, 1]$, so AP $\in [0, 1]$, too. This integral can be replace with a finite, as we have a finite number of predictions. So its formula

is:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where $P(k)$ is the precision until prediction k and Δr is the change in recall from $k-1$ to k.

Interpolated Precision At we can see in 1.18, P-R curve has a zigzag pattern as it goes down with false predictions, and goes up with correct. So, before calculation AP, we need to smooth out this zigzag pattern using Intenpolated precision, as introduced in [**?**]. Interpolated precision is calculated at each recall level r by taking the maximum precision measured for that r and it is defined as:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r}\geq r} p(\tilde{r})$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$. Graphically, at each recall level, we replace each precision value with the maximum precision value to the right of that recall level. At Figure 1.19 are shown both P-R curvers. The previous P-R curve has blue colour and the interpolated has red color.

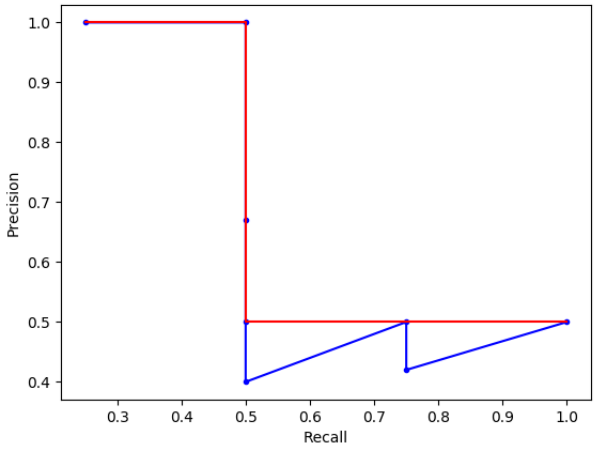


Figure 1.19: Both P-R curves.

In order to calculate AP, we sample the curve at all unique recalls values, whenever the maximum precision drops. On top of that, we define mean Average Precision (mAP) the mean of the AP for each class. So, AP and mAP are defined as

$$AP = \sum (rn + 1 - r_n) p_{interp}(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r}\geq r_{n+1} p(\tilde{r})} p(\tilde{r})$$

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Mean Average Best Overlap - MABO

In order to evaluate more the quality of our proposals, both during TPN and connecting tube stages, recall metric isn't enough. That's because recall metric tells us only in how many known objects there was at least 1 proposal that satified the detection criterion. However, it doen't tells us how close the proposals are to the groundtruth tubes. In order to quantify this performance, Mean Average Best Overlap (MABO) was introduced by [**?**]. The importance of MABO can be clarified we consider figure 1.20.

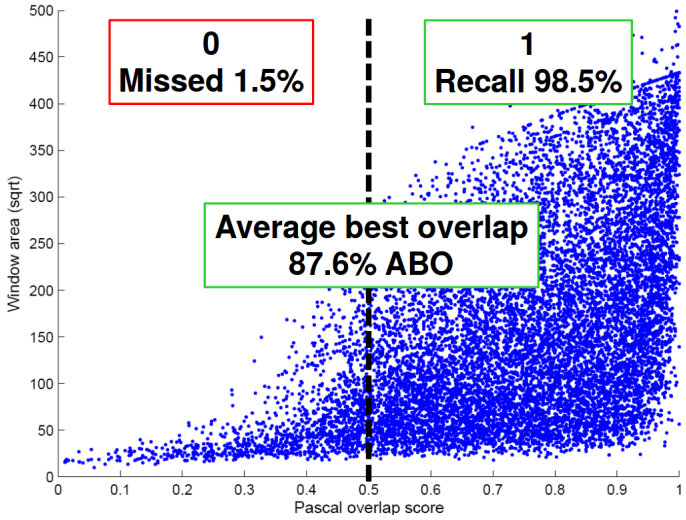


Figure 1.20: Recall versus MABO exaple

As we can see, recall performance is almost perfect, but, MABO perfonmance, which tells us where most proposal overlap scores are gather, is just fine. In order to define MABO, we need first to define Average Best Overlap. Let $c \in C$ denote a class c from the set of all classes C and G^c the set of ground truth annotations of this class in all images; let L be the set of all generated object proposals for all images. Average Best Overlap is defined as the average value of the maximum overlap score, (in our situation, we use intersection over union) of L with each groundtruth annotation $g \in G^c$. The Mean Average Best Overlap (MABO) is defined as the average value of all class ABO values :

$$MABO = \frac{1}{|C|} \sum_{c \in C} [\frac{1}{|G^C|} \sum_{g \in G^c} \max_{l \in L} IoU(g, l)]$$

In order our situation, we consider only one class, for defining proposals, so, MABO identifies with ABO.