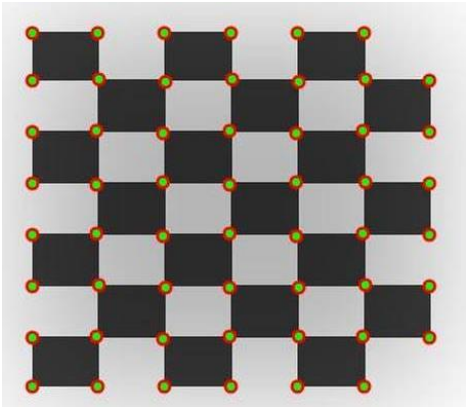
Unit-5 - Feature Extraction

Syllabus : *Extracting Interest Points and Their Descriptors (with Harris, SIFT and SURF) in Image Pairs, Principal Component Analysis (PCA) and Linear Discriminant Analysis for Image Recognition- Image Classification using SVM-ANN- Feedforward and Back propagation-Object Detection using CNN-RCNN.*

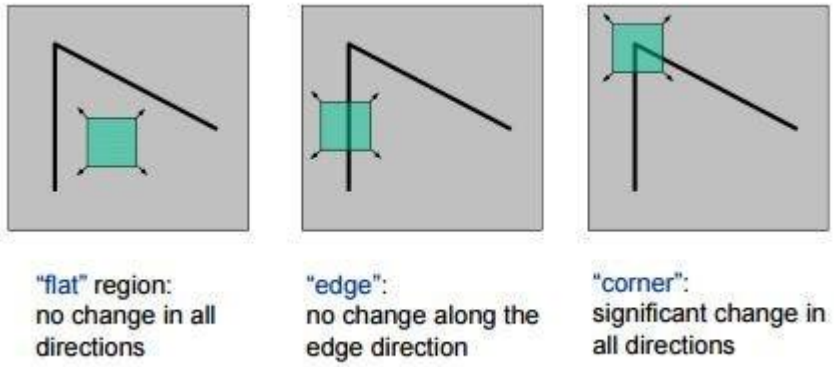
Extracting Interest Points and Their Descriptors (Harris Detector)

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. It was first introduced by Chris Harris and Mike Stephens in 1988 upon the improvement of Moravec's corner detector. Compared to the previous one, Harris' corner detector takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angles, and has been proved to be more accurate in distinguishing between edges and corners. Since then, it has been improved and adopted in many algorithms to preprocess images for subsequent applications.



Corner

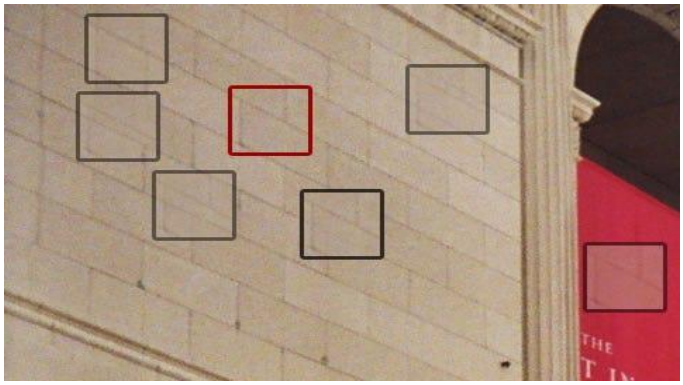
A corner is a point whose local neighborhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness. Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation, and illumination.



So let's understand why corners are considered better features or good for patch mapping. In the above figure, if we take the flat region then no gradient change is observed in any direction. Similarly, in the edge region, no gradient change is observed along the edge direction. So both flat region and edge region are bad for patch matching since they not very distinctive (there are many similar patches in along edge in edge region). While in corner region we observe a significant gradient change in all direction. Due this corners are considered good for patch matching(shifting the window in any direction yield a large change in appearance) and generally more stable over the change of viewpoint.

Corner Detection

The idea is to consider a small window around each pixel p in an image. We want to identify all such pixel windows that are unique. Uniqueness can be measured by shifting each window by a small amount in a given direction and measuring the amount of change that occurs in the pixel values.



More formally, we take the sum squared difference (SSD) of the pixel values before and after the shift and identifying pixel windows where the SSD is large for shifts in all 8 directions. Let us define the change function E(u,v) as the **sum** of all the sum squared differences (SSD), where u,v are the x,y coordinates of every pixel in our 3 x 3 window and I is the intensity value of the pixel. The features in the image are all pixels that have large values of E(u,v), as defined by some threshold.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

We have to maximize this function $E(u,v)$ for corner detection. That means, we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps, we get the final equation as:

$$E(u,v) \approx [u \quad v]\left(\sum \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix}$$

Now, we rename the summed-matrix, and put it to be M:

$$M = \sum w(x,y)\begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

So the equation now becomes:

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

Remember that we want the SSD to be large in shifts for all eight directions, or conversely, for the SSD to be small for none of the directions. By solving for the eigenvectors of M, we can obtain the directions for both the largest and smallest increases in SSD. The corresponding eigenvalues give us the actual value amount of these increases. A score, R, is calculated for each window:

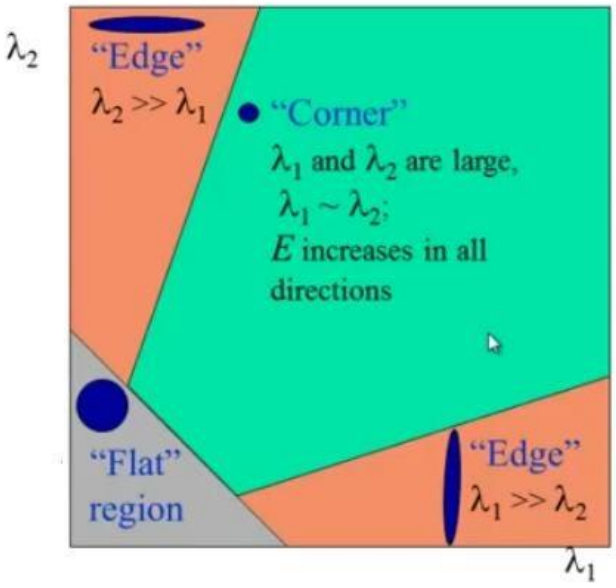
$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

$\lambda 1$ and $\lambda 2$ are the eigenvalues of M. So the values of these eigenvalues decide whether a region is a corner, edge or flat.

- When $|R|$ is small, which happens when $\lambda 1$ and $\lambda 2$ are small, the region is flat.
- When $R<0$, which happens when $\lambda 1 >> \lambda 2$ or vice versa, the region is an edge.
- When R is large, which happens when $\lambda 1$ and $\lambda 2$ are large and $\lambda 1 \sim \lambda 2$, the region is a corner.



High-level pseudocode

1. Take the grayscale of the original image
2. Apply a Gaussian filter to smooth out any noise
3. Apply Sobel operator to find the x and y gradient values for every pixel in the grayscale image
4. For each pixel p in the grayscale image, consider a 3×3 window around it and compute the corner strength function. Call this its Harris value.
5. Find all pixels that exceed a certain threshold and are the local maxima within a certain window (to prevent redundant dupes of features)
6. For each pixel that meets the criteria in 5, compute a feature descriptor.

SIFT (Scale-Invariant Feature Transform)

SIFT Detector is used in the detection of **interest points** on an input image. It allows the identification of localized features in images which is essential in applications such as:

- Object Recognition in Images
- Path detection and obstacle avoidance algorithms
- Gesture recognition, Mosaic generation, etc.

Unlike the **Harris Detector**, which is dependent on properties of the image such as viewpoint, depth, and scale, SIFT can perform feature detection independent of these properties of the image. This is achieved by the transformation of the image data into **scale-invariant coordinates**. The SIFT Detector has been said to be a close approximation of the system used in the primate visual system.

Steps for Extracting Interest Points

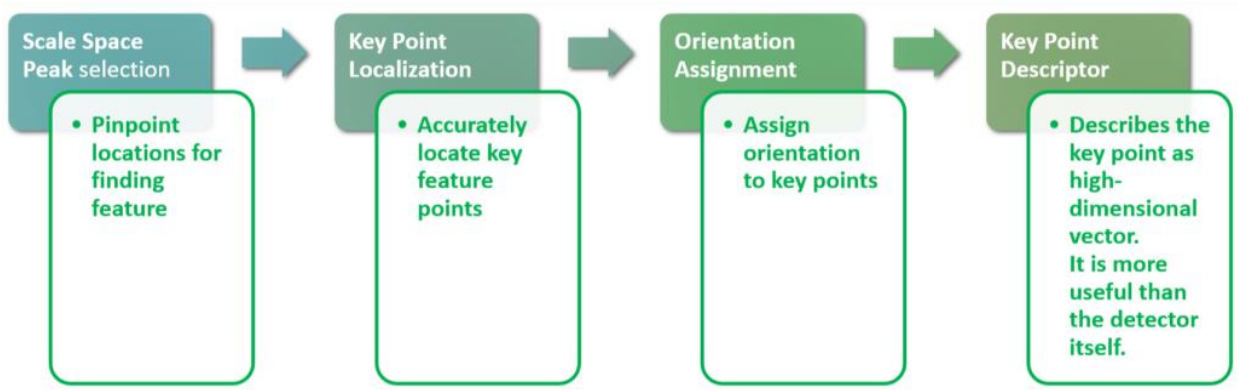


Fig 01: Sequence of steps followed in SIFT Detector

Phase I: Scale Space Peak Selection

The concept of Scale Space deals with the application of a continuous range of Gaussian Filters to the target image such that the chosen Gaussian have differing values of the sigma parameter. The plot thus obtained is called the **Scale Space**. Scale Space Peak Selection depends on the **Spatial Coincidence Assumption**. According to this, if an edge is detected at the **same location in multiple scales** (indicated by zero crossings in the scale space) **then we classify it as an actual edge.**

In 2D images, we can detect the Interest Points using the local maxima/minima in **Scale Space of Laplacian of Gaussian.** A potential SIFT interest point is determined for a given sigma value by picking the potential interest point and considering the pixels in the level above (with higher sigma), the same level, and the level below (with lower sigma than current sigma level). If the point is maxima/minima of all these 26 neighboring points, it is a potential SIFT interest point – and it acts as a starting point for interest point detection.

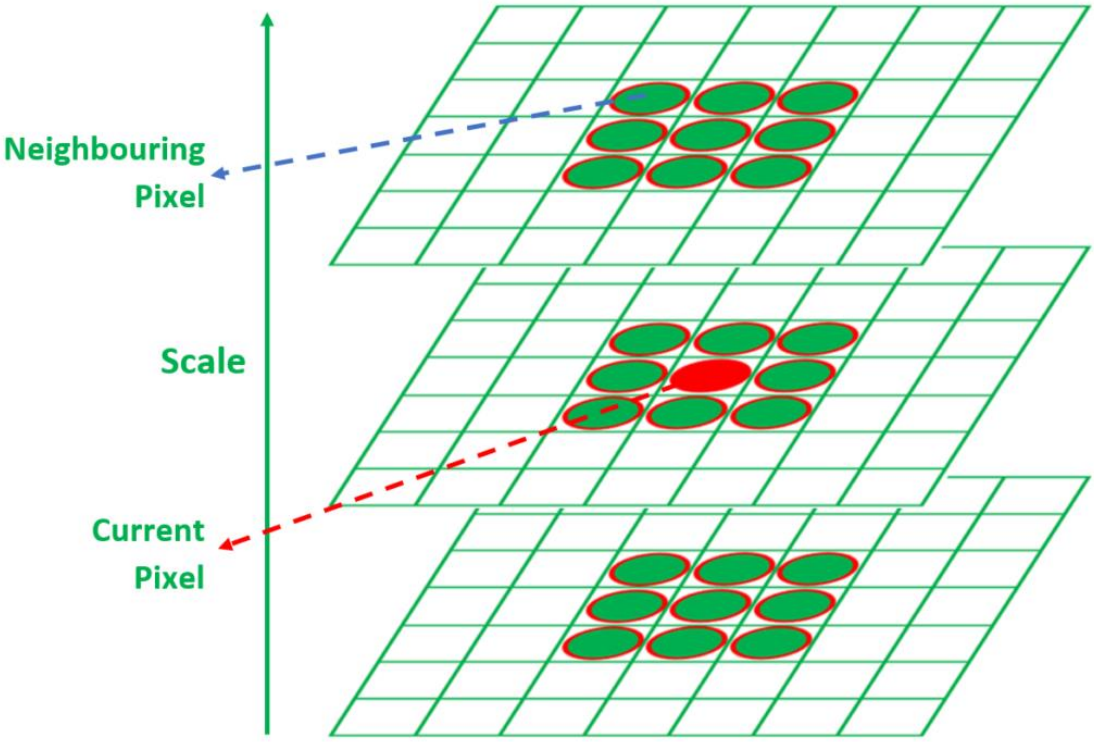


Fig 2: Peaks are selected across Scales.

Phase II: Key Point Localization

Key point localization involves the refinement of keypoints selected in the previous stage. Low contrast key-points, unstable key points, and keypoints lying on edges are eliminated. This is achieved by calculating the **Laplacian** of the keypoints found in the previous stage. The extrema values are computed as follows:

$$z = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x}$$

In the above expression, D represents the Difference of Gaussian. To remove the unstable key points, the value of *z* is calculated and if the function value at z is below a threshold value then the point is excluded.

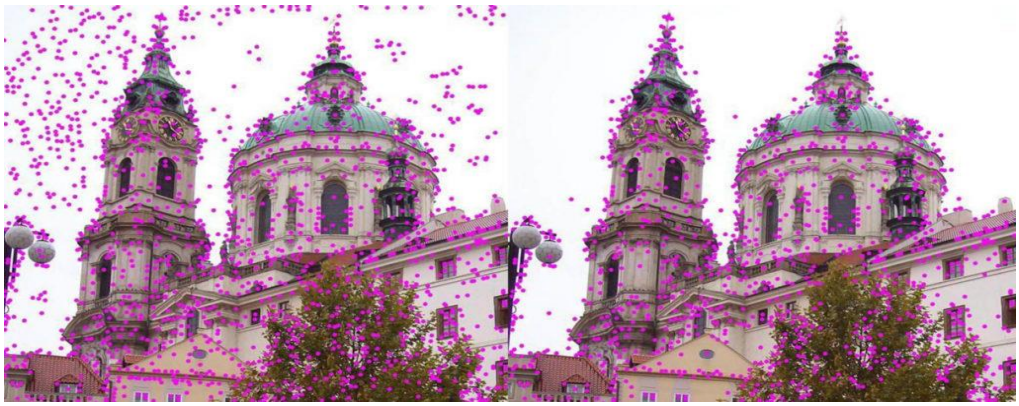


Fig 3 Refinement of Keypoints after Keypoint Localization

Phase III: Assigning Orientation to Keypoints

To achieve detection which is invariant with respect to the rotation of the image, orientation needs to be calculated for the key-points. This is done by considering the neighborhood of the keypoint and calculating the magnitude and direction of gradients of the neighborhood. Based on the values obtained, a histogram is constructed with 36 bins to represent 360 degrees of orientation(10 degrees per bin). Thus, if the gradient direction of a certain point is, say, 67.8 degrees, a value, proportional to the gradient magnitude of this point, is added to the bin representing 60-70 degrees. Histogram peaks above 80% are converted into a new keypoint are used to decide the orientation of the original keypoint.

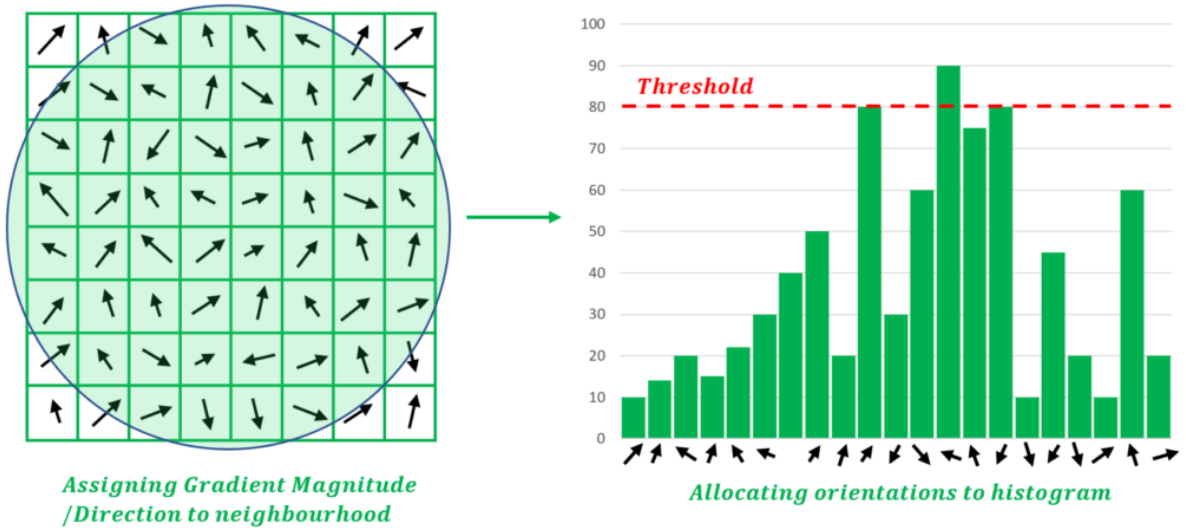


Fig 04: Assigning Orientation to Neighborhood and creating Orientation Histogram

Phase IV: Key Point Descriptor

Finally, for each keypoint, a descriptor is created using the keypoints neighborhood. These descriptors are used for matching keypoints across images. A 16×16 neighborhood of the keypoint is used for defining the descriptor of that key-point. This 16×16 neighborhood is divided into sub-block. Each such sub-block is a non-overlapping, contiguous, 4×4 neighborhood. Subsequently, for each sub-block, an 8 bin orientation is created similarly as discussed in Orientation Assignment. These 128 bin values (16 sub-blocks * 8 bins per block) are represented as a vector to generate the keypoint descriptor.

SURF method (Speeded Up Robust Features)

The SURF method (Speeded Up Robust Features) is a fast and robust algorithm for local, similarity invariant representation and comparison of images. The main interest of the SURF approach lies in its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition

SURF is composed of two steps

- **Feature Extraction**
- **Feature Description**

Feature Extraction

The approach for interest point detection uses a very basic Hessian matrix approximation.

Integral images

The Integral Image or *Summed-Area Table* was introduced in 1984. The Integral Image is used as a quick and effective way of calculating the sum of values (pixel values) in a given image — or a rectangular subset of a grid (the given image). It can also, or is mainly, used for calculating the average intensity within a given image.

They allow for fast computation of box type convolution filters. The entry of an integral image $I_{\sum}(x)$ at a location $x = (x,y)^T$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and x.

$$I_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \leqslant x} \sum_{j=0}^{j \leqslant y} I(i,j)$$

With I_{Σ} calculated, it only takes four additions to calculate the sum of the intensities over any upright, rectangular area, independent of its size.

Hessian matrix-based interest points

Surf uses the Hessian matrix because of its good performance in computation time and accuracy. Rather than using a different measure for selecting the location and the scale (Hessian-Laplace detector), surf relies on the **determinant of the Hessian matrix** for both. Given a pixel, the Hessian of this pixel is something like:

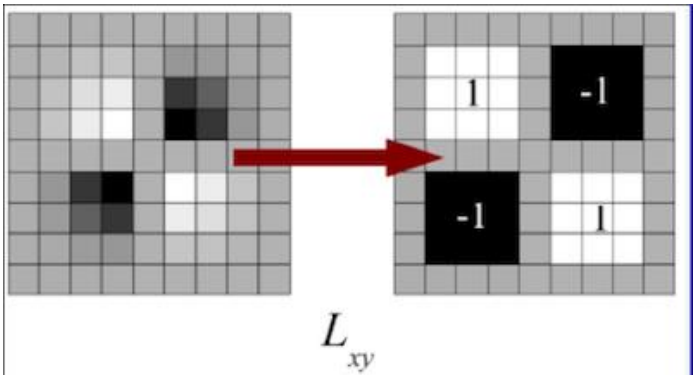
$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

For adapt to any scale, we filtered the image by a Gaussian kernel, so given a point X = (x, y), the Hessian matrix H(x, σ) in x at scale σ is defined as:

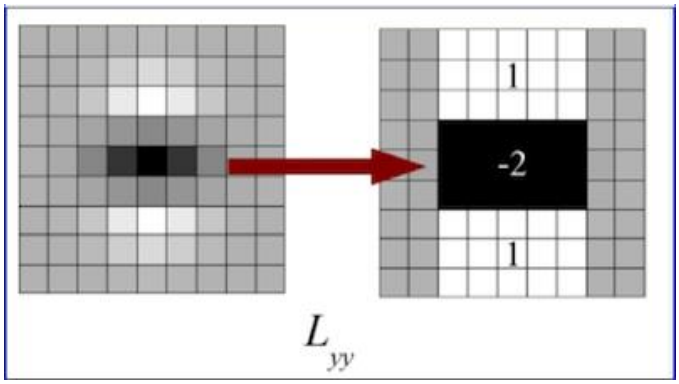
$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

where Lxx(x, σ) is the convolution of the Gaussian second order derivative with the image I in point x, and similarly for Lxy (x, σ) and Lyy (x, σ). Gaussians are optimal for scale-space analysis but in practice, they have to be discretized and cropped. This leads to a loss in repeatability under image rotations around odd multiples of π /4. This weakness holds for Hessian-based detectors in general. Nevertheless, the detectors still perform well, and the slight decrease in performance does not outweigh the advantage of fast convolutions brought by the discretization and cropping.

In order to calculate the determinant of the Hessian matrix, first we need to apply convolution with Gaussian kernel, then second-order derivative. After Lowe's success with LoG approximations(SIFT), SURF pushes the approximation(both convolution and second-order derivative) even further with box filters. These approximate second-order Gaussian derivatives and can be evaluated at a very low computational cost using integral images and independently of size, and this is part of the reason why SURF is fast.



Gaussian partial derivative in xy



Gaussian partial derivative in y

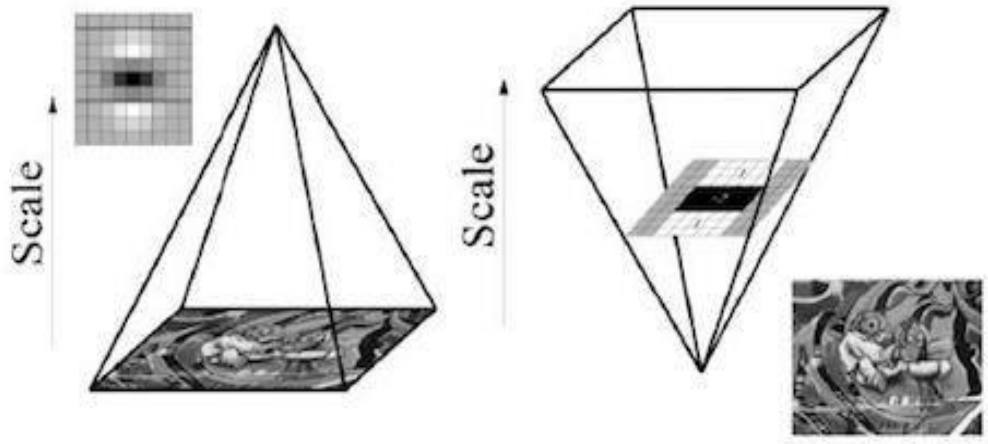
The 9 × 9 box filters in the above images are approximations for Gaussian second order derivatives with σ = 1.2. We denote these approximations by Dxx, Dyy, and Dxy. Now we can represent the determinant of the Hessian (approximated) as:

$$\det(\mathcal{H}_{\text{approx}}) = D_{xx}D_{yy} - (wD_{xy})^2.$$

w=0.9 (Bay's suggestion)

Scale-space representation

Scale spaces are usually implemented as image pyramids. The images are repeatedly smoothed with a Gaussian and subsequently sub-sampled in order to achieve a higher level of the pyramid. Due to the use of box filters and integral images, surf does not have to iteratively apply the same filter to the output of a previously filtered layer but instead can apply such filters of any size at exactly the same speed directly on the original image, and even in parallel. Therefore, the scale space is analyzed by up-scaling the filter size(9×9 → 15×15 → 21×21 → 27×27, etc) rather than iteratively reducing the image size. So for each new octave, the filter size increase is doubled simultaneously the sampling intervals for the extraction of the interest points(σ) can be doubled as well which allow the up-scaling of the filter at constant cost. In order to localize interest points in the image and over scales, a nonmaximum suppression in a 3 × 3 × 3 neighborhood is applied.



Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right).

Feature Description

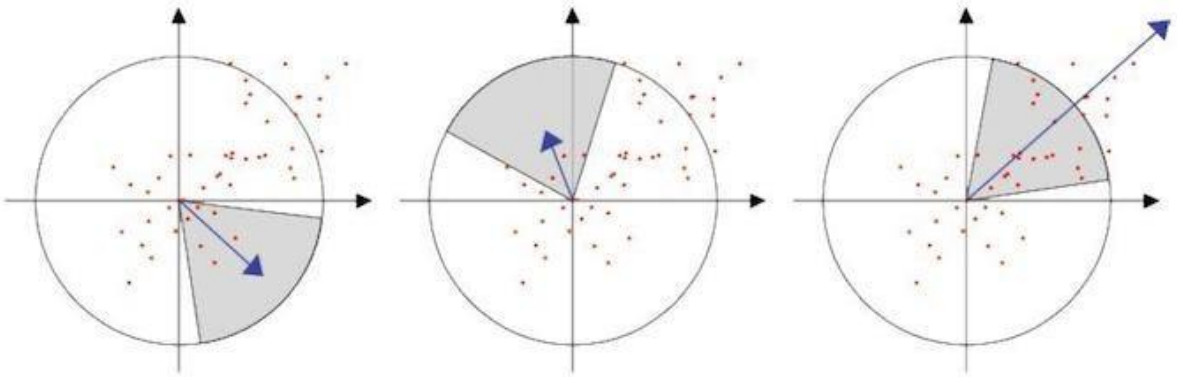
The creation of SURF descriptor takes place in two steps. The first step consists of fixing a reproducible orientation based on information from a circular region around the keypoint. Then, we construct a square region aligned to the selected orientation and extract the SURF descriptor from it.

Orientation Assignment

In order to be invariant to rotation, surf tries to identify a reproducible orientation for the interest points. For achieving this:

1. Surf first calculate the Haar-wavelet responses in x and y-direction, and this in a circular neighborhood of radius 6s around the keypoint, with s the scale at which the keypoint was detected. Also, the sampling step is scale dependent and chosen to be s, and the wavelet responses are computed at that current scale s. Accordingly, at high scales the size of the wavelets is big. Therefore integral images are used again for fast filtering.

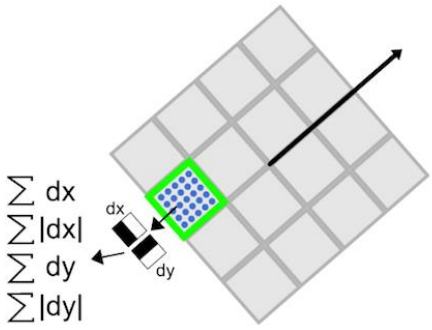
2. Then we calculate the sum of vertical and horizontal wavelet responses in a scanning area, then change the scanning orientation (add $\pi/3$), and re-calculate, until we find the orientation with largest sum value, this orientation is the main orientation of feature descriptor.



Descriptor Components

Now it's time to extract the descriptor

1. The first step consists of constructing a square region centered around the keypoint and oriented along the orientation we already got above. The size of this window is 20s.



2. Then the region is split up regularly into smaller 4 × 4 square sub-regions. For each sub-region, we compute a few simple features at 5×5 regularly spaced sample points. For reasons of simplicity, we call **dx** the Haar wavelet response in the horizontal direction and **dy** the Haar wavelet response in the vertical direction (filter size 2s). To increase the robustness towards geometric deformations and localization errors, the responses dx and **dy** are first weighted with a Gaussian ($\sigma = 3.3s$) centered at the keypoint.

Then, the wavelet responses **dx** and **dy** are summed up over each subregion and form a first set of entries to the feature vector. In order to bring in information about the polarity of the intensity changes, we also extract the sum of the absolute values of the responses, **|dx|** and **|dy|**. Hence, each sub-region has a four-dimensional descriptor vector v for its underlying intensity structure **V = (∑ dx, ∑ dy, ∑|dx|, ∑|dy|)**. This results in a descriptor vector for all 4×4 sub-regions of **length 64**(In **Sift**, our descriptor is the **128-D vector**, so this is part of the reason that SURF is faster than Sift).

Principal Component Analysis

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

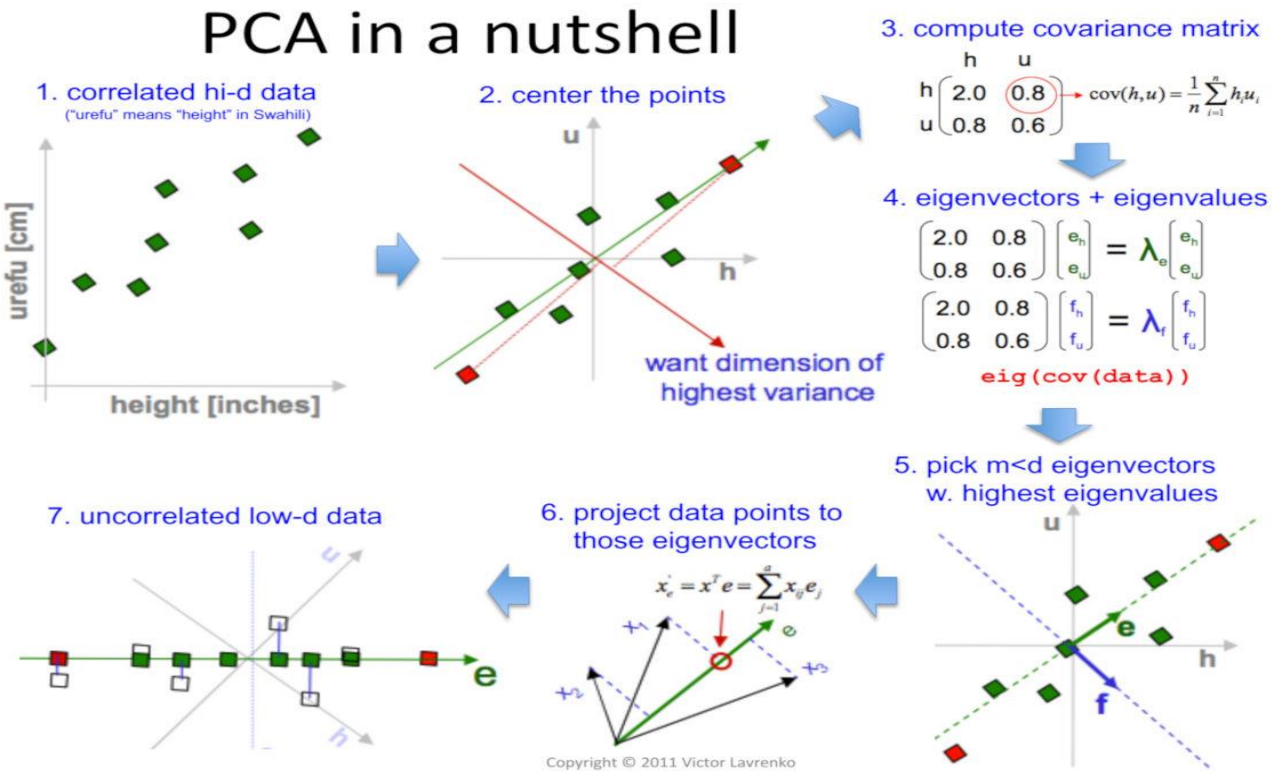
PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.



Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. **Getting the dataset** Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure** Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
3. **Standardizing the data** In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance. If the importance of features is independent of the variance of the feature, then we will divide

each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

4. **Calculating the Covariance of Z** To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

5. **Calculating the Eigen Values and Eigen Vectors** Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. **Sorting the Eigen Vectors** In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components** Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. **Remove less or unimportant features from the new dataset.** The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**

- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Linear Discriminant Analysis (LDA)

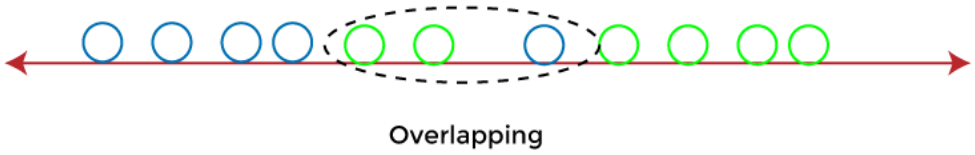
Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

What is Linear Discriminant Analysis (LDA)?

Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

Linear Discriminant analysis is one of the most popular dimensionality reduction techniques used for supervised classification problems in machine learning. It is also considered a pre-processing step for modeling differences in ML and applications of pattern classification.

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.

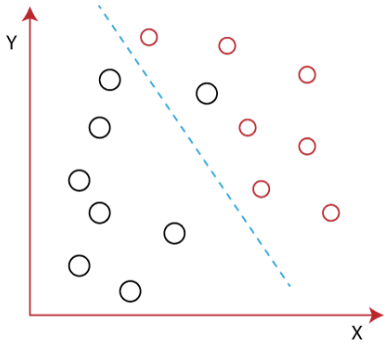


Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.

Example:

Let's assume we have to classify two different classes having two sets of data points in a 2-dimensional plane as shown below image:



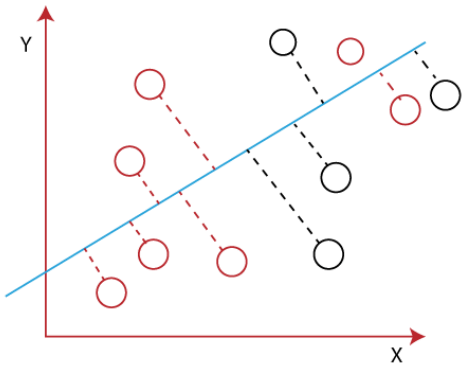
However, it is impossible to draw a straight line in a 2-d plane that can separate these data points efficiently but using linear Discriminant analysis; we can dimensionally reduce the 2-D plane into the 1-D plane. Using this technique, we can also maximize the separability between multiple classes.

How Linear Discriminant Analysis (LDA) works?

Linear Discriminant analysis is used as a dimensionality reduction technique in machine learning, using which we can easily transform a 2-D and 3-D graph into a 1-dimensional plane.

Let's consider an example where we have two classes in a 2-D plane having an X-Y axis, and we need to classify them efficiently. As we have already seen in the above example that LDA enables us to draw a straight line that can completely separate the two classes of the data points. Here, LDA uses an X-Y axis to create a new axis by separating them using a straight line and projecting data onto a new axis.

Hence, we can maximize the separation between these classes and reduce the 2-D plane into 1-D.



To create a new axis, Linear Discriminant Analysis uses the following criteria:

- It maximizes the distance between means of two classes.
- It minimizes the variance within the individual class.

Using the above two conditions, LDA generates a new axis in such a way that it can maximize the distance between the means of the two classes and minimizes the variation within each class.

In other words, we can say that the new axis will increase the separation between the data points of the two classes and plot them onto the new axis.

Why LDA?

- Logistic Regression is one of the most popular classification algorithms that perform well for binary classification but falls short in the case of multiple classification problems with well-separated classes. At the same time, LDA handles these quite efficiently.
- LDA can also be used in data pre-processing to reduce the number of features, just as PCA, which reduces the computing cost significantly.
- LDA is also used in face detection algorithms. In Fisherfaces, LDA is used to extract useful data from different faces. Coupled with eigenfaces, it produces effective results.

Drawbacks of Linear Discriminant Analysis (LDA)

Although, LDA is specifically used to solve supervised classification problems for two or more classes which are not possible using logistic regression in machine learning. But LDA also fails in some cases where the Mean of the distributions is shared. In this case, LDA fails to create a new axis that makes both the classes linearly separable.

To overcome such problems, we use **non-linear Discriminant analysis** in machine learning.

Extension to Linear Discriminant Analysis (LDA)

Linear Discriminant analysis is one of the most simple and effective methods to solve classification problems in machine learning. It has so many extensions and variations as follows:

1. **Quadratic Discriminant Analysis (QDA):** For multiple input variables, each class deploys its own estimate of variance.

2. **Flexible Discriminant Analysis (FDA):** it is used when there are non-linear groups of inputs are used, such as splines.

3. **Flexible Discriminant Analysis (FDA):** This uses regularization in the estimate of the variance (actually covariance) and hence moderates the influence of different variables on LDA.

Real-world Applications of LDA

Some of the common real-world applications of Linear discriminant Analysis are given below:

- **Face Recognition** Face recognition is the popular application of computer vision, where each face is represented as the combination of a number of pixel values. In this case, LDA is used to minimize the number of features to a manageable number before going through the classification process. It generates a new template in which each dimension consists of a linear combination of pixel values. If a linear combination is generated using Fisher's linear discriminant, then it is called Fisher's face.
- **Medical** In the medical field, LDA has a great application in classifying the patient disease on the basis of various parameters of patient health and the medical treatment which is going on. On such parameters, it classifies disease as mild, moderate, or severe. This classification helps the doctors in either increasing or decreasing the pace of the treatment.
- **Customer Identification** In customer identification, LDA is currently being applied. It means with the help of LDA; we can easily identify and select the features that can specify the group of customers who are likely to purchase a specific product in a shopping mall. This can be helpful when we want to identify a group of customers who mostly purchase a product in a shopping mall.
- **For Predictions** LDA can also be used for making predictions and so in decision making. For example, "will you buy this product" will give a predicted result of either one or two possible classes as a buying or not.
- **In Learning** Nowadays, robots are being trained for learning and talking to simulate human work, and it can also be considered a classification problem. In this case, LDA builds similar groups on the basis of different parameters, including pitches, frequencies, sound, tunes, etc.

Difference between Linear Discriminant Analysis and PCA

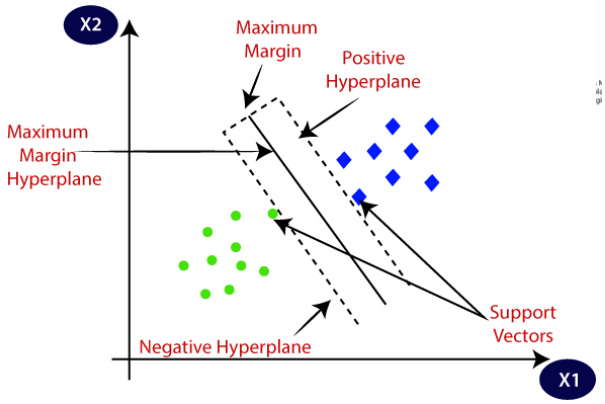
- PCA is an unsupervised algorithm that does not care about classes and labels and only aims to find the principal components to maximize the variance in the given dataset. At the same time, LDA is a supervised algorithm that aims to find the linear discriminants to represent the axes that maximize separation between different classes of data.
- LDA is much more suitable for multi-class classification tasks compared to PCA. However, PCA is assumed to be an as good performer for a comparatively small sample size.
- Both LDA and PCA are used as dimensionality reduction techniques, where PCA is first followed by LDA.

Support Vector Machine Algorithm

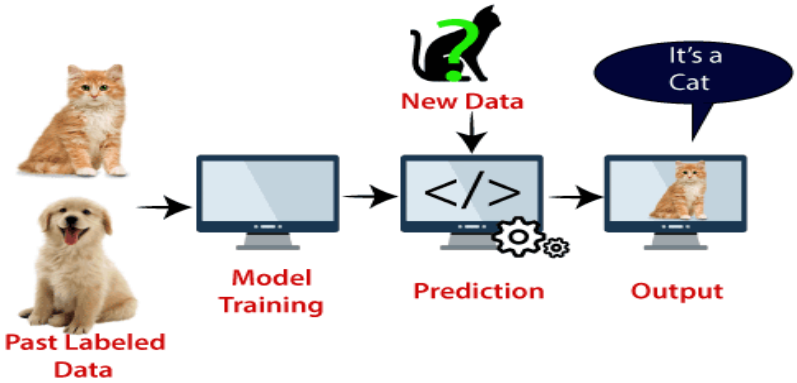
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

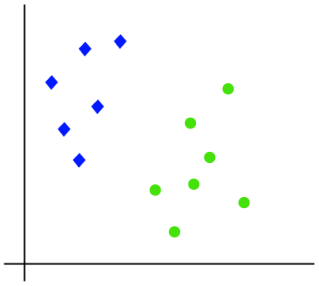
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

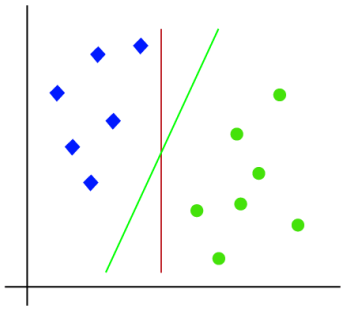
How does SVM works?

Linear SVM:

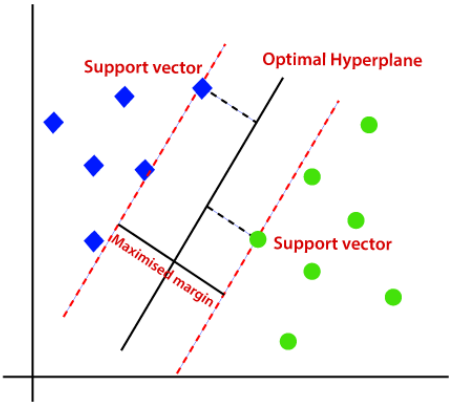
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

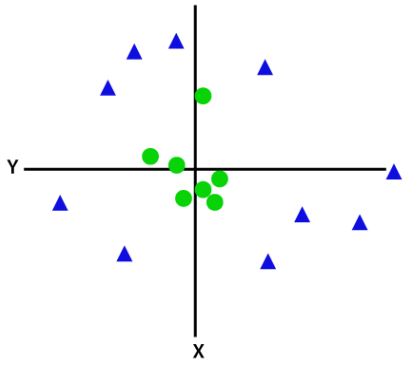


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

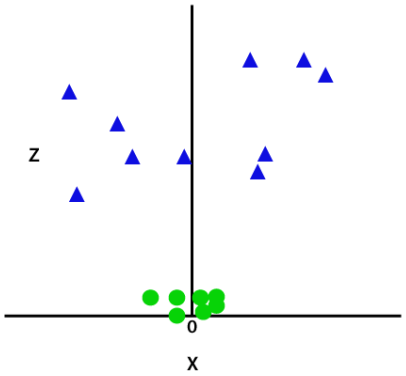
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



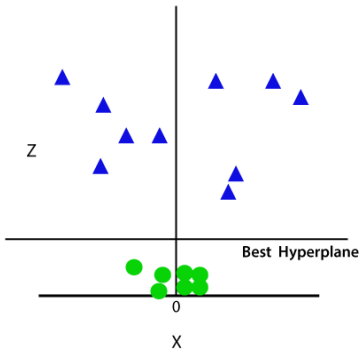
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

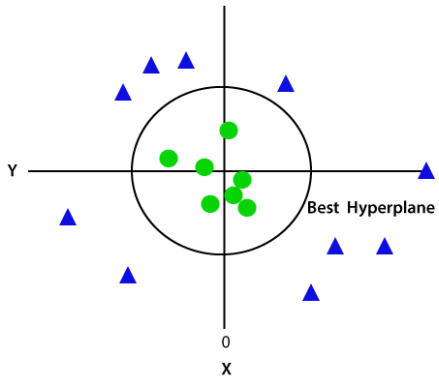
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

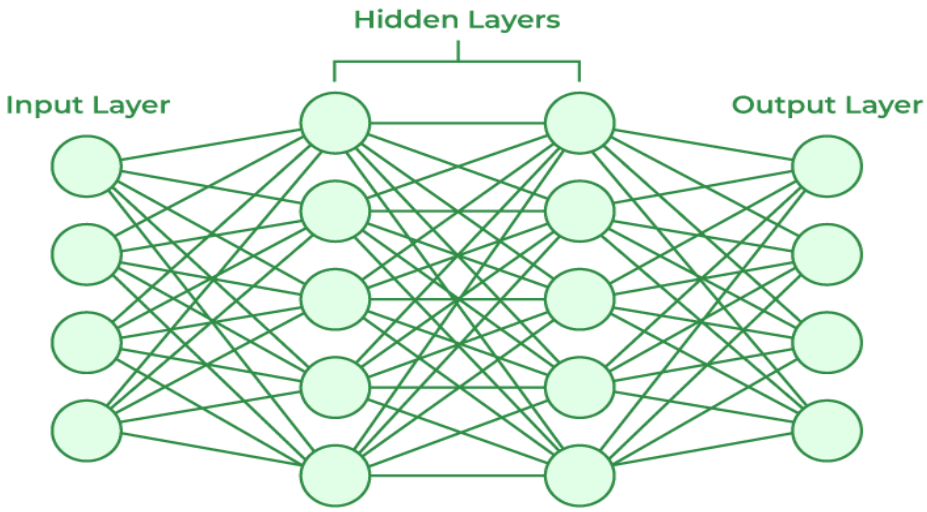
As you read this content, which organ in your body is thinking about it? It's the brain of course! But do you know how the brain works? Well, it has neurons or nerve cells that are the primary units of both the brain and the nervous system. These neurons receive sensory input from the outside world which they process and then provide the output which might act as the input to the next neuron.

Each of these neurons is connected to other neurons in complex arrangements at synapses. Now, are you wondering how this is related to **Artificial Neural Networks**? Well, Artificial Neural Networks are modeled after the neurons in the human brain. Let's check out what they are in detail and how they learn information.

Artificial Neural Networks

Artificial Neural Networks contain artificial neurons which are called **units**. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.



Neural Networks Architecture

The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

Artificial neurons vs Biological neurons

The concept of artificial neural networks comes from biological neurons found in animal brains So they share a lot of similarities in structure and function wise.

- **Structure**: The structure of artificial neural networks is inspired by biological neurons. A biological neuron has a cell body or soma to process the impulses, dendrites to receive them, and an axon that transfers them to other neurons. The input nodes of artificial neural networks receive input signals, the hidden layer nodes compute these input signals, and the output layer nodes compute the final output by processing the hidden layer's results using activation functions.

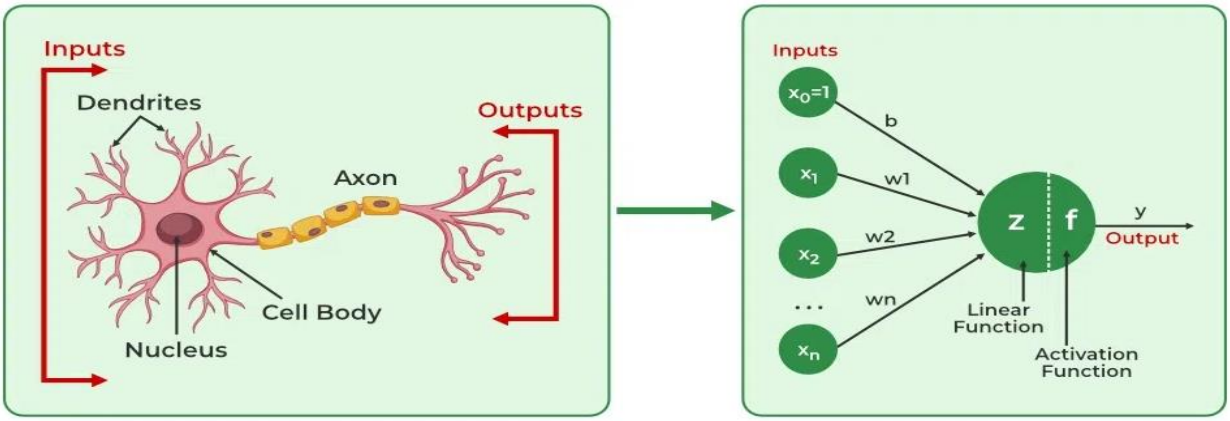
Biological Neuron	Artificial Neuron
Dendrite	Inputs
Cell nucleus or Soma	Nodes
Synapses	Weights
Axon	Output

- **Synapses**: Synapses are the links between biological neurons that enable the transmission of impulses from dendrites to the cell body. Synapses are the weights that join the one-layer nodes to the next-layer nodes in artificial neurons. The strength of the links is determined by the weight value.
- **Learning**: In biological neurons, learning happens in the cell body nucleus or soma, which has a nucleus that helps to process the impulses. An action potential is produced and travels through the axons if the impulses are powerful enough to reach the threshold. This becomes possible by synaptic plasticity, which represents the ability of synapses to become stronger or weaker over time in reaction to changes in their activity. In artificial neural networks, backpropagation is a technique used for learning, which adjusts the weights between nodes according to the error or differences between predicted and actual outcomes.

Biological Neuron	Artificial Neuron
Synaptic plasticity	Backpropagations

- **Activation**: In biological neurons, activation is the firing rate of the neuron which happens when the impulses are strong enough to reach the threshold. In artificial neural networks, A mathematical function known as an activation function maps the input to the output, and executes activations.

Biological neurons to artificial neurons



How do Artificial Neural Networks learn?

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then back-propagation is used to adjust whatever it has learned during training. Backpropagation is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates.

What are the types of Artificial Neural Networks?

- **Feedforward Neural Network:** The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist. So the feedforward neural network has a front-propagated wave only and usually does not have backpropagation.
- **Convolutional Neural Network:** A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit. But a CNN has one or more than one convolutional layer that uses a convolution operation on the input and then passes the result obtained in the form of output to the next layer. CNN has applications in speech and image processing which is particularly useful in computer vision.
- **Modular Neural Network:** A Modular Neural Network contains a collection of different neural networks that work independently towards obtaining the output with no interaction between them. Each of the different neural networks performs a different sub-task by obtaining unique inputs compared to other networks. The advantage of this modular neural network is that it breaks down a large and complex computational process into smaller components, thus decreasing its complexity while still obtaining the required output.

- **Radial basis function Neural Network:** Radial basis functions are those functions that consider the distance of a point concerning the center. RBF functions have two layers. In the first layer, the input is mapped into all the Radial basis functions in the hidden layer and then the output layer computes the output in the next step. Radial basis function nets are normally used to model the data that represents any underlying trend or function.
- **Recurrent Neural Network:** The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.

Applications of Artificial Neural Networks

1. **Social Media:** Artificial Neural Networks are used heavily in Social Media. For example, let's take the **'People you may know'** feature on Facebook that suggests people that you might know in real life so that you can send them friend requests. Well, this magical effect is achieved by using Artificial Neural Networks that analyze your profile, your interests, your current friends, and also their friends and various other factors to calculate the people you might potentially know. Another common application of Machine Learning in social media is **facial recognition**. This is done by finding around 100 reference points on the person's face and then matching them with those already available in the database using convolutional neural networks.
2. **Marketing and Sales:** When you log onto E-commerce sites like Amazon and Flipkart, they will recommend your products to buy based on your previous browsing history. Similarly, suppose you love Pasta, then Zomato, Swiggy, etc. will show you restaurant recommendations based on your tastes and previous order history. This is true across all new-age marketing segments like Book sites, Movie services, Hospitality sites, etc. and it is done by implementing **personalized marketing**. This uses Artificial Neural Networks to identify the customer likes, dislikes, previous shopping history, etc., and then tailor the marketing campaigns accordingly.
3. **Healthcare**: Artificial Neural Networks are used in Oncology to train algorithms that can identify cancerous tissue at the microscopic level at the same accuracy as trained physicians. Various rare diseases may manifest in physical characteristics and can be identified in their premature stages by using **Facial Analysis** on the patient photos. So the full-scale implementation of Artificial Neural Networks in the healthcare environment can only enhance the diagnostic abilities of medical experts and ultimately lead to the overall improvement in the quality of medical care all over the world.
4. **Personal Assistants:** I am sure you all have heard of Siri, Alexa, Cortana, etc., and also heard them based on the phones you have!!! These are personal assistants and an example of speech recognition that uses **Natural Language Processing** to interact with the users and formulate a response accordingly. Natural Language Processing uses artificial neural networks that are made to handle many tasks of these personal assistants such as managing the language syntax, semantics, correct speech, the conversation that is going on, etc.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers:

1. Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).

2. Hidden Layer: The input from the Input layer is then fed into the hidden layer. There can be many hidden layers depending on our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of the output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.

3. Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

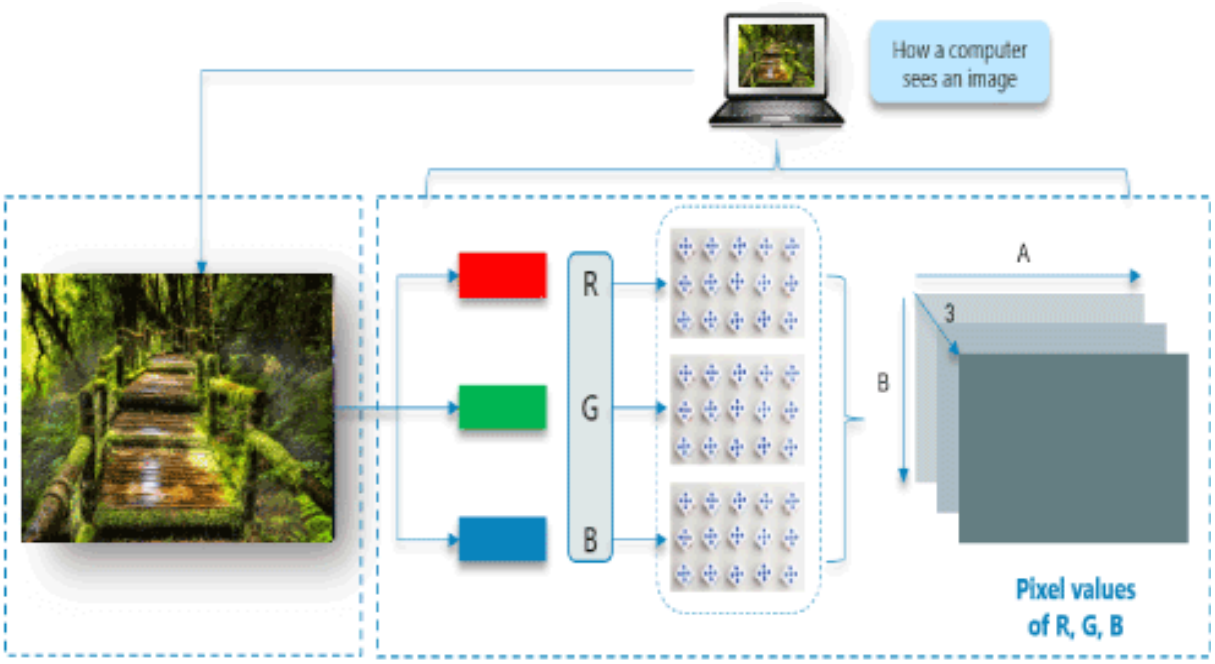
The intelligence of neural networks is unnatural. While the artificial neural network is researched as early in the 1960s by Rosenblatt, it was only in late 2000s when deep learning using neural networks took off. The key enabler was the scale of computation power and datasets with Google developing research into deep learning. In July 2012, researchers at Google disclosed an advanced neural network to a series of unlabeled, static images sliced from YouTube videos.

For example,

Consider this image of Nature, upon first glance; we will see a lot of buildings and colors.

How Does a Computer read an image?

The image is broken into 3 color-channels which is Red, Green, and Blue. Each of these color channels is mapped to the image's pixel.



Some neurons fires when exposed to vertices edges and some when shown horizontal or diagonal edges. CNN utilizes spatial correlations which exist with the input data. Each concurrent layer of the neural network connects some input neurons. This region is called a local receptive field. The local receptive field focuses on hidden neurons.

The hidden neuron processes the input data inside the mentioned field, not realizing the changes outside the specific boundary.

Convolutional Neural Networks have the following 4 layers:

- Convolutional
- ReLU Layer
- Pooling
- Fully Connected

Convolutional layer

Convolution layer is the first layer to derive features from the input image. The convolutional layer conserves the relationship between pixels by learning image features using a small square of input data. It is the mathematical operation which takes two inputs such as image matrix and kernel or any filter.

- The dimension of image matrix is h×w×d.
- The dimension of any filter is fh×fw×d.
- The dimension of output is (h-fh+1)×(w-fw+1)×1.

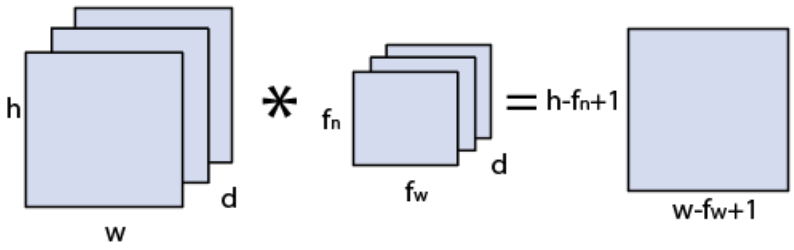


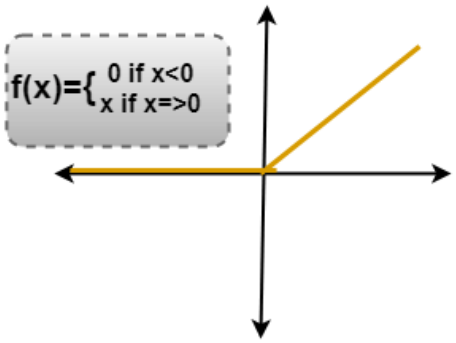
Image matrix multiplies kernl or filter matrix

ReLU Layer

Rectified Linear unit(ReLU) transform functions only activates a node if the input is above a certain quantity. While the data is below zero, the output is zero, but when the input rises above a certain threshold. It has a linear relationship with the dependent variable.

In this layer, we remove every negative value from the filtered images and replaces them with zeros.

It is happening to avoid the values from adding up to zero.



Pooling Layer

Pooling layer plays a vital role in pre-processing of any image. Pooling layer reduces the number of the parameter when the image is too large. Pooling is "downscaling" of the image achieved from previous layers. It can be compared to shrink an image to reduce the image's density. Spatial pooling is also called downsampling and subsampling, which reduce the dimensionality of each map but remains essential information. These are the following types of spatial pooling.

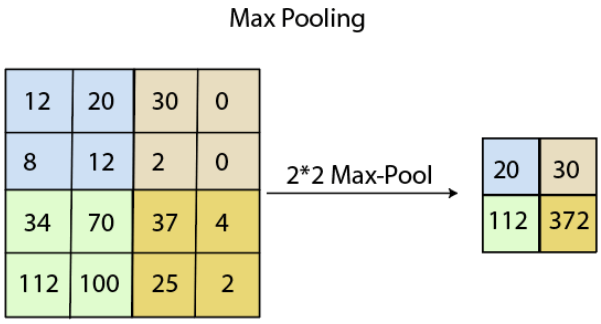
We do this by implementing the following 4 steps:

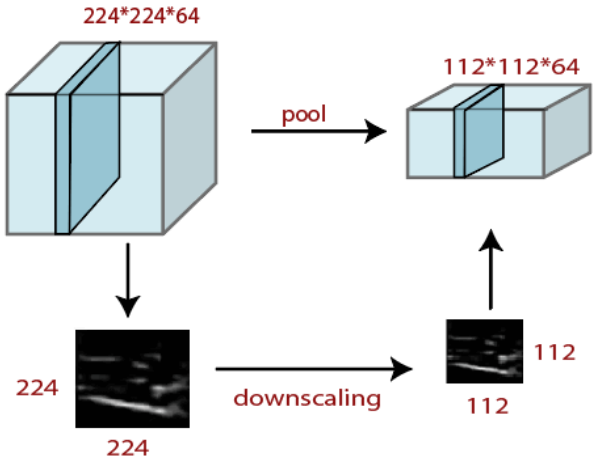
- Pick a window size (usually 2 or 3)
- Pick a stride (usually 2)
- Walk your window across your filtered images
- From each window, take the maximum value

Max Pooling

Max pooling is a sample-based discretization process. The main objective of max-pooling is to downscale an input representation, reducing its dimension and allowing for the assumption to be made about feature contained in the sub-region binned.

Max pooling is complete by applying a max filter in non-overlapping sub-regions of initial representation.





Average Pooling

Down-scaling will perform by average pooling by dividing the input into rectangular pooling regions and computing the average values of each area.

Syntax

1. layer = averagePooling2dLayer(pool Size)
2. layer = averagePooling2dLayer(poolSize, Name, Value)

Sum Pooling

The sub-region for sum pooling and mean pooling are set the same as for max-pooling but instead of using the max function we use sum or mean.

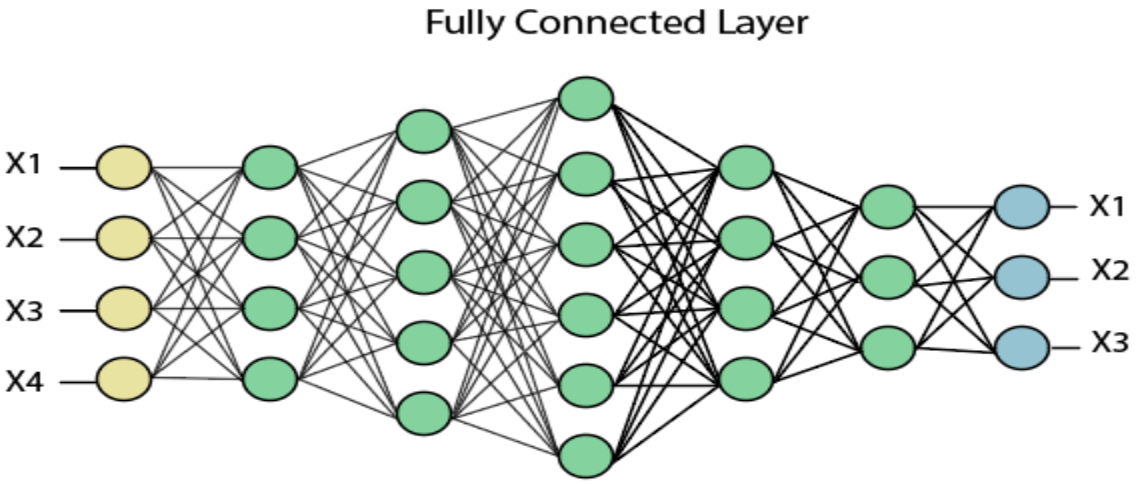
In this layer we shrink the image stack into a smaller size steps;

1. Pick a window size (usually 2 or 3)
2. Pick a stride (usually 2)
3. Walk our window across our filtered images.
4. From each window, take the maximum value.

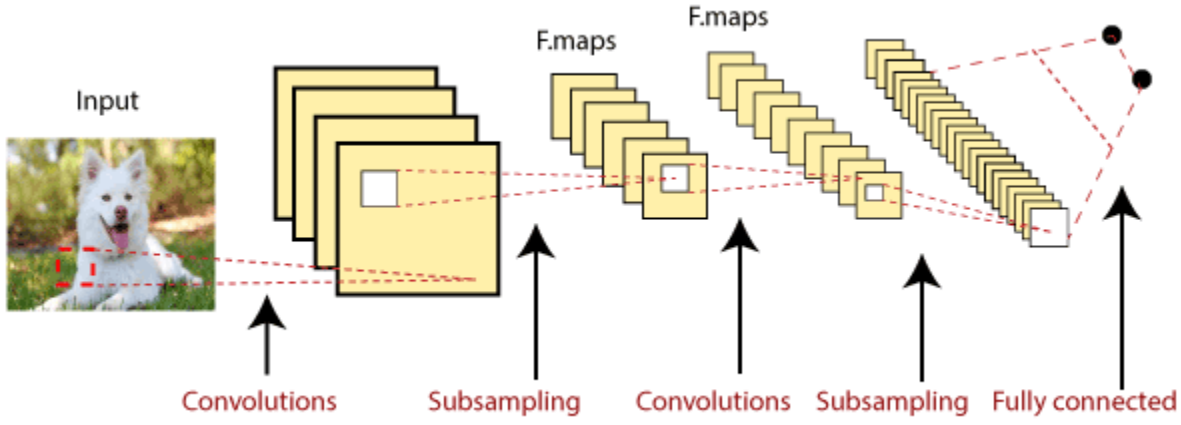
Performing pooling with a window size two and stride 2.

Fully Connected (Dense) Layer

The fully connected layer (dense layer) is a layer where the input from other layers will be depressed into the vector. It will transform the output into any desired number of classes into the network.



In the above diagram, the map matrix is converted into the vector such as x1, x2, x3... xn with the help of a fully connected layer. We will combine features to create any model and apply activation function like as softmax or sigmoid to classify the outputs as a car, dog, truck, etc.



This is the final where the actual classification happens.

Region-based Convolutional Neural Network (R-CNN)

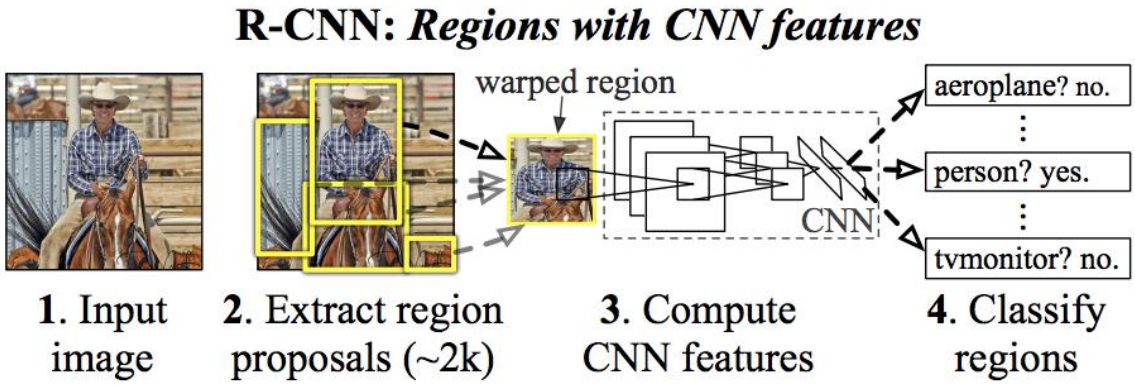
While CNNs had already gained prominence for image classification tasks, object detection required a more intricate solution. This is where the R-CNN algorithm entered the scene, offering a novel approach to this complex problem.

R-CNN paved the way for subsequent innovations in object detection, including Fast R-CNN, Faster R-CNN, and Mask R-CNN, each building upon and enhancing the capabilities of its predecessor. To grasp the nuances of these advanced R-CNN variants, it is essential to establish a solid foundation in the original R-CNN architecture.

In this blog post, we will delve deep into the workings of R-CNN, providing a comprehensive understanding of its inner workings.

What is R-CNN? How Does R-CNN Work?

Region-based Convolutional Neural Network (R-CNN) is a type of deep learning architecture used for object detection in computer vision tasks. RCNN was one of the pioneering models that helped advance the object detection field by combining the power of convolutional neural networks and region-based approaches.



The R-CNN Architecture, featuring the steps of taking in an input image, extracting region proposals, computing CNN features, and classifying regions. Source

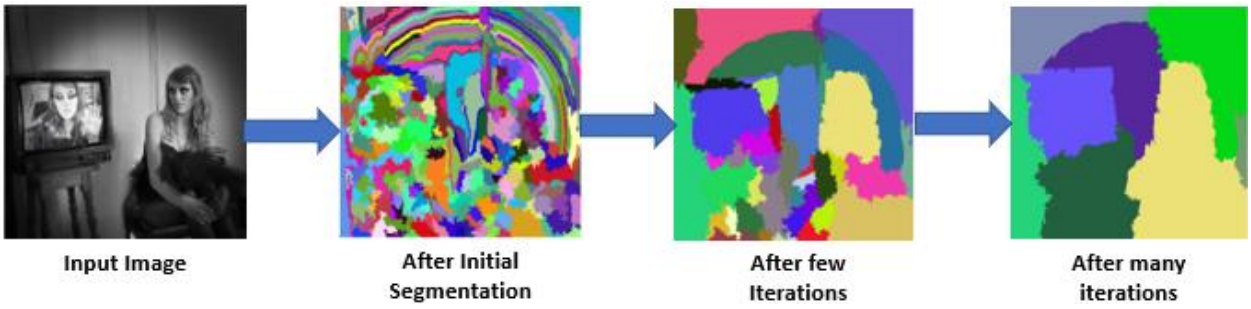
Let's dive deeper into how R-CNN works, step by step.

Region Proposal

R-CNN starts by dividing the input image into multiple regions or subregions. These regions are referred to as "region proposals" or "region candidates." The region proposal step is responsible for generating a set of potential regions in the image that are likely to contain objects. R-CNN does not generate these proposals itself; instead, it relies on external methods like Selective Search or EdgeBoxes to generate region proposals.

Selective Search, for example, operates by merging or splitting segments of the image based on various image cues like color, texture, and shape to create a diverse set of region proposals.

Below we show how Selective Search works, which shows an input image, then an image with many segmented masks, then fewer masks, then masks that comprise the main components in the image.



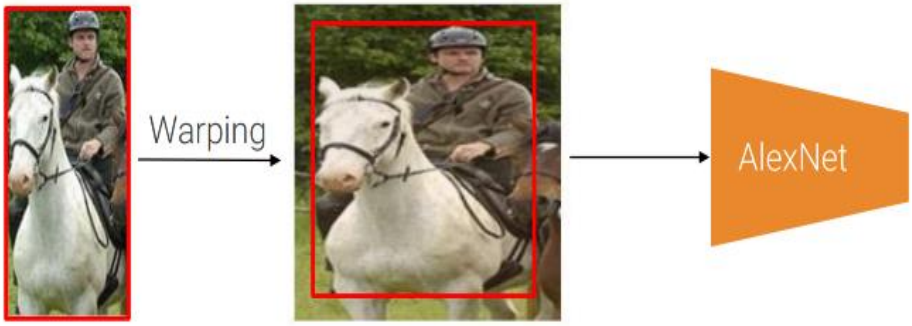
Example of the Selective Search algorithm applied to an image. Source.

Feature Extraction

Once the region proposals are generated, approximately 2,000 regions are extracted and anisotropically warped to a consistent input size that the CNN expects (e.g., 224x224 pixels) and then it is passed through the CNN to extract features.

Before warping, the region size is expanded to a new size that will result in 16 pixels of context in the warped frame. The CNN used is AlexNet and it is typically fine-tuned on a large dataset like ImageNet for generic feature representation.

The output of the CNN is a high-dimensional feature vector representing the content of the region proposal.



An example of how warping works in R-CNN. Note the addition of 16 pixels of context in the middle image.

Object Classification

The extracted feature vectors from the region proposals are fed into a separate machine learning classifier for each object class of interest. R-CNN typically uses Support Vector Machines (SVMs)

for classification. For each class, a unique SVM is trained to determine whether or not the region proposal contains an instance of that class.

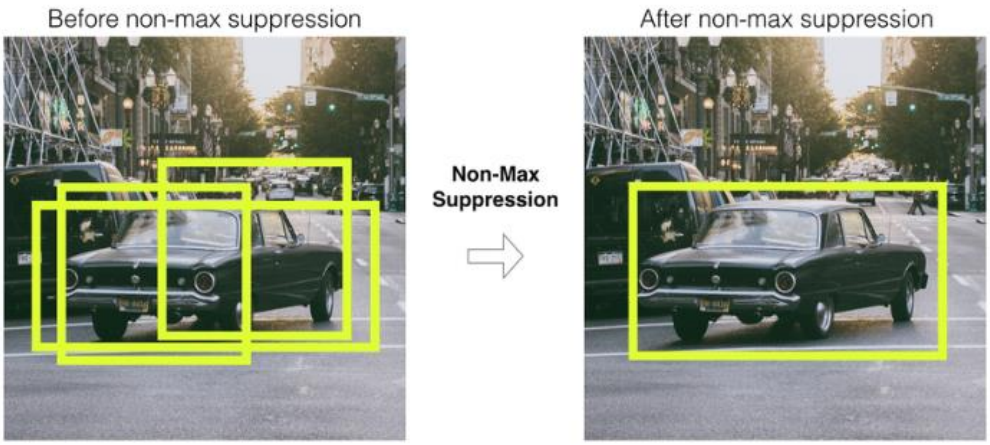
During training, positive samples are regions that contain an instance of the class. Negative samples are regions that do not.

Bounding Box Regression

In addition to classifying objects, R-CNN also performs bounding box regression. For each class, a separate regression model is trained to refine the location and size of the bounding box around the detected object. The bounding box regression helps improve the accuracy of object localization by adjusting the initially proposed bounding box to better fit the object's actual boundaries.

Non-Maximum Suppression (NMS)

After classifying and regressing bounding boxes for each region proposal, R-CNN applies non-maximum suppression to eliminate duplicate or highly overlapping bounding boxes. NMS ensures that only the most confident and non-overlapping bounding boxes are retained as final object detections.



Example of Non-Maximum Suppression. Source.

R-CNN Strengths and Disadvantages

Now that we've covered what R-CNN is and how it works, let’s delve into the strengths and weaknesses of this popular object detection framework. Understanding the strengths and disadvantages of R-CNN can help you make an informed decision when choosing an approach for your specific computer vision tasks.

Strengths of R-CNN

Below are a few of the key strengths of the R-CNN architecture.

- Accurate Object Detection: R-CNN provides accurate object detection by leveraging region-based convolutional features. It excels in scenarios where precise object localization and recognition are crucial.
- Robustness to Object Variations: R-CNN models can handle objects with different sizes, orientations, and scales, making them suitable for real-world scenarios with diverse objects and complex backgrounds.
- Flexibility: R-CNN is a versatile framework that can be adapted to various object detection tasks, including instance segmentation and object tracking. By modifying the final layers of the network, you can tailor R-CNN to suit your specific needs.

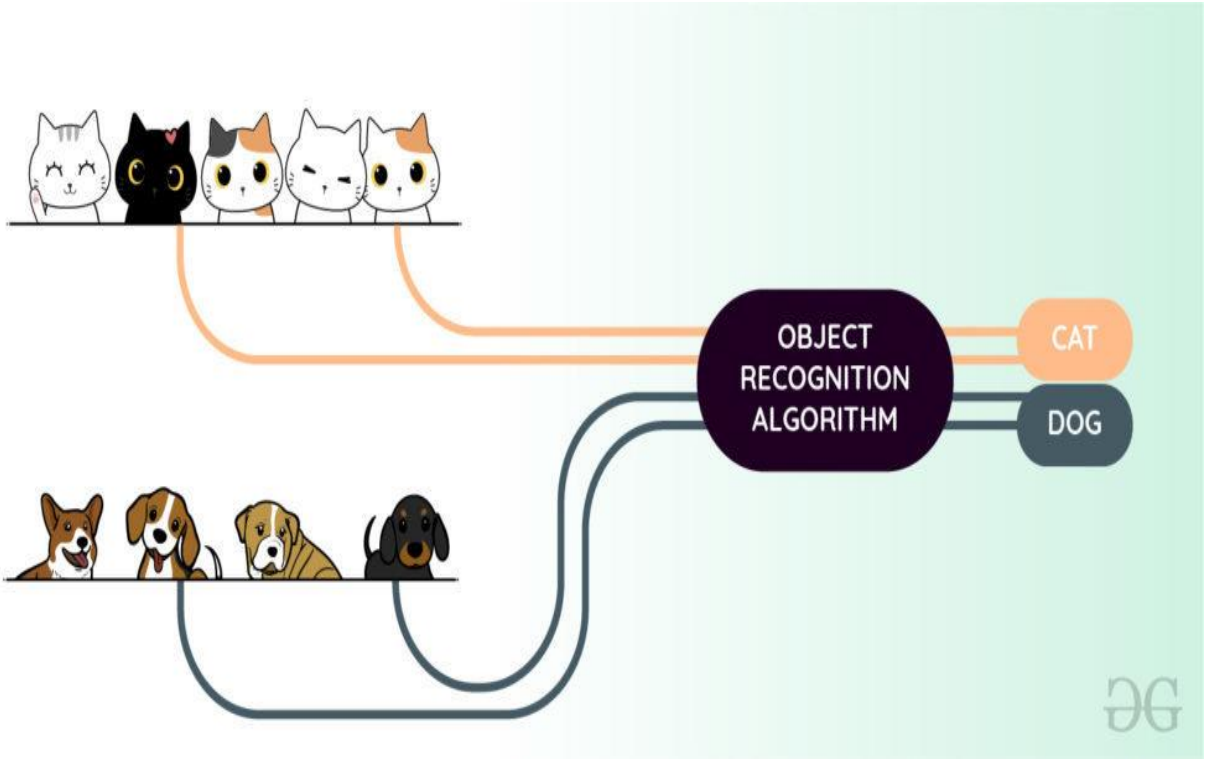
Disadvantages of R-CNN

Below are a few disadvantages of the R-CNN architecture.

- Computational Complexity: R-CNN is computationally intensive. It involves extracting region proposals, applying a CNN to each proposal, and then running the extracted features through a classifier. This multi-stage process can be slow and resource-demanding.
- Slow Inference: Due to its sequential processing of region proposals, R-CNN is relatively slow during inference. Real-time applications may find this latency unacceptable.
- Overlapping Region Proposals: R-CNN may generate multiple region proposals that overlap significantly, leading to redundant computation and potentially affecting detection performance.
- R-CNN is Not End-to-End: Unlike more modern object detection architectures like Faster R-CNN, R-CNN is not an end-to-end model. It involves separate modules for region proposal and classification, which can lead to suboptimal performance compared to models that optimize both tasks jointly.

Object Detection:

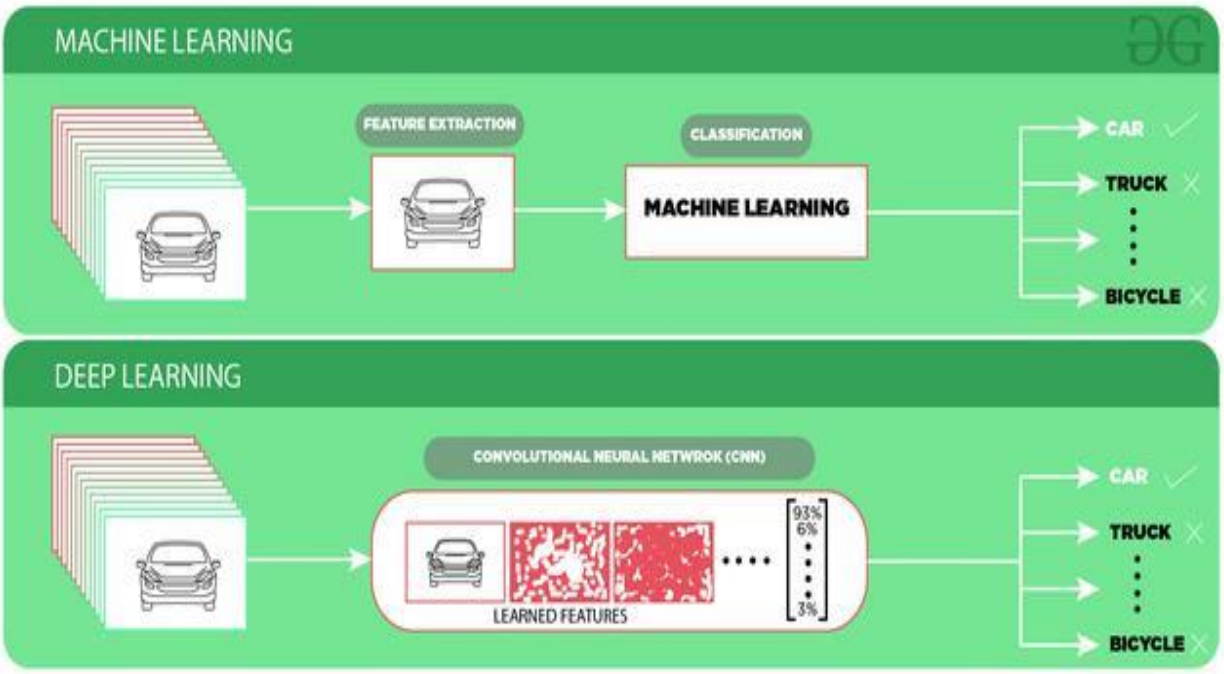
Object Detection is the technique of identifying the object present in images and videos. It is one of the most important applications of machine learning and deep learning. The goal of this field is to teach machines to understand (recognize) the content of an image just like humans do.



Object Recognition

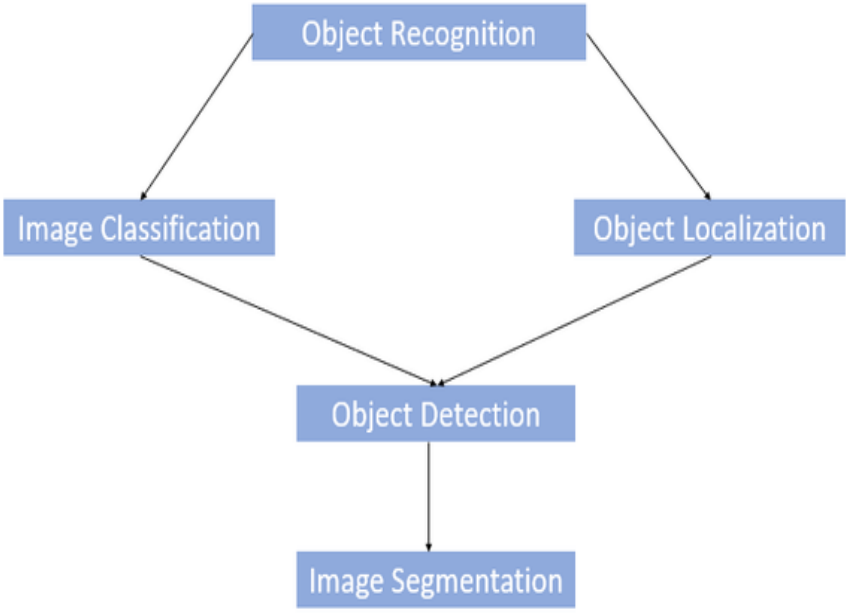
Object Recognition Using Deep Learning

Convolution Neural Network (CNN) is one of the most popular ways of doing object recognition. It is widely used and most state-of-the-art neural networks used this method for various object recognition related tasks such as image classification. This CNN network takes an image as input and outputs the probability of the different classes. If the object present in the image then it's output probability is high else the output probability of the rest of classes is either negligible or low. The advantage of Deep learning is that we don't need to do feature extraction from data as compared to machine learning.



Challenges of Object Recognition:

- Since we take the output generated by last (fully connected) layer of the CNN model is a single class label. So, a simple CNN approach will not work if more than one class labels are present in the image.
- If we want to localize the presence of an object in the bounding box, we need to try a different approach that not only outputs the class label but also outputs the bounding box locations.



Overview of tasks related to Object Recognition

Image Classification :

In Image classification, it takes an image as an input and outputs the classification label of that image with some metric (probability, loss, accuracy, etc). For Example: An image of a cat can be classified as a class label “cat” or an image of Dog can be classified as a class label “dog” with some probability.

Image Classification

Object Localization: This algorithm locates the presence of an object in the image and represents it with a bounding box. It takes an image as input and outputs the location of the bounding box in the form of (position, height, and width).

Object Detection:

Object Detection algorithms act as a combination of image classification and object localization. It takes an image as input and produces one or more bounding boxes with the class label attached to each bounding box. These algorithms are capable enough to deal with multi-class classification and localization as well as to deal with the objects with multiple occurrences.

Challenges of Object Detection:

- In object detection, the bounding boxes are always rectangular. So, it does not help with determining the shape of objects if the object contains the curvature part.
- Object detection cannot accurately estimate some measurements such as the area of an object, perimeter of an object from image.

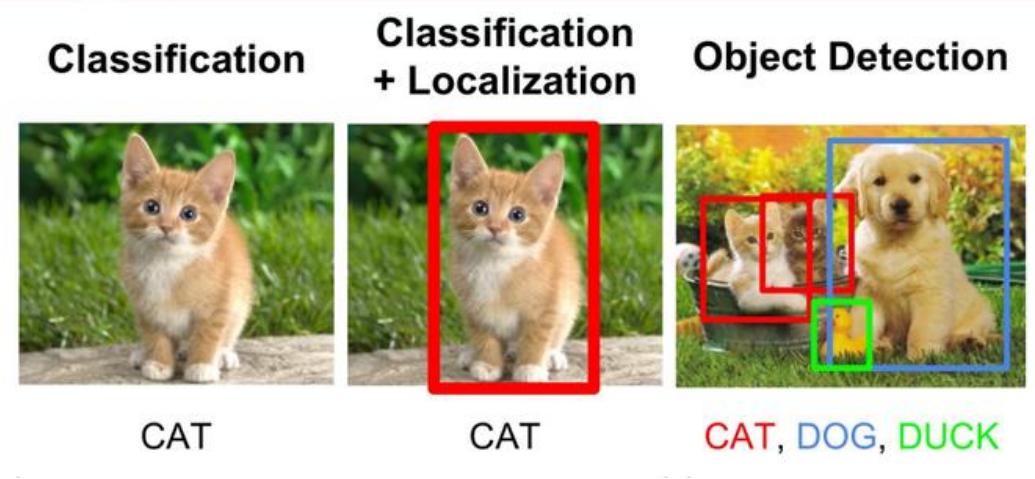


Fig Difference between classification. Localization and Detection (Source: Link)

Image Segmentation:

Image segmentation is a further extension of object detection in which we mark the presence of an object through pixel-wise masks generated for each object in the image. This technique is more granular than bounding box generation because this can helps us in determining the shape of each object present in the image because instead of drawing bounding boxes , segmentation helps to figure out pixels that are making that object. This granularity helps us in various fields such as medical image processing, satellite imaging, etc. There are many image segmentation approaches proposed recently. One of the most popular is Mask R-CNN proposed by *K He et al*. in 2017.

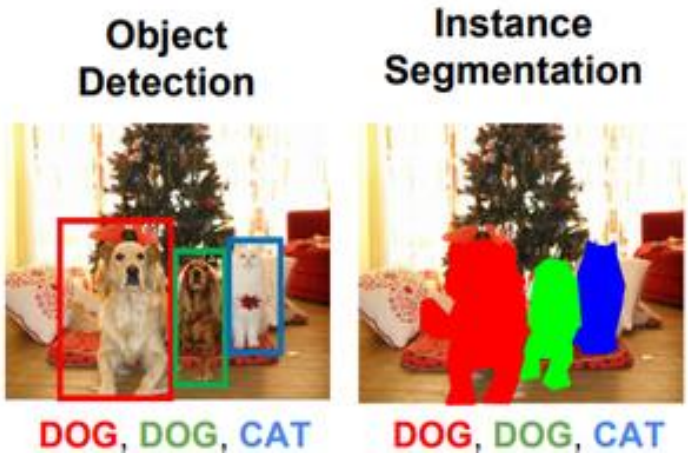
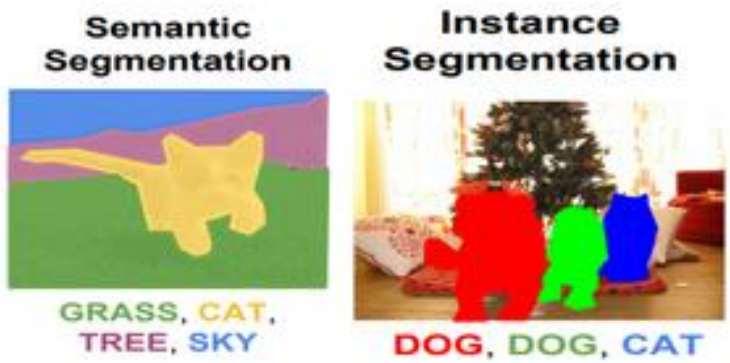


Fig. Object Detection vs Segmentation (Source: Link)

There are primarily two types of segmentation:

- **Instance Segmentation:** Multiple instances of same class are separate segments i.e. objects of same class are treated as different. Therefore, all the objects are coloured with different colour even if they belong to same class.

- **Semantic Segmentation:** All objects of same class form a single classification ,therefore , all objects of same class are coloured by same colour.



Semantic vs Instance Segmentation (Source: Link)

Applications:

The above-discussed object recognition techniques can be utilized in many fields such as:

- **Driver-less Cars:** Object Recognition is used for detecting road signs, other vehicles, etc.
- **Medical Image Processing:** Object Recognition and Image Processing techniques can help detect disease more accurately. Image segmentation helps to detect the shape of the defect present in the body . For Example, Google AI for breast cancer detection detects more accurately than doctors.
- **Surveillance and Security:** such as Face Recognition, Object Tracking, Activity Recognition, etc.