SUMMARY

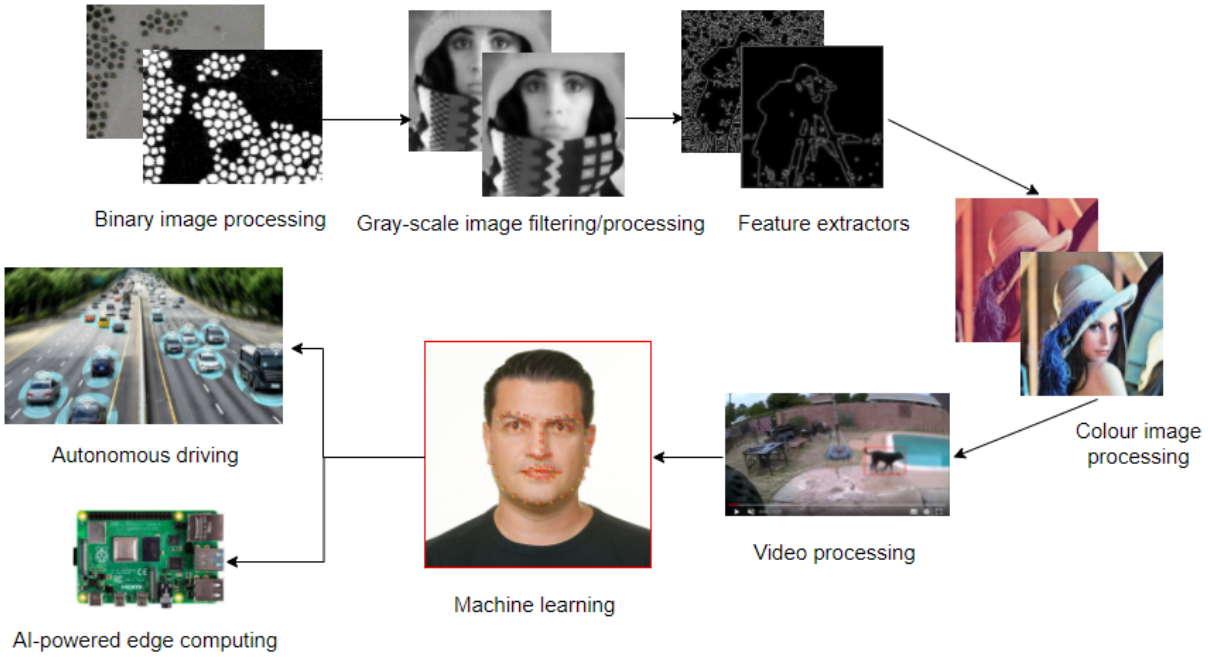
Fundamentals of Object Detection

In this module, you learnt one of the common applications of deep learning in computer vision, that is, **object detection**.

Introduction to Deep Learning

Computer vision (CV) allows a computer **to perceive** and **perform operations** or **tasks** on **digital data**.

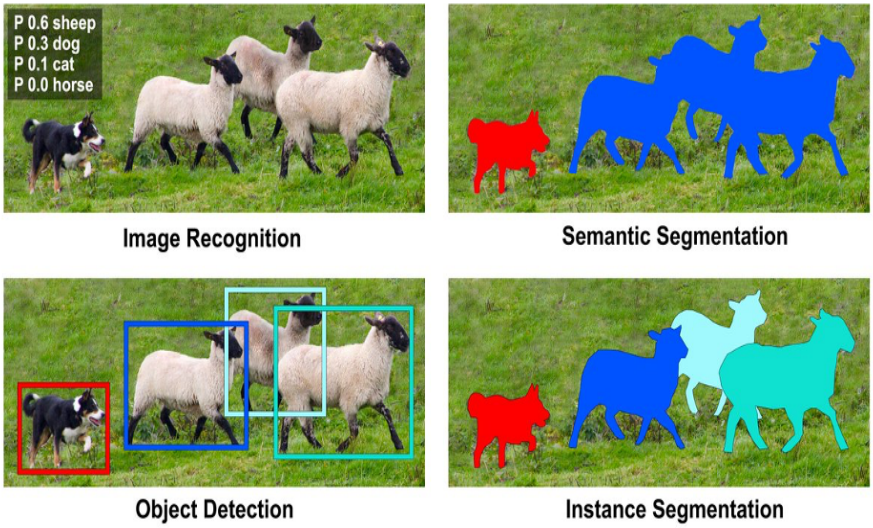
CV began in the 1960s as a simple collection of methods for processing black and white images produced by early industrial cameras. Over time, it has evolved to process greyscale and colour images (both 2D and 3D) as well as videos, utilising increasingly complex algorithms. Now, CV applications can be used for complex tasks, such as self-driving cars and face detection. The image given below shows the evolution of CV's applications.



Modern CV, in combination with machine learning (ML), is often tasked with addressing four common problems, which are:

- Image classification,
- Object detection,
- Semantic segmentation and
- Instance segmentation.

The image given below shows the applications of CV in addressing these four problems.



Introduction to Object Detection

You started this session by learning the key differences between object detection and image classification. **Image classification** algorithms will be able to categorise different objects inside an image into different classes. However, **object detection** algorithms would not only be able to classify the objects but also provide you with their locations in the image. Let's take the following image as an example to understand the difference between these algorithms better.

Let's try and understand the difference between the outputs of the two algorithms in the context of this image:

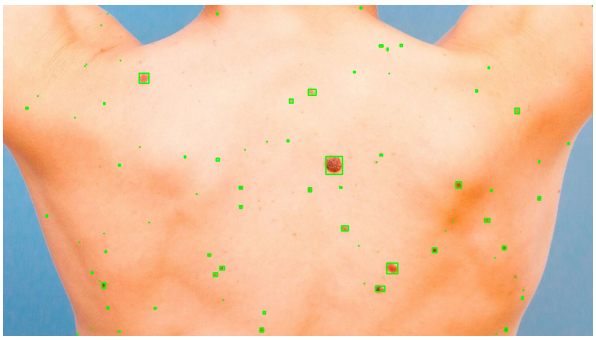
1. Image classification algorithms analyse images and the objects present inside them and then divide them into different classes:
    a. A binary classifier, as the name suggests, categorises the objects in this image into two classes, such as vehicle or not, and so on.

b. A multiclass classifier, on the other hand, categorises objects into multiple classes and tells whether the object present is a car, a bus, a bike, etc.

2. The output in the case of object detection algorithm will include the above-mentioned classes along with the locations of the car, the bus and the bike, as shown in the image below.

Some of the common applications of object detection algorithms are mentioned below:

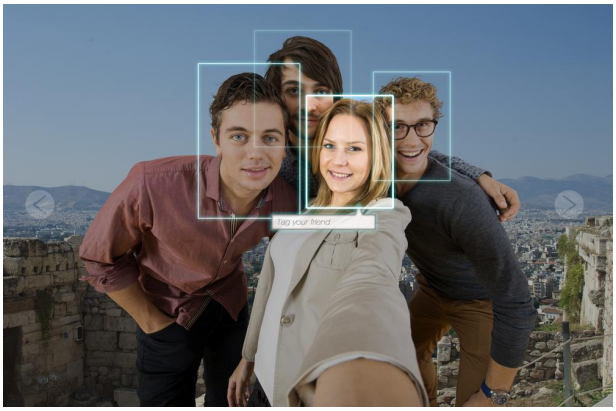
1. **Medical imaging:** Object detection is a commonly used technique that detects the presence of unwanted and harmful cells by analysing images of the human brain, lungs, etc. It can also be used to detect the presence of moles as shown in the image below.



2. **Remote sensing:** The object detection algorithm can be used to detect objects in the images captured by satellites. The image given below shows one such example, where an oil tanker is being detected from a satellite image.

3. **Self-driving cars:** One of the actively discussed and newest innovations in technology, self-driving cars, use object detection algorithms to detect obstacles on roads. The image given below shows the dashboard of a self-driving car, which uses this technology.

4. **Face recognition:** Another common application of object detection is detecting human faces. The image given below depicts this application.



5. **Traffic monitoring:** Traffic monitoring is one of the widely used applications of object detection algorithms. You already learnt, at the beginning of this segment, how object detection algorithms detect the presence of different vehicles, along with their classes.

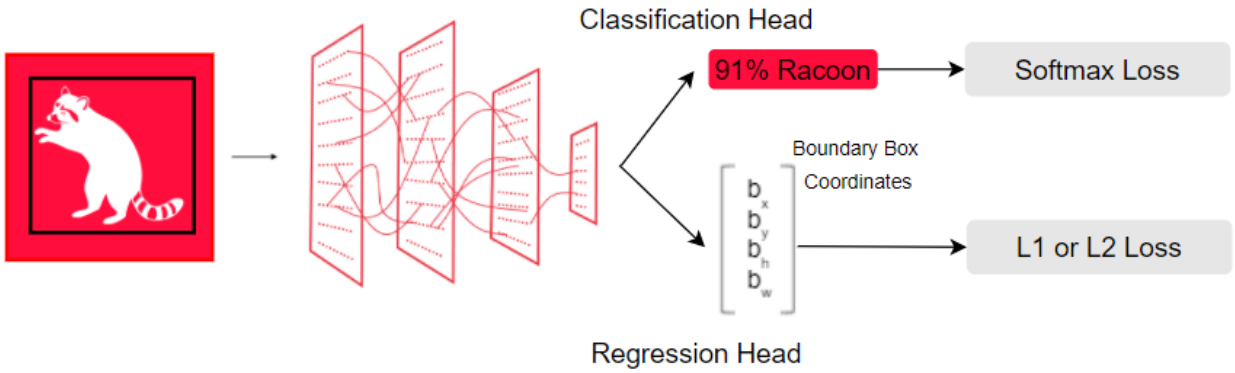
The coloured boxes that you see around the objects in this image are called **bounding boxes**. These boxes provide the coordinates of the objects detected inside an image.

Object Localisation

An object detection algorithm performs two tasks, which are:

a. Detecting an object's class and
b. Identifying the object's location through 'bounding boxes'.

The image given below provides a summary of the working of object detection algorithms.



Let's now learn each of these elements that we have discussed so far. **Object localisation** refers to determining the locations of the objects present in an image. The location of the objects is given by the coordinates of the **bounding boxes**. A **bounding box vector** consists of four elements, which are:

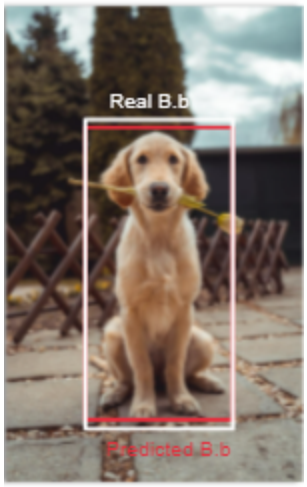
- The x coordinate of the centroid,
- The y coordinate of the centroid,
- The height of the bounding box and
- The width of the bounding box.

In vector form, a bounding box is represented as shown below..

$$\begin{bmatrix} b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

Now, since the output of an object detection model is primarily a bounding box, you need a method to be able to evaluate the efficiency of a model to localise an object and predict its bounding box accurately. The standard industry metric that you need to consider for this purpose is **intersection over union** (IoU)**.**

This metric evaluates the accuracy of your model's prediction of the bounding box by comparing it with the '**ground truth**' bounding box present in the training set. Take a look at the following image, which shows the predicted and ground truth bounding boxes.

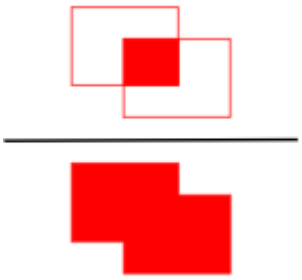


Recall that this is similar to the actual and predicted outputs, which you have been comparing in the case of ML and deep learning (DL) algorithms so far. However, instead of having a single value of the outputs, in the case of object detection algorithms, you will have a vector.

The IoU is measured by taking the intersection of the two boxes and dividing it by their union. IoU can be expressed by the following formula:

$$\text{IoU} = \text{Area of intersection} / \text{Area of union}$$

The image given below depicts the numerator and the denominator of this formula.



Note that IoU lies between 0 and 1, since it is a fraction.

Next, you understood how to compute loss. Object detection algorithms have two losses, which are:

1. Classification or softmax loss, and
2. Regression losses – L1 and L2 losses.

You are already familiar with these losses, but let's revisit the L1 and L2 losses for object detection algorithms. **L1 loss** is employed to minimise the error that is defined as the sum of all the absolute deviations between the true and the projected values. The formula for L1 loss is as follows:

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

L2 loss is used to reduce the error that is defined as the total of all the squared discrepancies between the true and the predicted values. The formula for L2 loss is as follows:

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

As you already know, L2 loss is the more preferred option in the case of outliers in data.

SUMMARY

Region-Based Detectors

In this session, you were introduced to the first set of object detectors, that is, region-based detectors.

FCNs for Object Detection

You learnt about fully connected convolutional neural networks (FCNs) in one of the previous modules, wherein you used them to classify images. However, can you perform the task of detecting objects using FCNs? Well, no. You cannot.

FCNs cannot be used as object detection algorithms because some of their properties act as limitations in the case of object detection tasks. These limitations are as follows:

1. The length of the output layer in object detection tasks is variable, as the number of objects/occurrences is unknown and varies. This unknown length of the output layer cannot be handled using FCNs.

2. FCNs use sliding windows, with a stride of one, which will end up slowing down the operation.

   Sliding windows slide through the image in rectangular steps of equal sizes to extract different areas from the entire image. This windowed image is then fed to feature extractors and classifiers in the case of FCNs. Using sliding windows to extract a subset of images that have to be fed to the feature extractor will give you thousands of such images in that subset.

3. Feeding thousands of these images to the classifier will increase the time and memory requirement of the model, which is counterproductive.

Therefore, using FCNs is not the most effective way to deal with object detection problems.

One alternative to deal with the limitations of FCNs is to divide the input image into smaller images and then feed them to the CNN, i.e., the classifier. However, the technique of slicing the input image into smaller images has its own disadvantages, which are as follows:

1. Important objects may have different spatial locations within the image.

2. The objects of interest will have different aspect ratios in the smaller images.

3. This technique requires you to feed a very large number of smaller images with overlapping regions, which, even though reduces the time required for computation as compared to FCNs, still requires large computational resources.

Therefore, either using FCNs or feeding multiple sub-images of the input image to the CNN is not a suitable approach for real-time object detection problems.

RCNN

FCNs cannot be used as object detectors because they require you to divide the input image into a subset of multiple smaller images, which, when fed to the classifier, require a lot of processing time. One way to solve this problem would be to partition the input image into **several regions** and to use a CNN to classify the object within each region. This approach is, essentially, the principle behind region-based methods, which will be covered in this session.

As the name suggests, a **region-based approach** works with a **region proposal** that extracts the important areas/regions from the image and performs the tasks of **feature extraction** and **classification** on these regions. There are three types of region-based methods, which are as follows:

1. RCNN
2. Fast RCNN
3. Faster RCNN

As you learnt earlier, using sliding windows to extract a subset of images that will be fed to the classifier will result in generation of thousands of images. Do you require all of these images?

Instead of feeding thousands of input images into the feature extractor, RCNNs use **region proposals.** They propose certain regions using a selective search technique to extract 2,000 different regions from the given image.

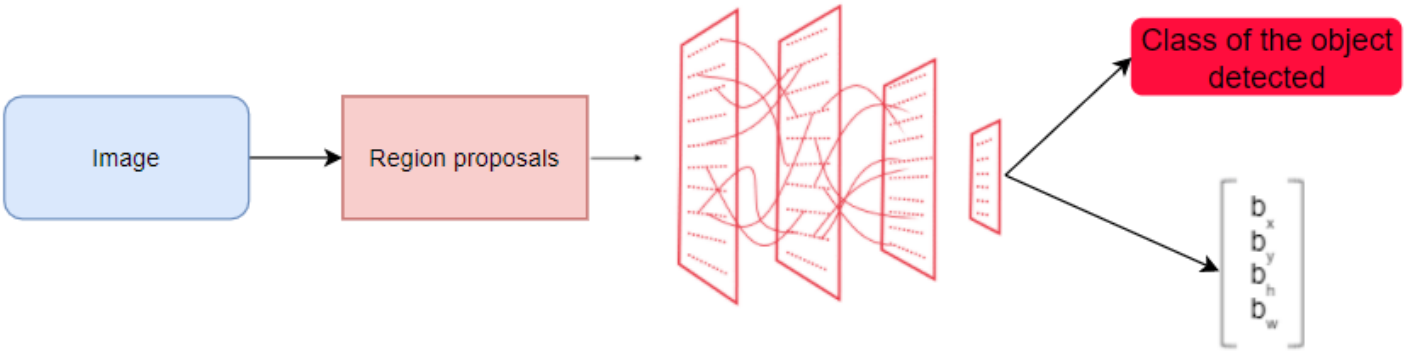
The process of extracting region proposals occurs in three steps, which are as follows:

1. The algorithm calculates an initial picture sub-segmentation. This process generates a large number of possible regions.

2. The algorithm then recursively combines comparable sections into larger ones using the **greedy approach**.

3. Then, the algorithm proposes the final potential regions from these larger regions.

These regions are then warped and squared and finally fed into a CNN, which functions as a feature extractor. Once the features are extracted, they are fed into an SVM to verify the presence of objects in them. Additionally, the values of the bounding box vector are given as an output by this algorithm, as shown in the image given below.



However, this method has certain limitations due to which it is not actively used in the industry today. Some of these limitations are as follows:

1. Feeding 2,000 region proposals per image to the feature extractor requires a lot of time and resources, thereby making the process extremely slow.

2. Even deployment is quite slow, which makes it difficult to use this algorithm for real-world applications.

3. The selective search algorithm does not learn, which can lead to the generation of bad proposals.

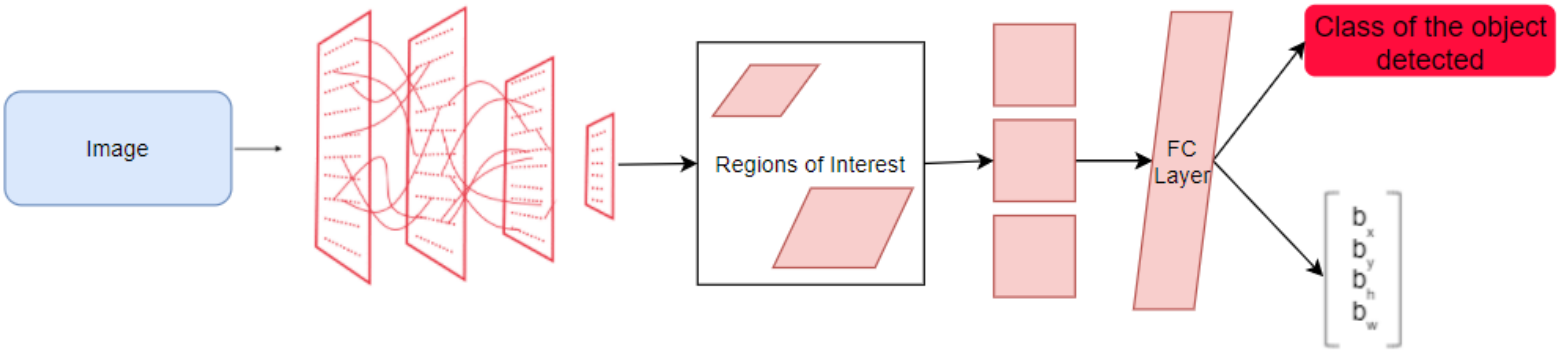
4. The results produced using support vector machines (SVM) as classifiers have low accuracy.

Fast RCNN and Faster RCNN

In the previous segment, you learnt about RCNN and how the 2,000 region proposals makes the algorithm slow. This problem can be solved by using fast RCNN, which is a modified version of the RCNN algorithm.

As you learnt in this video, the fast RCNN algorithm feeds the entire image as the input to the feature extractor, instead of feeding 2,000 region proposals. The feature extractor, CNN, then produces a feature map for the image that is used by the algorithm to identify the region proposals. Again, the region proposals are identified using a selective search method, similar to the one employed in RCNN.
Each region proposed for a single image is of different shape and size. Therefore, they need to be converted to the same shape. Each of these region proposals is then wrapped into a square and transformed into the same shape using a **region of interest (RoI) pooling layer.**

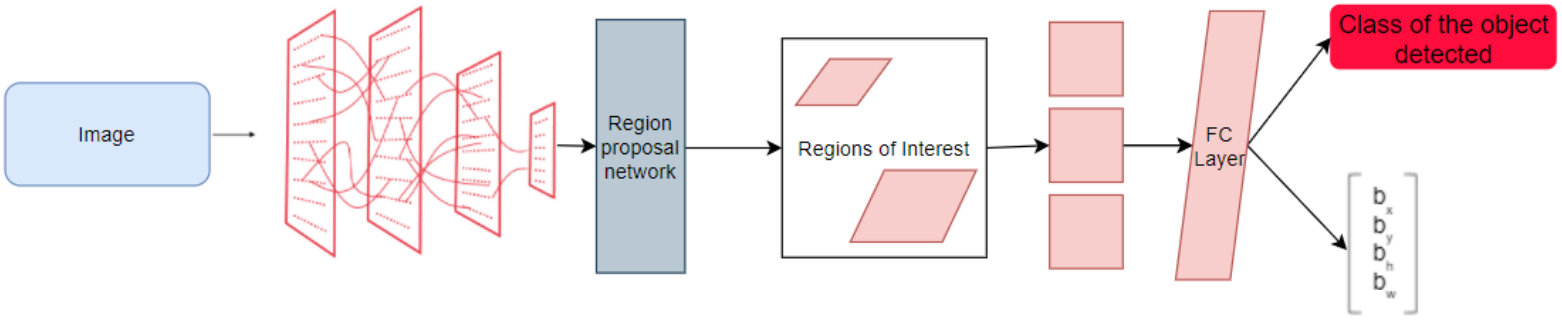
Finally, the output of the pooling layer is fed to the **fully connected layer**, which predicts the classes using the **softmax layer** and provides the bounding box coordinates for the detected object, as illustrated below.



This solves the problem of having a large number of region proposals, which, in turn, saves a lot of time. However, in fast RCNN, you are still using the selective search method, which does not learn and might generate incorrect region proposals, as in the case of RNN. The selective search method consumes a lot of time and slows down the Fast-RCNN detector. This is one of the main limitations of the fast RCNN technique.

This limitation can be overcome by using **faster RCNN.** Instead of using a selective search method to propose regions, faster RCNN uses a **region proposal network (RPN)** to select regions from the given image.

The working of this algorithm is similar to that of the fast RCNN algorithm. However, fast RCNN has one additional layer, which is the RPN. The feature map produced by the CNN is fed to the RPN. The RPN provides region proposals that are re-shaped and fed into the RoI pooling layer. The final step of the process is the same as it was in the case of fast RCNN, wherein the output of the pooling layer is classified along with the bounding box vector. Thus, this algorithm helps overcome the limitations posed by the selective search techniques. The architecture of faster RCNN is illustrated below.



Even though the faster RCNN algorithm helps counter the shortcomings of the selective search techniques, it still cannot be used for some real-time applications due to certain limitations of its own. These limitations are attributed to the following properties of the region-based techniques:

1. These algorithms work on a two-stage approach:
    a. Proposing different regions
    b. Processing these regions

2. The proposed regions, which have a high probability of the presence of the object, are processed. The object is then localised inside the image.

Even after various advancements from RCNN to faster RCNN, these methods are still not fast enough to deal with real-time problems. Hence, they are not used actively in the industry today.

SUMMARY

One-Shot Detectors

In this session, you learnt the second set of detectors called the one-shot detectors. You also learnt how to implement the real-time tasks using pre-trained models of these detectors.

YOLO - Introduction

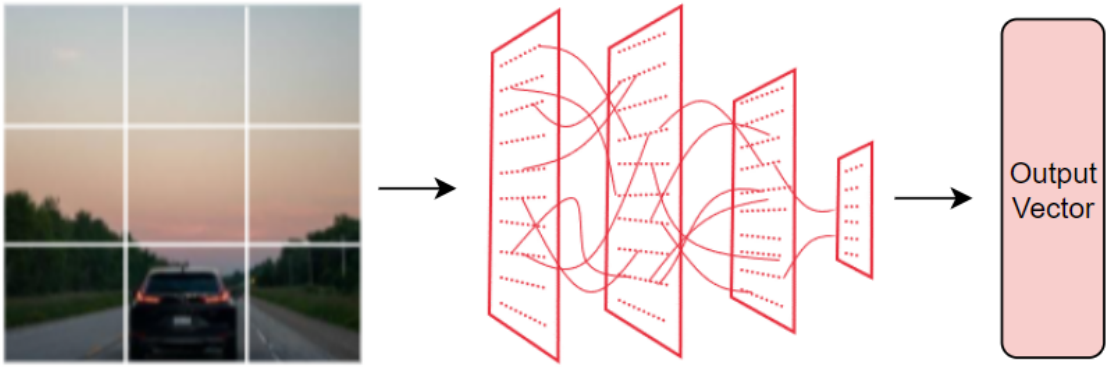
The first one-shot object detector that you learnt is **YOLO - You only look once**. You also understood the YOLO algorithm through an object detection example. The input image for this demonstration is given below.

Problem Statement: Detecting the presence of humans, cars and bikes in the image provided

Now, using this input image, you understood the working of the YOLO model step by step. The steps in this algorithm are discussed below:

1. **Preprocessing:** The YOLO model divides the image into a 3 × 3 grid as shown below.

2. **Train the network:** The resultant image is then passed through a single convolutional neural network (CNN) as shown in the following image.



3. **Analysing the output:** The model predicts the output of each grid cell in a single forward pass (feedforward propagation) through CNN. If the centroid of an object is present in a certain grid cell, the cell containing the object is labelled such that the same object is not detected again.

   The output of the algorithm can be divided into two parts as shown below:

   a. The output of each grid cell: The output of each grid cell contains c + 5 elements, where c = 3. On substituting the value of c, you will get c + 5 = 3 + 5 = 8 elements in the given example. Take a look at the following output vector.

$$Y=\begin{bmatrix} p_c \\ c_1 \\ c_2 \\ c_3 \\ b_{cx} \\ b_{cy} \\ b_w \\ b_h \end{bmatrix}$$

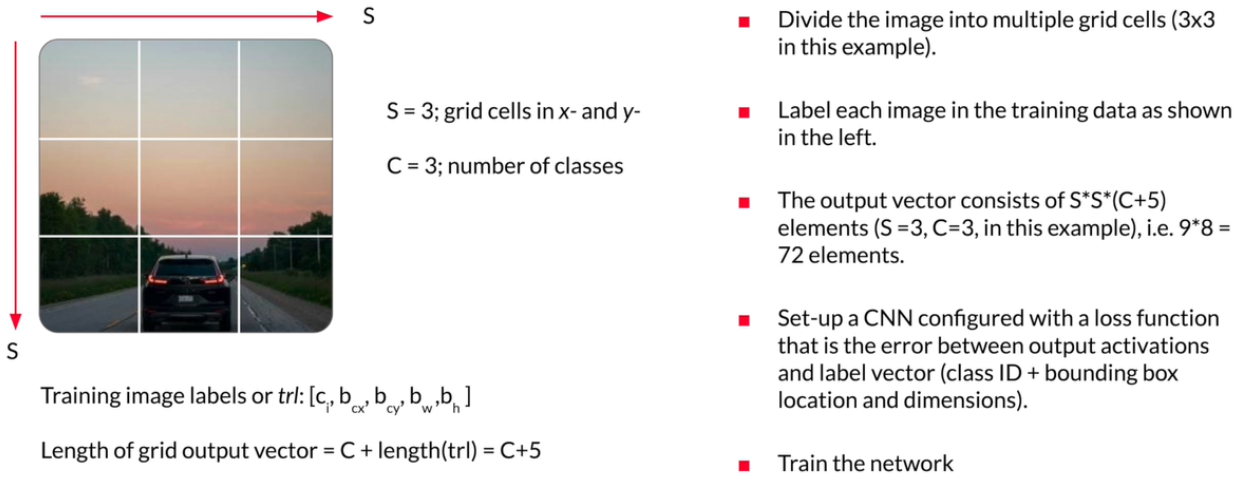
   Here, p_c is the softmax probability of the object present in the image.
   c_1, c_2 and c_3 are, respectively, the different classes, that is, humans, cars and bikes.
   b_{cx}, b_{cy}, b_w and b_h represent, respectively, the bounding box's centroid, that is, x; centroid, that is, y; width and height.

   NOTE: In this example, there is only one object in a grid cell. Therefore, only one probability value, that is, p_c, will be present. In the upcoming segment, you will also come across examples with more than one object in a grid cell.

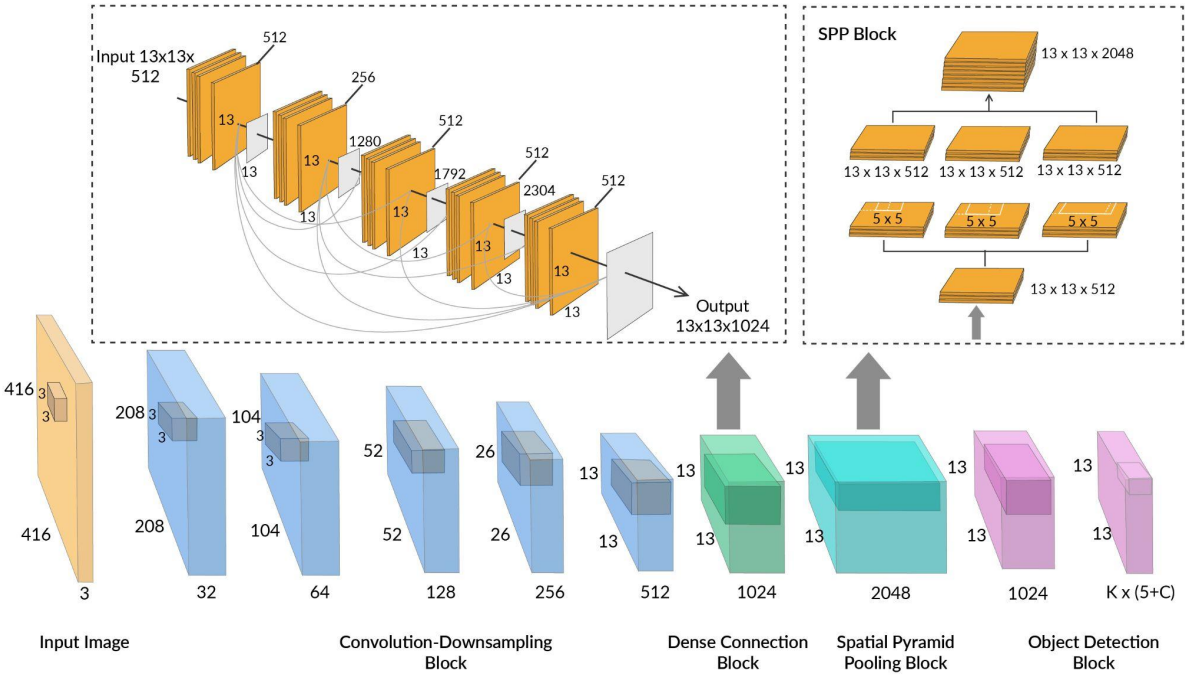
   b. **Overall output:** The overall output of the model is the resultant output of all the grid cells. The output vector contains s * s * (c + 5) elements, where s * s = 3 * 3

= 9 (number of grid cells) and c = 3 (number of classes). Hence, the output vector in the given example will have 72 elements.

All the above-mentioned steps are summarised in the image given below:



The overall architecture of the YOLO model is depicted in the following image.



The YOLO model that is currently being used is YOLOv3. This third iteration of the YOLO model is a variant of the DarkNet project, which you will use to train your YOLO model in the upcoming segments. This version of the YOLO model offers greater accuracy than the previous versions, although it takes a longer time for computation due to a higher number of convolutional layers.

As you saw in this image, the input image with a 3 × 3 grid is passed through a series of convolutional downsampling networks such that the 416 × 416 image is downsampled to a vector of 13 × 13 × 512 dimensions.

This vector is then passed through a dense network, followed by a pooling layer. The final step of the process comprises detection of the objects. The output of the bounding box and class is given as the output of the network.

YOLO - Working

In this segment, you learnt the process that an image goes through during the training step in the YOLOv3 model. The input image, when divided into a 3 × 3 grid, goes through the network and each block that has no centroid is skipped from the object search process, as depicted in the following image.

For each of these grid cells, the probability of an object belonging to any class is 0, i.e., $p_c = 0$. Therefore, the coordinates and class of the bounding box do not matter and, hence, they will not be computed. Next, we focussed on the grid cell that contains the centroid of the object, as shown in the following image.

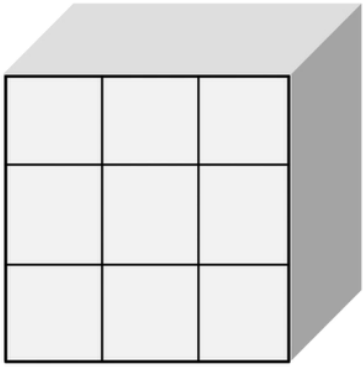
The probability of an object being detected in this grid cell is 1. Further, $c_1 = 0$, $c_2 = 1$ and $c_3 = 0$, where c_1, c_2 and c_3 are the classes for humans, cars and bikes, respectively. The green box around the object is the bounding box of the detected object.

In this example, you learnt that the bounding box coordinates obtained from the YOLO model are $b_{cx} = 0.5$, $b_{cy} = 0.6$, $b_w = 0.3$ and $b_h = 0.3$.

Considering this information, the output of the grid well is depicted in the image given below.

$$Y = \begin{bmatrix} p_c \\ c_1 \\ c_2 \\ c_3 \\ b_{cx} \\ b_{cy} \\ b_w \\ b_h \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0.5 \\ 0.6 \\ 0.3 \\ 0.3 \end{bmatrix}$$

The overall output of the model, i.e., the outputs of all the grid cells will consist of 72 elements as shown in the image given below.

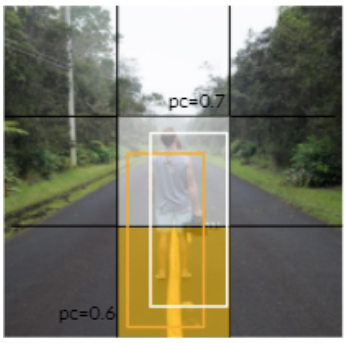


YOLO - Analysing The Output

If you recall, in ML models, you computed accuracy, the F1 score, precision and recall, etc. in order to determine the quality of the prediction. In the case of object detectors, you calculate intersection over union (IoU), which serves as an evaluation metric for model prediction, i.e., the bounding box predicted by the model.

While working with complex images, you can get multiple bounding boxes for the same object; however, you do not require all of them. So, how will you select the most accurate bounding box?

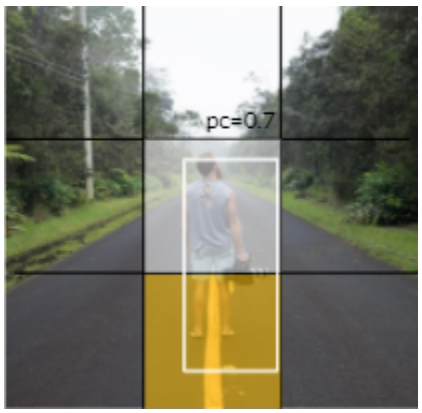
Take a look at the following image and try to predict which of the two bounding boxes is more accurate for the given object.



Each bounding box in this image is accompanied by the probability of the object belonging to a particular class. In such cases, where you have multiple bounding boxes, you can check the IoU values. If the IoU value is less than the threshold value, then the corresponding bounding box can be rejected.

However, if the IoU values of more than one bounding box are greater than the threshold value, then you can use the technique of NMS. This method selects the bounding box with the higher probability (p_c) and rejects the others. So, for the image given above, the white bounding box provides better prediction as p_c= 0.7 is greater than 0.6.

The output image after performing NMS will look as shown below.



YOLO - Anchor Boxes

So far, you were dealing with images with only one object in any grid cell. You also tackled the challenge of multiple bounding boxes covering a single object. However, there is another problem: What if there are more than one objects in the grid cell or overlapping objects in the image? How will the YOLO model detect different objects in such a situation?

It might be possible that the same grid cell contains overlapping objects. In such cases, the concept of **anchor boxes** comes into play. In order to understand this concept, let's take a look at the example shown in the following image.

So, as you observed in this image, the marked grid cell contains two objects: the person and the bike. In this case, the centroids of both these objects lie in this bounding box. Through the techniques that we have discussed so far, the YOLO model would register the object once, which can either be the bike or the person, and then, this cell will be blocked for further inspection. This means that the second object will not be detected properly.

Anchor boxes would allow you to register both the objects, even though they are in the same grid cell. The bounding boxes of the two objects would look as shown in the following image.

Next, you understood the working of the bounding boxes step by step, as discussed below:

1. Anchor boxes are a set of bounding boxes for each object in an image. These boxes have a predefined width and height.

2. The anchor boxes reflect the following factors for the objects being detected:
    a. Aspect ratio
    b. Scale

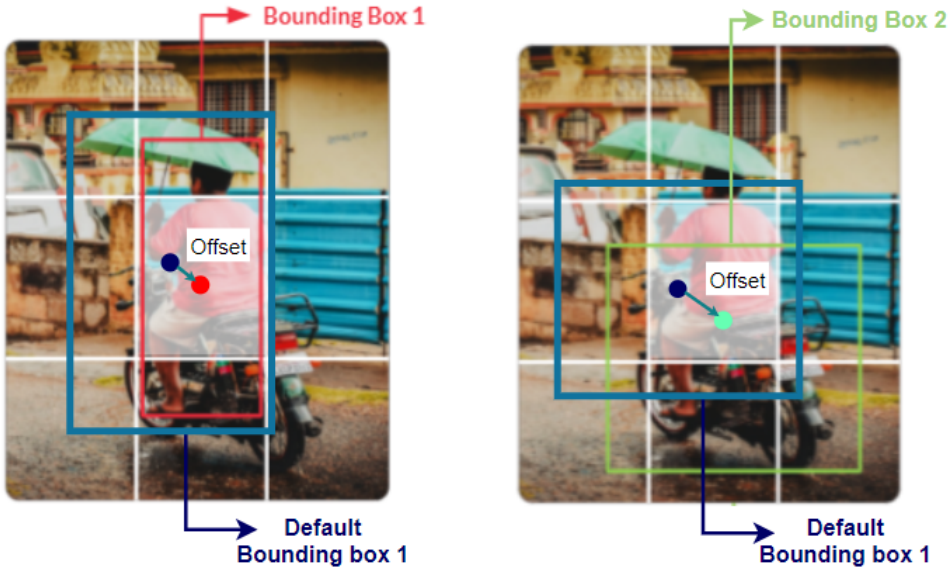
3. The boxes are selected on the basis of the size of the training data by clustering the dimensions of the ground truth boxes provided with the data.

4. The number of anchor boxes depends upon the number of objects being detected.

5. The final boxes are refined and corrected based on the following parameters:
    a. Background
    b. IoU
    c. Offsets for each anchor box

6. YOLO predicts the probabilities and changes them in correspondence to each anchor box. Then, it uses them to refine the anchor box predictions.

Next, you visualised the default bounding boxes next to the final bounding boxes and observed the offsets shown in the images below.



Considering the sliding window technique, which processes each part of an image separately for making predictions, and then comparing it to the anchor box method, which can simultaneously predict all the objects present in an image, you were able to understand how this technique requires much less time and effort.

Through techniques such as anchor boxes, YOLO is able to process an image and make predictions in one go. Hence, it is known as a one-shot detector.

Finally, you also implemented a traffic surveillance task using a YOLO pre-trained model.

SSD - Introduction

As the name suggests, similar to the YOLO detector, SSD uses a single pass to detect the objects in the input image. The SSD algorithm comprises two main parts, which are:

1. A neural network at the base and
2. Convolutional layers.

The neural network works as a **feature extractor**. The SSD uses a VGG-16 network, which is a CNN of superior quality. However, it does not use the output layers of this CNN, which are used for the image classification process.

The **convolutional layers** work as object detectors. They locate the bounding boxes and classify the objects present in an image. These layers reduce the size of images as they pass through these layers, thus allowing the network to detect objects at different scales.

Next, you understood the working of this algorithm step by step, as discussed below:

1. First, the feature extractor, VGG-16, provides the feature maps of the input image.

2. Next, the convolutional layers detect objects at different scales and provide **four predictions per cell**.

3. Each prediction comprises:
    a. The boundary box coordinates,

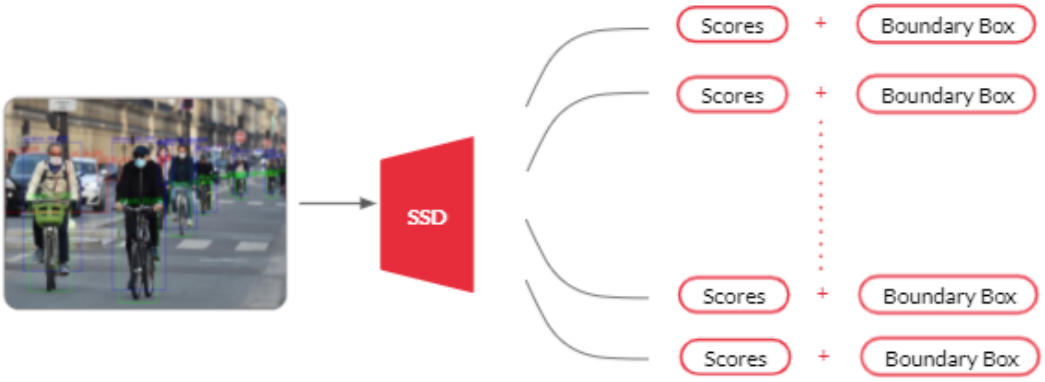
    b. 21 scores for each class and

    c. An extra class for no object being detected.

4. The class with the highest score is considered to be the class of the object being bound.

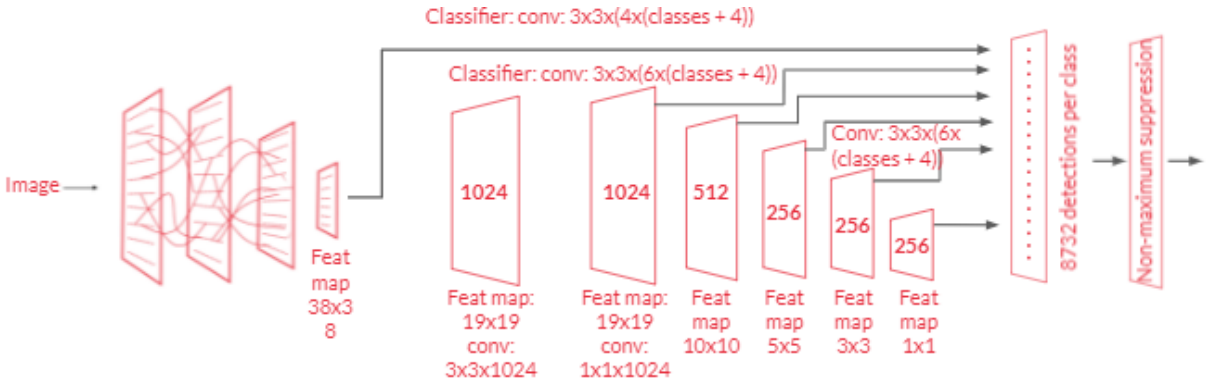
5. This technique of making multiple predictions for each object is called **multibox**.

The image given below shows the working of multibox techniques.



SSD - Working of The Algorithm

In this segment, you learnt, in detail, about this algorithm. The architecture of the SSD model is depicted in the image given below.



So, as you saw in this image, the SSD comprises a total of six convolution layers of different sizes and provide different scales of the same image. This allows the model to detect the objects at multiple scales.

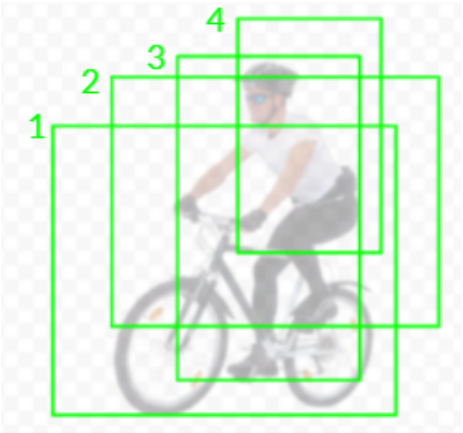
The initial layers have a small receptive field, i.e., they cover a low spatial region. These layers detect the smaller objects in an image. However, the latter layers have a large receptive field and detect the bigger objects.

In the YOLO model, you learnt about anchor boxes, which help detect multiple overlapping objects. The SSD uses **default bounding boxes**, which serve the same purpose. There are 4 or 6 default bounding boxes, which are defined with one prediction per box.

This serves as a starting point for the algorithm by giving the model an idea about how the objects look and where they might be located. The SSD then uses feature maps to detect objects at different scales. Now, you might wonder how the SSD selects the final bounding box out of the 4 or 6 default boxes.

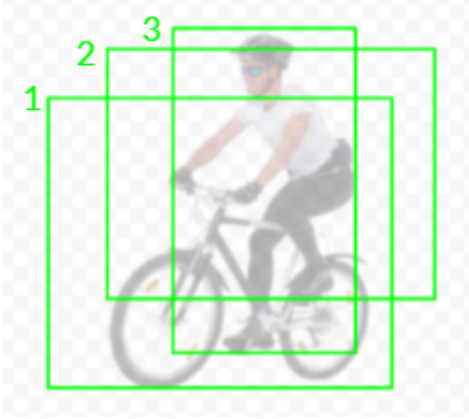
The technique of selecting the final bounding box is referred to as the **matching strategy** in the SSD. Take a look at the image given below:

Let's draw four default bounding boxes for this image. The objects in the image include the human and the bike. The default bounding boxes for this image are shown below.

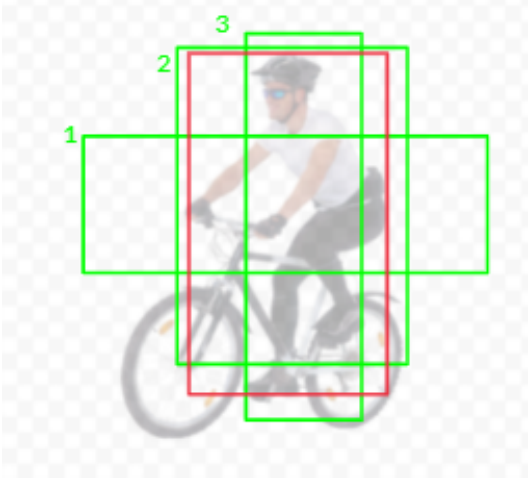


The conv4_3 layer divides the image into a 38 × 38 grid and predicts four boxes per cell. Therefore, the overall predictions made by the conv4_3 layer become 38 × 38 × 4. As you already know, each prediction has the bounding box coordinates and 21 scores, and the highest score denotes the class of the detected object.

Of the four boxes, box 4 contains only one object, the human, whereas the others have both the human and the bike. Therefore, boxes 1, 2 and 3 can be taken into consideration to understand this concept. The resultant image is shown below.



The ground truth box is shown in red in the following image.



Now, these three predictions (excluding the ground truth) are segregated into positive or negative based on whether their IoU is lower or higher than the ground truth, i.e., 50%. If the value of IoU is less than 50%, then it is said to be a **negative match** and is rejected. If the value is more than 50%, then it is said to be a **positive match** and is accepted.

In this example, the IoU of box 1 is less than 50%, which implies that it is a negative match. It will, thus, be rejected. On the other hand, the IoU of boxes 2 and 3 are greater than 50%, i.e., a positive match.

This prediction comprises the following two types of losses:

1. **Confidence loss:** It refers to the loss incurred during the prediction of the class, which is calculated using the categorical cross-entropy loss, which you learnt in the previous modules:
    a. In the case of a positive match, the loss is penalised according to the confidence score of the corresponding class.

    b. In the case of a negative match, the loss is penalised according to the confidence score of the class '0'.

2. **Location loss:** It refers to the loss incurred while calculating the bounding box coordinates. It is calculated using the L1 or L2 loss formula, which you learnt about and implemented previously.

The total loss can be calculated using the following formula:

$$\text{Multibox or SSD loss} = \text{Confidence loss} + (\alpha \times \text{Location loss})$$

Here, α is used to balance the overall loss.

Finally, you detected the faces in images using the SSD pretrained model.