CHAPTER 5

Object Detection Using Deep Learning

Just because something doesn't do what you planned it to do doesn't mean it's useless.

—Thomas Edison

To solve a particular problem, we try multiple solutions, and many times after a few iterations, we find the best solution. Machine learning and Deep Learning are no different. During the discovery phase, improvements and constant modifications are done to improve the performance of the previous version of the algorithms. The weakness observed in the last phase, the slowness in the computation, the incorrect classifications made – all pave the way for a better solution.

In the last two chapters, we understood and created solutions to classify images into binary or multiple classes. But most of the images had only one object in them. And we did not identify the location of an object in the image. We simply said whether or not the object is present in that image.

In this chapter, we will be identifying an object in any image. At the same time, its position will also be determined by creating a bounding box around it. It is a step ahead from the image classification solutions we have developed so far.

V. Verdhan, *Computer Vision Using Deep Learning*,
https://doi.org/10.1007/978-1-4842-6616-8_5

There are quite a few architectures for Object Detection like R-CNN, Fast R-CNN, Faster R-CNN, SSD (Single Shot MultiBox Detector), and YOLO (You Only Look Once).

We are going to study these network architectures in this chapter and create Python solutions for the same.

We will cover the following topics in this chapter:

1. Object Detection and the use cases
2. R-CNN, Fast R-CNN, and Faster R-CNN architectures
3. Single Shot MultiBox Detector
4. You Only Look Once (YOLO)
5. Python implementation of the algorithms

Welcome to the fifth chapter and all the very best!

5.1 Technical requirements

The code and datasets for the chapter are uploaded at the GitHub link `https://github.com/Apress/computer-vision-using-deep-learning/tree/main/Chapter5` for this book. As always, we will be using the Python Jupyter Notebook for this chapter. For this chapter, a GPU might be required to execute the code, and you can use Google Colaboratory. The instructions to set Google Colab. Same as last chapter of the book.

Let's proceed with the Deep Learning architectures in the next section.

5.2 Object Detection

Object Detection is one of the most cited and acknowledged solutions in the area of Machine Learning and Deep Learning. It is quite a novel solution and very interesting to solve. The use cases for object detection are quite a few. And hence, organizations and researchers are spending huge time and resources to uncover this capability. As the name suggests, Object Detection is a computer vision technique to locate objects in an image or in a video. The detection can be intended to be done in a live stream video too. When we humans look at a picture, we can quickly identify the objects and their respective position in an image. We can quickly categorize if it is an apple or a car or a human being. We can also determine from any angle. The reason is that our minds have been trained in such a way that it can identify various objects. Even if the size of an object gets smaller or bigger, we are able to locate them and detect them.

The goal is to replicate this decision-making intelligence using Machine Learning and Deep Learning. We will be examining the concepts of object detection, localization, and classification and developing Python codes.

But before examining the basics of object detection, we should first study the difference between object classification, object localization, and object detection. They are the building concepts for object detection. We are studying the difference in the three components in the next section now.

5.2.1 Object classification vs. object localization vs. object detection

Look at the images in Figure 5-1 of a vacuum cleaner. In the previous chapters, we have developed the image classification solutions to classify such images into “a Vacuum Cleaner” or “not.” So we could have easily labeled the first image as a vacuum cleaner.

On the other hand, localization refers to finding the position of the object in an image. So when we do Image Localization, it means that the algorithm is having a dual responsibility of classifying an image as well as drawing a bounding box around it, which is depicted in the second image. In the first image of Figure 5-1, we have a vacuum cleaner, and in the second image, we have localized it.

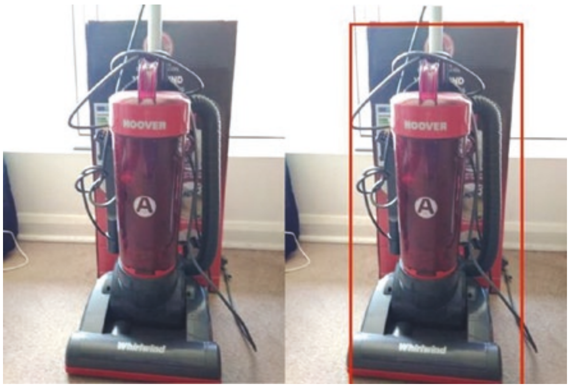

Figure 5-1. *Object detection means identifying and localization of the object. In the first image, we can classify if it is a vacuum cleaner, while in the second image, we are drawing a box around it, which is the localization of the image*

To scale the solution, we can have multiple objects in the same image and even multiple objects of different categories in the same image, and we have to identify all of them. And draw the bounding boxes around them. An example can be of a solution trained to detect cars. On a busy road, there will be many cars, and hence the solution should be able to detect each of them and draw bounding boxes around them.

Object detection is surely a fantastic solution. We will now discuss the major object detection use cases in the next section.

5.2.2 Use cases of Object Detection

Deep Learning has expanded many capabilities across domains and organizations. Object detection is a key one and is a very powerful solution which is making huge ripples in our business and personal world. The major use cases of object detection are

1. Object Detection is the key intelligence behind autonomous driving technology. It allows the users to detect the cars, pedestrians, the background, motorbikes, and so on to improve road safety.

2. We can detect objects in the hands of people, and the solution can be used for security and monitoring purposes. Surveillance systems can be made much more intelligent and accurate. Crowd control systems can be made more sophisticated, and the reaction time will be reduced.

3. A solution might be used for detecting objects in a shopping basket, and it can be used by the retailers for the automated transactions. This will speed up the overall process with less manual intervention.

4. Object Detection is also used in testing of mechanical systems and on manufacturing lines. We can detect objects present on the products which might be contaminating the product quality.

5. In the medical world, the identification of diseases by analyzing the images of a body part will help in faster treatment of the diseases.

There are very less areas where the usage is not envisioned. It is one of the areas which are highly researched, and every day new progress is being made in this domain. Organizations and researchers across the globe are making huge ripples in this area and creating path-breaking solutions.

We have examined the major use cases of object detection. Now let's look at some methods for object detection.

5.3 Object Detection methods

We can perform object detection using both Machine Learning and Deep Learning. We will be discussing the Deep Learning methods in this book, but for curious readers, here are a few solutions:

1. Image segmentation using simple attributes like shape, size, and color of an object.
2. We can use an aggregated channel feature (ACF), which is a variation of channel features. ACF does not calculate the rectangular sums at various locations or scales. Instead, it extracts features directly as pixel values.
3. Viola-Jones algorithm can be used for face detection. The suggested papers are at the end of the chapter.

There are other solutions like RANSAC (random sample consensus), Haar feature–based cascade classifier, SVM classification using HOG features, and so on which can be used for object detection. In this book, we are focusing on Deep Learning methods.

The following Deep Learning architectures are commonly being used for Object Detection:

1. R-CNN: Regions with CNN features. It combines Regional Proposals with CNN.
2. Fast R-CNN: A Fast Region–based Convolutional Neural Network.

3. Faster R-CNN: Object detection networks on Region Proposal algorithms to hypothesize object locations.
4. Mask R-CNN: This network extends Faster R-CNN by adding the prediction of segmentation masks on each region of interest.
5. YOLO: You Only Look Once architecture. It proposes a single Neural Network to predict bounding boxes and class probabilities from an image in a single evaluation.
6. SSD: Single Shot MultiBox Detector. It presents a model to predict objects in images using a single deep Neural Network.

Now, we will examine the Deep Learning frameworks in the next section. There are some base concepts to be studied which form the base of object detection techniques. We will study them too. Post the discussion of Deep Learning frameworks, we will create Python solutions.

5.4 Deep Learning frameworks for Object Detection

We will now start with Deep Learning–based object detection algorithms and architectures. They are made of a few components and concepts. Before diving deep into the architectures, let us first recognize a few of the important components of Object Detection. The key ones are

- Sliding window approach for Object Detection
- Bounding box approach

- Intersection over Union (IoU)
- Non-max suppression
- Anchor boxes concept

We will start with the sliding window approach in the next section.

5.4.1 Sliding window approach for Object Detection

When we want to detect objects, a very simple approach can be: why not divide the image into regions or specific areas and then classify each one of them. This approach for object detection is *sliding window*. As the name suggests, it is a rectangular box which slides through the entire image. The box is of fixed length and width with a stride to move over the entire image.

Look at the image of the vacuum cleaner in Figure 5-2. We are using a sliding window at each part of the image. The red box is sliding over the entire image of the vacuum cleaner. From left to right and then vertically, we can observe that different parts of the image are becoming the point of observation. Since the window is sliding, it is referred to as the sliding window approach.

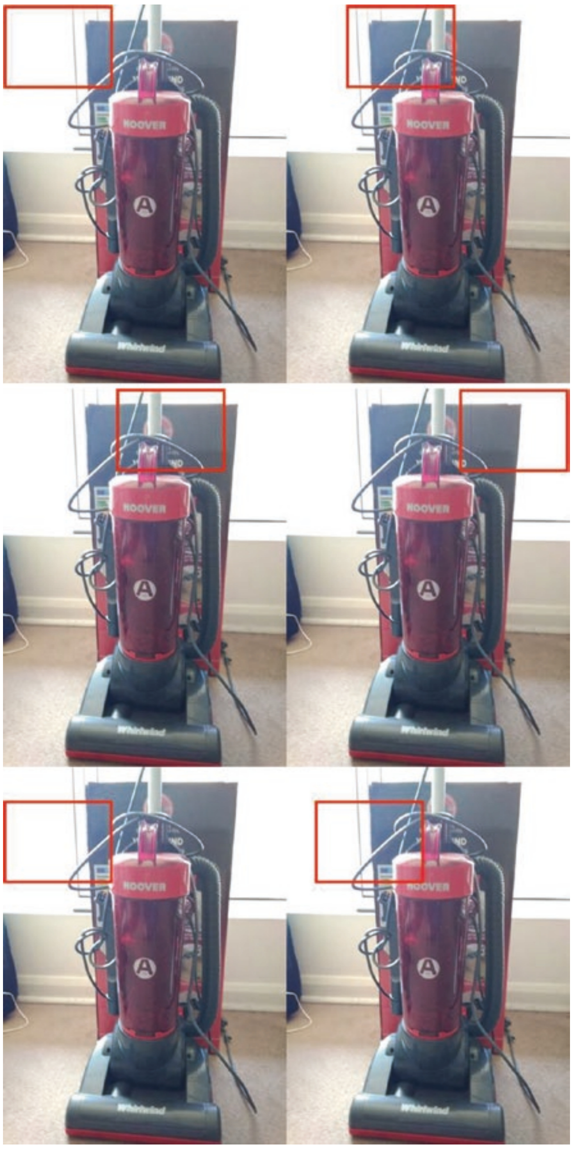

Figure 5-2. *The sliding window approach to detect an object and identify it. Notice how the sliding box is moving across the entire image; the process is able to detect but is really a time-consuming process and computationally expensive too*

Then for each of these regions cropped, we can classify whether this region contains an object that interests us or not. And then we increase the size of the sliding window and continue the process.

Sliding window has proven to work, but it is a computationally very expensive technique and will be slow to implement as we are classifying all the regions in an image. Also, to localize the objects, we need a small window size and small stride. But still it is a simple approach to understand.

The next approach is the bounding box approach.

5.5 Bounding box approach

We discussed the sliding window approach. It outputs less accurate bounding boxes as it is dependent on the size of the window. And hence we have another approach wherein we divide the entire image into grids (x by x), and then for each grid, we define our target label. We can show a bounding box in Figure 5-3.

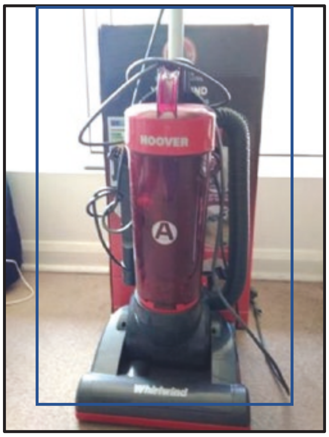

Figure 5-3. *Bounding box can generate the x coordinate, y coordinate, height, and width of the bounding box and the class probability score*

A bounding box can give us the following details:

> Pc: Probability of having an object in the grid cell (0: no object, 1: an object).
>
> Bx: If Pc is 1, it is the x coordinate of the bounding box.
>
> By: If Pc is 1, it is the y coordinate of the bounding box.
>
> Bh: If Pc is 1, it is the height of the bounding box.
>
> Bw: If Pc is 1, it is the width of the bounding box.
>
> C1: It is the class probability that the object belongs to Class 1.
>
> C2: It is the class probability that the object belongs to Class 2.

Note The number of classes depends on whether the problem at hand is a binary classification or a multilabel classification.

If an object lies over multiple grids, then the grid that contains the midpoint of that object is responsible for detecting that object.

Info It is advisable to use a 19x19 grid as a general practice. Moreover, it is less probable that the midpoint of an object will lie in two separate grids.

So far, we are studying the approaches to determine the object; the next topic deals with measuring the performance of that detection which is the Intersection over Union.

5.6 Intersection over Union (IoU)

We have examined some of the methods for object detection. And in the subsequent sections, we will study Deep Learning architectures too. But still, we have to determine the accuracy of our predictions in object detection. Intersection over Union is a test to ascertain how close is our prediction to the actual truth.

It is represented by Equation 5-1 and is shown in Figure 5-4.

IoU = Overlapping region/Combined entire region (Equation 5-1)

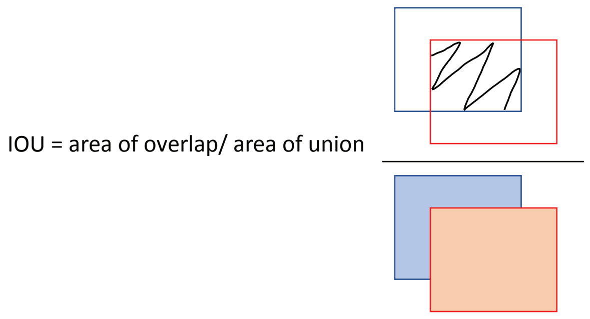

Figure 5-4. *Intersection over Union is used to measure the performance of detection. The numerator is the common area, while the denominator is the complete union of the two areas. The higher the value of IoU, the better it is*

So, if we get a higher value of Intersection over Union, it means the overlap is better. Hence, the prediction is more accurate and better. It is depicted in the example in Figure 5-5 to visualize.

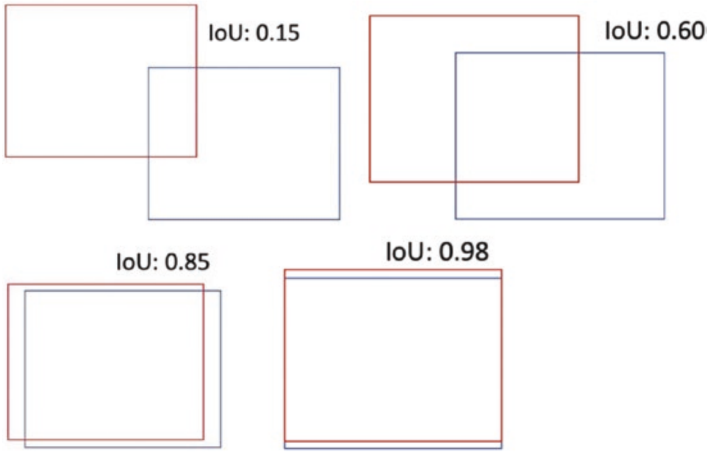

***Figure 5-5.** IoU values for different positions of the overlapping blocks. If the value is closer to 1.0, it means that the detection is more accurate as compared to the value of 0.15*

As we can see in Figure 5-5, for IoU of 0.15, there is very less overlap between the two boxes as compared to 0.85 or 0.90. It means that the one with 0.85 IoU is a better solution to the one with 0.15 IoU. The detection solution can hence be compared directly.

Intersection over Union allows us to measure and compare the performance of various solutions. It also makes it easier for us to distinguish between useful bounding boxes and not-so-important ones. Intersection over Union is an important concept with wide usages. Using it, we can compare and contrast the acceptability of all the possible solutions and choose the best one from them.

We will now study non-max suppression techniques which are useful to filter significant bounding boxes.

5.7 Non-max suppression

When we are trying to detect an object in an image, we can have an object in multiple grids. It can be represented in Figure 5-6. And obviously, the grid with the highest probability will be the final prediction for that object.

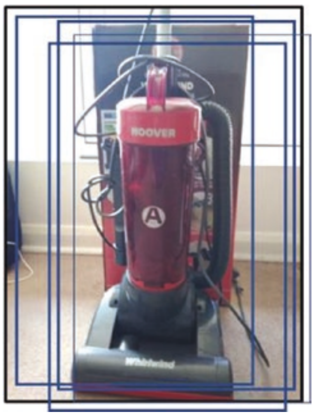

Figure 5-6. *One object can be across multiple grids, and whichever gives the best result is the chosen grid to be used finally for detection purpose*

This entire process is done in the following steps:

1. Get the respective probabilities for all the grids.
2. Set a threshold for the probability and threshold for IoU.
3. Discard the ones which are below that threshold.
4. Choose the box with the best probability.

5. Calculate the IoU of the remaining boxes.
6. Discard the ones which are below the IoU threshold.

Using non-max suppression, we prune most of the bounding boxes which are below a certain level of threshold.

Info Generally, the value is kept at 0.5. You are advised to iterate with different values to analyze the difference.

Hence, the important and significant ones are kept by the algorithm, while the noisier ones are removed. We will now proceed to anchor boxes, another important ingredient in object detection processes.

5.8 Anchor boxes

We wish to detect objects in Deep Learning, and we need a fast and accurate method to get the location and size of the object. An anchor box is a helpful concept for detecting objects.

Anchor boxes are used to capture the scaling and aspect ratio of the objects we wish to detect. They are of predefined size (height and width) and are sized based on the size of the object we want to detect. We are showing anchor boxes in Figure 5-7.

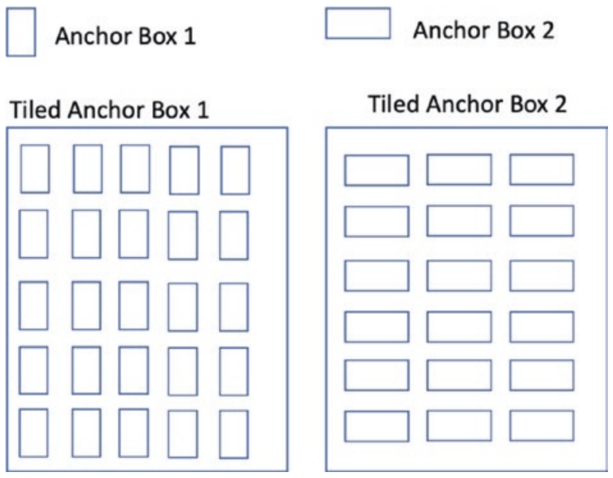

Figure 5-7. *Anchor boxes are used to capture the scaling and aspect ratio. We can tile the anchor boxes, and the Neural Network will output a unique set of predictions*

During the process of object detection, each anchor box is tiled across the image, and the Neural Network outputs a unique set of predictions for each of the anchor boxes. The output consists of the probability score, IoU, background, and offset for each anchor box. Based on the predictions made, the anchor boxes can be further refined.

We can have anchor boxes of multiple sizes to detect objects of different sizes. Using anchor boxes, we can detect objects of different scales. Even to the extent that multiple or overlapping objects can be detected using anchor boxes. It is surely a great improvement over sliding windows as we can now process the entire image in a single shot, making faster real-time object detection possible. To be noted is that the network does not predict the bounding boxes. The network predicts the probability scores and refinements for the tiled anchor box.

Now, we have studied the key components; now let's start with the Deep Learning architecture.

5.9 Deep Learning architectures

Deep Learning helps in object detection. We can detect objects of interest in an image or in a video or even in the live video stream. We are going to create a live video stream solution later in the chapter.

We have seen earlier that there are some problems with the sliding window approach. Objects can have different locations in an image and can be of different aspect ratio or size. An object might be covering the entire region; on the other hand, somewhere it will be covering a small percentage only. There might be more than one object present in the image. The objects can be at various angles or dimensions. Or one object can lie in multiple grids. And moreover, some use cases require real-time predictions. It results in having a very large number of regions and hence huge computation power. It will take a considerable amount of time too. The traditional approaches of image analysis and detection will not be of much help in such situations. Hence, we require Deep Learning–based solutions to resolve and develop robust solutions for object detection.

Deep Learning–based solutions allow us to train better and hence get better results. We are discussing the architectures now.

Let's start with R-CNN as the very first architecture!

5.9.1 Region-based CNN (R-CNN)

We understand that having a very large number of regions is a challenge. Ross Girshick et al. proposed R-CNN to address the problem of selecting a large number of regions. R-CNN is Region-based CNN architecture. Instead of classifying a huge number of regions, the solution suggests to use selective search and extract only 2000 regions from the image. They are called "Region Proposals."

The architecture for R-CNN is shown in Figure 5-8.

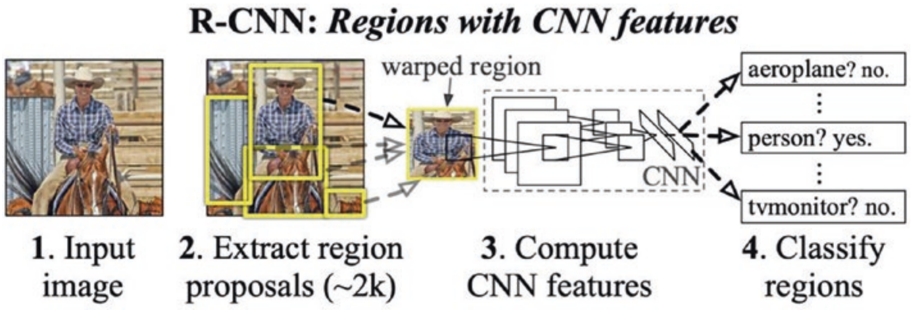

Figure 5-8. *The process in R-CNN. Here, we extract region proposals from the input image, compute the CNN features, and then classify the regions. Image source:* `https://arxiv.org/pdf/1311.2524.pdf` *and published here with the permission of the researchers*

With reference to Figure 5-8 where we have shown the process, let us understand the entire process in detail now:

1. The first step is to input an image, represented by step 1 in Figure 5-8.

2. Then get the regions we are interested in, which is shown in step 2 in Figure 5-8. These are the 2000 proposed regions. They are detected using the following steps:

   a) We create the initial segmentation for the image.

   b) Then we generate the various candidate regions for the image.

   c) We combine similar regions into larger ones iteratively. A greedy search approach is used for it.

   d) Finally, we use the generated regions to output the final region proposals.

3. Then in the next step, we reshape all the 2000 regions as per the implementation in the CNN.
4. We then pass through each region through CNN to get features for each region.
5. The extracted features are now passed through a support vector machine to classify the presence of objects in the region proposed.
6. And then, we predict the bounding boxes for the objects using bounding box regression. This means that we are making the final prediction about the image. As shown in the last step, we are making a prediction if the image is an airplane or a person or a TV monitor.

The preceding process is used by R-CNN to detect the objects in an image. It is surely an innovative architecture, and it proposes a region of interest as an impactful concept to detect objects.

But there are a few challenges with R-CNN, which are

1. R-CNN implements three algorithms (CNN for extracting the features, SVM for the classification of objects, and bounding box regression for getting the bounding boxes). It makes R-CNN solutions quite slow to be trained.
2. It extracts features using CNN for each image region. And the number of regions is 2000. It means if we have 1000 images, the number of features to be extracted is 1000 times 2000 which again makes it slower.

3. Because of these reasons, it takes 40–50 seconds to make a prediction for an image, and hence it becomes a problem for huge datasets.
4. Also, the selective search algorithm is fixed, and not much improvements can be made.

As R-CNN is not very fast and quite difficult to implement for huge datasets, the same authors proposed Fast R-CNN to overcome the issues. Let's understand the improvements suggested in the next section!

5.10 Fast R-CNN

In R-CNN, since we extract 2000 region proposals for an image, it is computationally a challenge to train or test the images. To tackle this issue, Ross Girshick et al. proposed that instead of executing CNN 2000 times for each image, we run it only once for an image and get all the regions of interest.

The architecture of the network is shown in Figure 5-9.

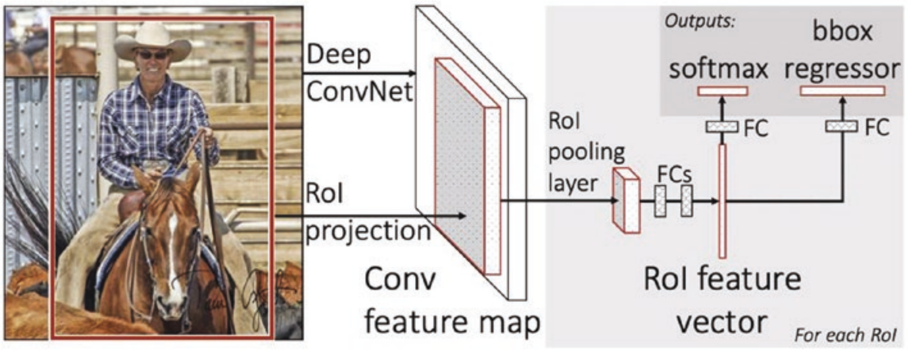

Figure 5-9. *The process in Faster R-CNN. We get the region of interest, and then we reshape all the inputs by applying a region of interest pooling layer. Then they are assessed by an FC layer, and finally softmax does the classification. Image source:* https://arxiv.org/pdf/1504.08083.pdf *and published here with the permission of the researchers*

The approach is similar to its predecessor barring the few changes:

1. The image is an input as shown in Figure 5-9.
2. The image is passed to the convolutional network which returns the respective regions of interest.
3. In the next step, we apply the region of interest pooling layer. It results in reshaping all the regions as per the input of the convolution. So it makes the size of all the regions of interest the same by applying the ROI pooling layer.
4. Now each of these regions is passed to the fully connected network.
5. Finally, the classification is done by the softmax layer. In parallel, the coordinates of the bounding boxes are identified using a bounding box regressor.

Fast R-CNN has a few advantages over R-CNN:

1. Fast R-CNN does not require feeding of 2000 region proposals to the CNN every time, and hence it is faster than R-CNN.
2. It uses only one convolution operation per image instead of three (extracting features, classification, and generating bounding boxes) used in the case of R-CNN. And hence there is no need to store a feature map, resulting in saving disk space.
3. Generally, softmax layers have better accuracy than SVM and have faster execution time.

The Fast R-CNN reduced the training time significantly and proved to be much more accurate too. But still the performance is not significantly fast enough due to the use of selective search as the proposed method to get the regions of interest. Hence, for a large dataset, the prediction is not fast enough. And that is why we have Faster R-CNN which we are discussing next.

5.11 Faster R-CNN

To overcome the slowness in R-CNN and Fast R-CNN, Shaoqing Ran et al. proposed Faster R-CNN. The intuition behind the Faster R-CNN is to replace the selective search which is slow and time-consuming. Faster R-CNN uses the Regional Proposal Network or RPN. The paper can be accessed at https://papers.nips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

The architecture of Faster R-CNN is shown in Figure 5-10.

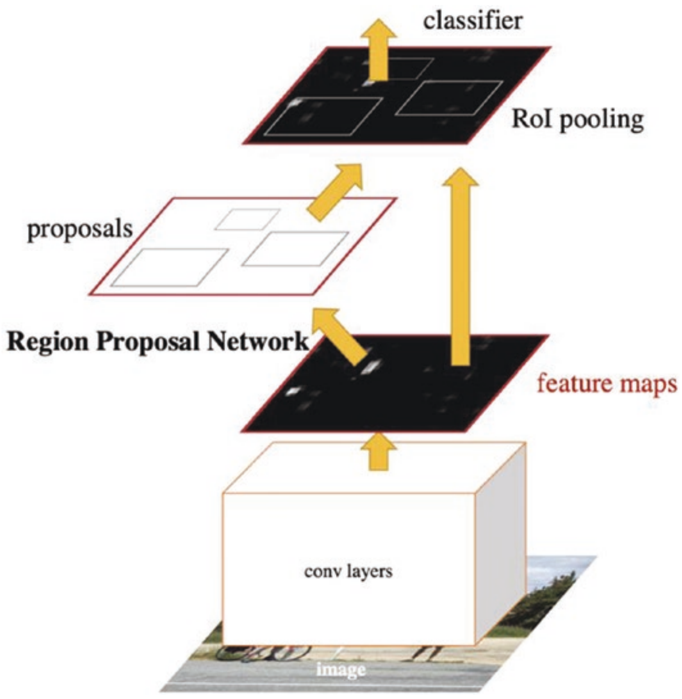

Figure 5-10. *Faster R-CNN is an improvement over the previous versions. It consists of two modules – one is a deep convolutional network, and the other is the Fast R-CNN detector*

To quote from the original paper at https://papers.nips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf:

> *Our object detection system, called Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions. The entire system is a single, unified network for object detection.*

Let's dive deep into the architecture. The way a Faster R-CNN works is as follows:

1. We take an input image and make it pass through CNN as shown in Figure 5-10.
2. From the feature maps received, we apply Region Proposal Networks (RPNs). The way an RPN works can be understood by referring to Figure 5-11.

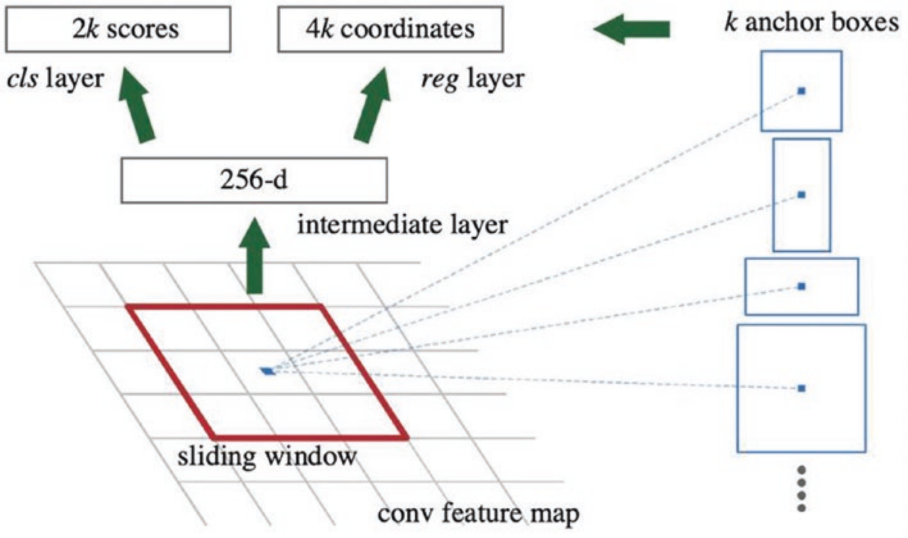

Figure 5-11. *Region proposal networks are used in Faster R-CNN. The image has been taken from the original paper*

The substeps followed are

a) RPN takes the feature maps generated from the last step.

b) RPN applies a sliding window and generates k anchor boxes. We have discussed anchor boxes in the last section.

c) The anchor boxes generated are of different shapes and sizes.

d) RPN will also predict that an anchor is an object or not.

e) It will also give the bounding box regressor to adjust the anchors.

f) To be noted is RPN has not suggested the class of the object.

g) We will get object proposals and the respective objectness scores.

3. Apply ROI pooling to make the size of all the proposals the same.

4. And then, finally, we feed them to the fully connected layers with softmax and linear regression.

5. We will receive the predicted Object Classification and respective bounding boxes.

Faster R-CNN is able to combine the intelligence and use deep convolution fully connected layers and Fast R-CNN using proposed regions. The entire solution is a single and unified solution for object detection.

Though Faster R-CNN is surely an improvement in terms of performance over R-CNN and Fast R-CNN, still the algorithm does not analyze all the parts of the image simultaneously. Instead, each and every part of the image is analyzed in a sequence. Hence, it requires a large number of passes over a single image to recognize all the objects. Moreover, since a lot of systems are working in a sequence, the performance of one depends on the performance of the preceding steps.

We will now proceed to one of the most famous algorithms – YOLO or You Only Look Once – in the next section.

5.12 You Only Look Once (YOLO)

You Only Look Once or YOLO is targeted for real-time object detection. The previous algorithms we discussed use regions to localize the objects in the image. Those algorithms look at a part of the image and not the complete image, whereas in YOLO a single CNN predicts both the bounding boxes and the respective class probabilities. YOLO was proposed in 2016 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. The actual paper can be accessed at https://arxiv.org/pdf/1506.02640v5.pdf.

To quote from the actual paper, "We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities."

As shown in Figure 5-12, YOLO divides an image into a grid of cells (represented by S). Each of the cells predicts bounding boxes (represented by B). Then YOLO works on each bounding box and generates a confidence score about the goodness of the shape of the box. The class probability for the object is also predicted. Finally, the bounding box having class probability scores above are selected, and they are used to locate the object within that image.

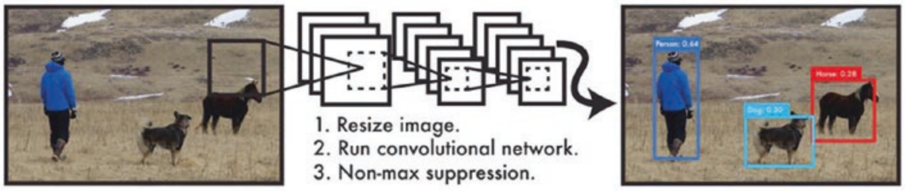

Figure 5-12. *The YOLO process is simple; the image has been taken from the original paper* https://arxiv.org/pdf/1506.02640v5.pdf

5.12.1 Salient features of YOLO

1. YOLO divides the input image into an SxS grid. To be noted is that each grid is responsible for predicting only one object. If the center of an object falls in a grid cell, that grid cell is responsible for detecting that object.

2. For each of the grid cells, it predicts boundary boxes (B). Each of the boundary boxes has five attributes – the x coordinate, y coordinate, width, height, and a confidence score. In other words, it has (x, y, w, h) and a score. This confidence score is the confidence of having an object inside the box. It also reflects the accuracy of the boundary box.

3. The width w and height h are normalized to the images' width and height. The x and y coordinates represent the center relative to the bound of the grid cells.

4. The confidence is defined as Probability(Object) times IoU. If there is no object, the confidence is zero. Else, the confidence is equal to the IoU between the predicted box and ground truth.

5. Each grid cell predicts C conditional class probabilities – Pr(Classi | Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.

6. At the test time, we multiply the conditional class probabilities and the individual class predictions. It gives us the class-specific confidence scores for each box. It can be represented in Equation 5-2:

$$\Pr(\text{Class}_i \mid \text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \qquad \text{(Equation 5-2)}$$

We will now examine how we calculate the loss function in YOLO. It is important to get the loss function calculation function before we can study the entire architecture in detail.

5.12.2 Loss function in YOLO

We have seen in the last section that YOLO predicts multiple bounding boxes for each cell. And we choose the bounding box which has the maximum IoU with the ground truth. To calculate the loss, YOLO optimizes for sum-squared error in the output in the model as sum-squared error is easy to optimize.

The loss function is shown in Equation 5-3 and comprises localization loss, confidence loss, and classification loss. We are first representing the complete loss function and then describing the terms in detail.

$$
\begin{aligned}
&\left.\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]
\end{aligned}\right\} \boxed{\text{Localization Loss}} \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \longrightarrow \boxed{\text{Confidence Loss if an object is detected in the box}} \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \longrightarrow \boxed{\text{Confidence Loss if an object is not detected in the box}} \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \longrightarrow \boxed{\text{Classification Loss}}
\end{aligned}
$$

(Equation 5-3)

In Equation 5-3, we have localization loss, confidence loss, and classification loss, where 1 $^{\text{obj}}{}_{\text{i}}$ denotes if the object appears in cell *i* and 1 $^{\text{obj}}{}_{\text{ij}}$ denotes that the *j*th bounding box predictor in cell *i* is "responsible" for that prediction.

Let's describe the terms in the preceding equation. Here, we have

A. Localization loss is to measure the errors for the predicted boundary boxes. It measures their location and size errors. In the preceding equation, the first two terms represent the localization loss. 1 $^{\text{obj}}{}_{\text{i}}$ is 1 if the jth boundary box in cell i is responsible for detecting the object, else the value is 0. λ_{coord} is responsible for the increase in the weight for the loss in the coordinates of the boundary boxes. The default value of λ_{coord} is 5.

B. Confidence loss is the loss if an object is detected in the box. It is the second loss term in the equation shown. In the term earlier, we have

$\hat{C}_i$ is the box confidence score of the box *j* in cell *i*.

$\mathbb{1}_{ij}^{obj} = 1$ if the *j* th boundary box in cell *i* is responsible for detecting the object, otherwise 0.

C. The next term is a confidence loss if the object is not detected. In the term earlier, we have

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

D. The final term is the classification loss. If an object is indeed detected, then for each cell it is the squared error of the class probabilities for each class.

$\mathbb{1}_{i}^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

The final loss is the sum total of all these components. As the objective of any Deep Learning solution, the objective will be to minimize this loss value.

Now we have understood the attributes of YOLO and the loss function; we will now proceed to the actual architecture of YOLO.

5.12.3 YOLO architecture

The network design is shown in Figure 5-13 and is taken from the actual paper at `https://arxiv.org/pdf/1506.02640v5.pdf`.

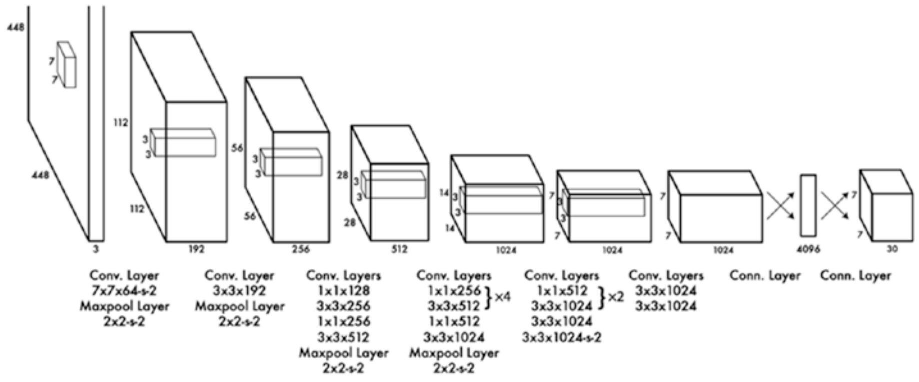

Figure 5-13. *The complete YOLO architecture; the image has been taken from the original paper at* *https://arxiv.org/pdf/1506.02640v5.pdf*

In the paper, the authors have mentioned that the network has been an inspiration from GoogLeNet. The network has 24 convolutional layers followed by 2 fully connected layers. Instead of Inception modules used by GoogLeNet, YOLO uses 1x1 reduction layers followed by 3x3 convolutional layers. YOLO might detect the duplicates of the same object. For this, non-maximal suppression has been implemented. This removes the duplicate lower confidence score.

In Figure 5-14, we have a figure having 13x13 grids. In total, 169 grids are there wherein each grid predicts 5 bounding boxes. Hence, there are a total of 169*5 = 845 bounding boxes. When we apply a threshold of 30% or more, we get 3 bounding boxes as shown in Figure 5-14.

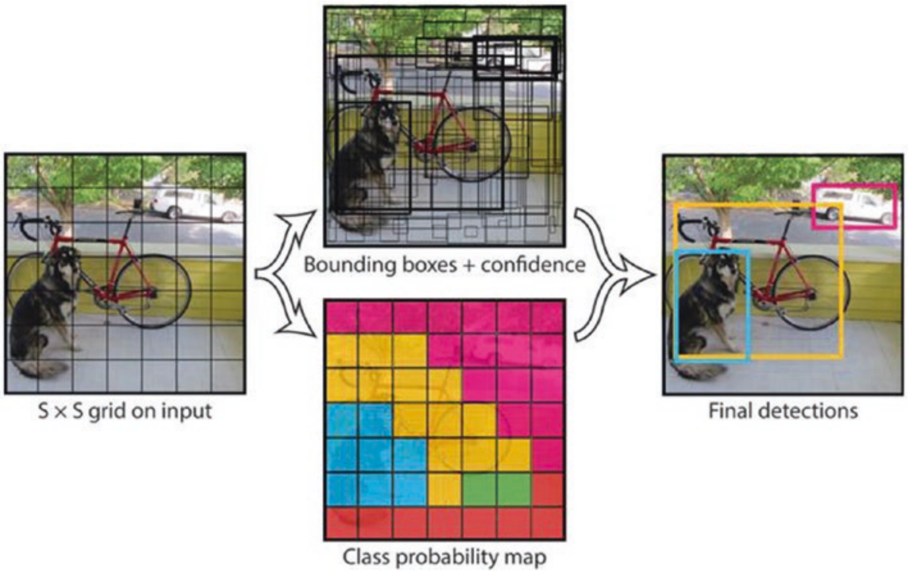

Figure 5-14. *The YOLO process divides the region into SxS grids. Each grid predicts five bounding boxes, and based on the threshold setting which is 30% here, we get the final three bounding boxes; the image has been taken from the original paper*

So, YOLO looks at the image only once but in a clever manner. It is a very fast algorithm for real-time processing. To quote from the original paper:

1. YOLO is refreshingly simple.

2. YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our Neural Network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

3. YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.

4. YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

There are a few challenges with YOLO too. It suffers from high localization error. Moreover, since each of the grid cells predicts only two boxes and can have only one class as the output, YOLO can predict only a limited number of nearby objects. It suffers from a problem of low recall too. And hence in the next version of YOLOv2 and YOLOv3, these issues were addressed. Interested readers can get in-depth knowledge from the official website at https://pjreddie.com/darknet/yolo/.

YOLO is one of the most widely used object detection solutions. Its uniqueness lies in its simplicity and speed. The next Deep Learning architecture we will examine is a Single Shot MultiBox Detector or SSD in the next section.

5.13 Single Shot MultiBox Detector (SSD)

We have so far discussed R-CNN, Fast R-CNN, Faster R-CNN, and YOLO in the last sections. To overcome the slowness in the networks to work in real-time object detection, C. Szegedy et al. proposed the SSD (Single Shot MultiBox Detector) network in November 2016. The paper can be accessed at https://arxiv.org/pdf/1512.02325.pdf.

SSD uses the VGG16 architecture which we have discussed in the previous chapters but with a few modifications. By using an SSD, only a single shot is required to detect multiple images in an object. It is hence called *single* shot since it utilizes a *single* forward pass for both object localization and classification. Regional Proposal Network (RPN)–based solutions like R-CNN, Fast R-CNN, need two shots – first one to get the region proposals and second to detect the object for each proposal. And hence SSD proves to be much faster than RPN-based approaches. Szegedy et al. called it *multibox,* and the significance of the word detector is obvious. Let's explore more on the multibox detector concept.

Refer to Figure 5-15. We can say that after applying and passing through a series of convolutions, we obtain a feature layer of size m x n and p channels.

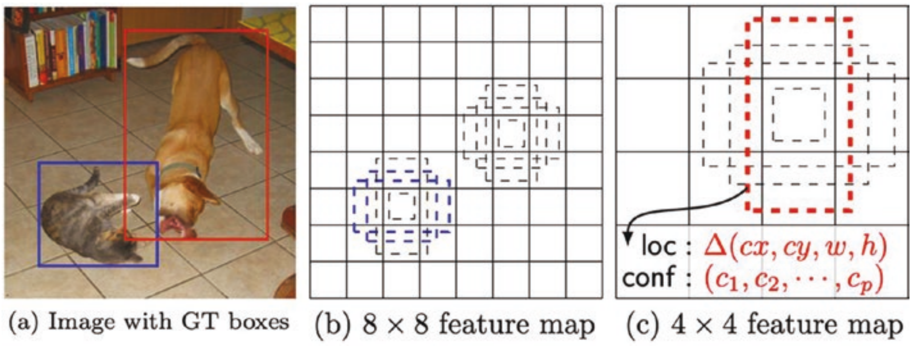
loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Figure 5-15. *SSD process is shown. We have an original image with ground truth (GT). 8x8 convolutions are done. We get different bounding boxes of sizes and location; the image has been taken from the original paper*

For each of the locations, we will get k bounding boxes which might be of varying sizes and aspect ratios. And for each of these k bounding boxes, we calculate c class scores and four offsets relative to the original default bounding box to finally receive (c+4)kmn outputs.

SSD implements a smooth L1 norm to calculate the location loss. It might not be as accurate as L1 but still be quite reasonably accurate.

More about multibox methods can be read at https://arxiv.org/abs/1412.1441.

The complete network is shown in Figure 5-16 as a comparison to YOLO. The image has been taken from the same paper with permission from the authors.

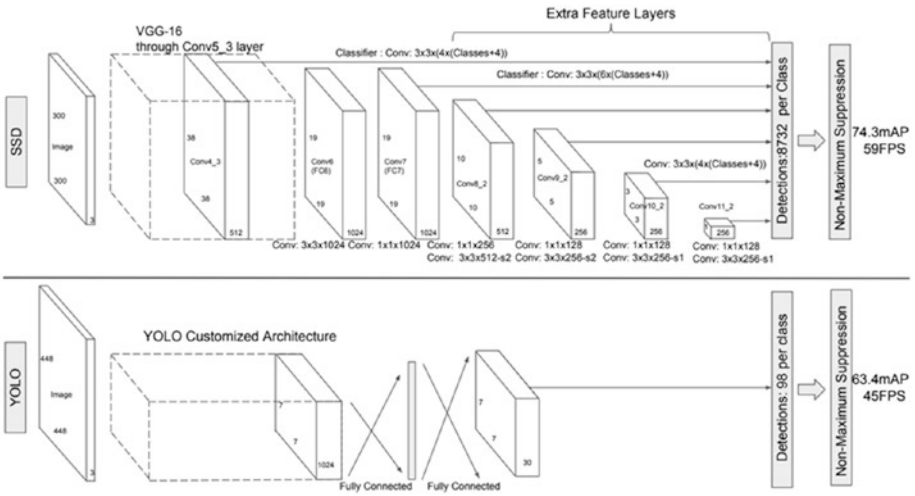

Figure 5-16. *A comparison of YOLO and SSD is shown here. The image is taken from the original paper at https://arxiv.org/pdf/1512.02325.pdf*

In SSD, different layers of feature maps are being passed through the 3x3 convolution layer to improve accuracy. If we analyze the preceding structure, we can observe that for the first layer of object detection (conv4_3), it has spatial dimensions of 38x38 which is quite a reduction in size, resulting in low accuracy for predicting smaller-sized objects. For the same conv4_3, we can calculate the output by using the formula in the discussion earlier. For conv4_3, the output will be 38x38x4x(c+4) where c is the number of classes to be predicted.

SSD uses two loss functions to calculate the loss – confidence loss (L_{conf}) and localization loss (L_{loc}). L_{conf} is the loss in making a class prediction, while L_{loc} is the mismatch between the ground truth and the predicted box. The mathematical formulas for both the losses are given in the previously mentioned paper, and their derivation is beyond the scope of the book.

There are a few other important processes followed in SSD:

1. Data augmentation by flipping, cropping, and color distortion is done to improve the accuracy. Each of the training examples is randomly sampled as follows:

    a. Utilize the original image.

    b. Sample a patch with IoU of 0.1, 0.3, 0.5, 0.7, or 0.9.

    c. Randomly sample a patch.

    d. The sampled patch has an aspect ratio between 0.5 and 2, and the size of each sampled patch is [0.1, 1] of the original size.

    Then each sampled image is resized to a fixed size and flipped horizontally. Photo distortions are also used for image augmentation.

2. SSD implements non-max suppression to remove the duplicate predictions. We have discussed non-max suppression at the start of this chapter.

3. SSD results in a higher number of predictions than the actual number of objects. We have more negative ones than the positive ones, resulting in class imbalance. To quote the actual paper: “Instead of using all the negative examples, we sort them using the highest confidence loss for each default box and pick the top ones so that the ratio between the negatives and positives is at most 3:1. We found that this leads to faster optimization and a more stable training.”

Based on the preceding architecture, we can conclude a few points about SSD:

1. Detection of small-sized objects can be a challenge. To tackle this problem, we can increase the image resolution.
2. Accuracy is inversely proportional to speed; if we wish to increase the speed, we can increase the number of boundary boxes.
3. SSD has a higher classification error than R-CNN, but the localization error is lesser.
4. It makes good use of smaller convolution filters to predict the class and multiscale feature maps for detection. It helps in improving the accuracy.
5. To quote from the original paper: “The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps.”

The accuracy of the SSD can be further improved. It confuses between objects with similar categories. And it is built on VGG16 which consumes a lot of training time. But SSD is a fantastic solution and can be easily used for end-to-end training. It is fast, can run in real time, and performs better than Faster R-CNN.

With this, we conclude the Deep Learning architectures for object detection. We have discussed the major algorithms, and in subsequent sections we will be developing actual Python code to implement the solution. But before that, we will examine the concept of Transfer Learning. It is an innovative solution which allows us to build on the state-of-the-art algorithms trained by the experts. Transfer learning is the next topic we are discussing.

5.14 Transfer Learning

As the name suggests, transfer learning is sharing the knowledge or transferring the learning to others. In the world of Deep Learning, researchers and organizations innovate and create novel Neural Network architectures. They use state-of-the-art capabilities of multicode powerful processors and train the algorithm on a large dataset carefully curated and chosen.

For us to create such an intelligence will be like reinventing the wheel. Hence, using transfer learning, we make use of those networks which have been trained on millions of data points. This allows us to use the intelligence generated by the researchers and implement the same on a real-world dataset. This process is referred to as *transfer learning*.

In transfer learning, we use a pre-trained model for our purpose. The pre-trained model has the final weights from the original model. The basic idea of using a pre-trained model is that the initial layers of the network detect the basic features, and as we move deeper, the features start to take shape. The basic feature extraction can be used for any type of images. So, if a model has been trained to distinguish between mobile phones, it can be used to distinguish between cars.

We can show the process in Figure 5-17. Here, the first layers of the network are frozen and will be used to extract the low-level features like edges, lines, and so on. And the last layers can be customized as per the business problem.

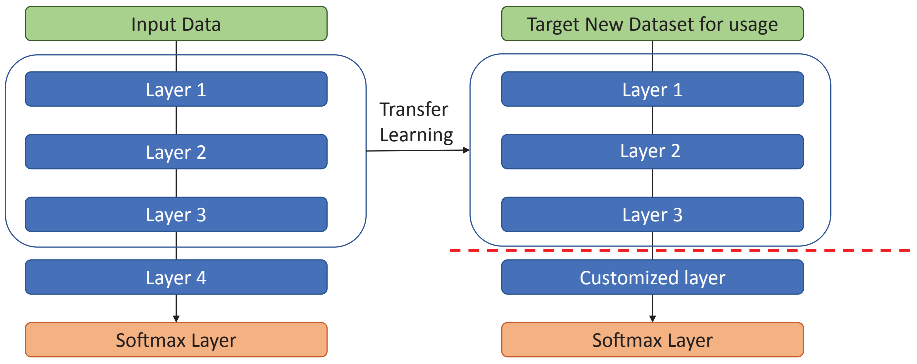

Figure 5-17. *Transfer learning utilizes the pre-trained network. The first layers are responsible for extracting the low-level features and are frozen. The last layers are customized for the problem at hand*

Transfer learning makes the learning faster than traditional machine learning and requires less training data. We will be discussing more details of the pre-trained model in Chapter 8.

Transfer learning is solving real-world business problems by leveraging the solutions developed in a different setting. We are going to use transfer learning now in the next section and in subsequent chapters of the book. In the previous chapters, we can employ transfer learning to use those networks.

Enough of theory, it's time to develop our Object Detection solutions. Time to hit the code!

5.15 Python implementation

We are going to implement real-time object detection using YOLO. The pre-trained weights have to be downloaded. The code, weights, labels, and the expected output can be downloaded from the GitHub repository link given at the start of the chapter.

Step 1: Import all the required libraries.

```
import cv2
from imutils.video import VideoStream
import os
import numpy as np
```

Step 2: Load the configurations from the local path. We are loading the weights, the configuration, and the labels. We are also setting the settings for the detection.

```
localPath_labels = "coco.names"
localPath_weights = "yolov3.weights"
localPath_config = "yolov3.cfg"
labels = open(localPath_labels).read().strip().split("\n")
scaling = 0.005
confidence_threshold = 0.5
nms_threshold = 0.005  # Non Maxima Supression Threshold Vlue
model = cv2.dnn.readNetFromDarknet(localPath_config, localPath_
weights)
```

Step 3: Now we are starting with the video in this step. We are then configuring for the layers which are unconnected by accessing from the object model.

```
cap = VideoStream(src=0).start()
layers_name = model.getLayerNames()
output_layer = [layers_name[i[0] - 1] for i in model.
getUnconnectedOutLayers()]
```

Note You are advised to explore the components of the model and print them to understand better.

Step 4: Now we are ready to perform the detection of the objects. This step is the core step for the solution. It detects the object and the boundary boxes and adds the text on top of the box.

We are starting with a while loop at the start. Then we are reading the frame. In the next step, the width and height of the frame are being set up. Then we are going in a loop to iterate through each and every frame. If the confidence is above the confidence threshold we have set up earlier, the object will be detected.

We then label and the respective confidence scores for the object are shown in a box on the detected boundary boxes. The output is shown in Figure 5-18.

```
while True:
    frame = cap.read()
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
    swapRB=True, crop=False)
    model.setInput(blob)
    nnoutputs = model.forward(output_layer)
    confidence_scores = []
    box_dimensions = []
    class_ids = []

    for output in nnoutputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5 :
                box = detection[0:4] * np.array([w, h, w, h])
                (center_x, center_y, width, height) = box.
                astype("int")
                x = int(center_x - (width / 2))
                y = int(center_y - (height / 2))
                box_dimensions.append([x, y, int(width),
                int(height)])
                confidence_scores.append(float(confidence))
                class_ids.append(class_id)
    ind = cv2.dnn.NMSBoxes(box_dimensions, confidence_scores,
    confidence_threshold, nms_threshold)
    for i in ind:
        i = i[0]
        (x, y, w, h) = (box_dimensions[i][0], box_dimensions[i]
        [1],box_dimensions[i][2], box_dimensions[i][3])
        cv2.rectangle(frame,(x, y), (x + w, y + h), (0, 255,
        255), 2)
        label = "{}: {:.4f}".format(labels[class_ids[i]],
        confidence_scores[i])
        cv2.putText(frame, label, (x, y - 5), cv2.FONT_HERSHEY_
        SIMPLEX, 0.5, (255,0,255), 2)
    cv2.imshow("Yolo", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
cv2.destroyAllWindows()
cap.stop()
```

The output has been shown in Figure 5-18. In real time, we are able to detect a cell phone with 99.79% accuracy.

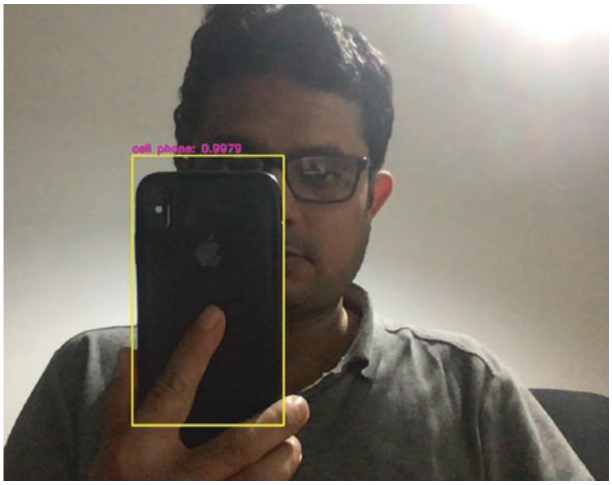

Figure 5-18. *The real-time object detection is shown here. We are able to detect a cell phone with 99.79% accuracy*

In this solution, the real-world objects can be identified. And a bounding box is created around the object along with the name and confidence score.

This solution can be used for multiple use cases. The same code can be customized for datasets and can be used for detecting objects in images and videos too.

We can now proceed to the summary of the chapter.

5.16 Summary

Object detection is a very powerful solution. It is utilized at a number of domains and operations, and almost all the industries can be benefitted from object detection. It can be used for optical character recognition, autonomous driving, tracking objects and people, crowd surveillance, safety mechanisms, and so on. This computer vision technique is really changing the face of real-time capabilities.

In this chapter, we discussed object detection architectures – R-CNN, Fast R-CNN, Faster R-CNN, YOLO, and SSD. All of the networks are Deep Learning based and novel in design and architecture. Yet some outperform others. And generally there is a trade-off of speed and accuracy. So based on the business problem at hand, we have to carefully choose the network.

We also discussed transfer learning in this chapter. Transfer learning is a novel solution to use the pre-trained networks which have been trained on millions of images. Transfer learning allows us to use the intelligence generated by researchers and authors by using powerful processors. It is a tool which is enabling everyone to use these really deep networks and customize them as per the need. We used transfer learning to use the pre-trained YOLO to detect the object in real time. We are going to employ the transfer learning methodology moving ahead in other chapters.

Object detection can be in many practical solutions, but the input dataset defined the final accuracy of the solution. Hence, if you are using the networks for implementation on a custom dataset, be ready for some serious work in the data collection phase. The data will decide and define your success!

Now, in the next chapter, we are going to work on another exciting topic – face detection and recognition. Let's continue this journey!

You should be able to answer the questions in the exercise now!

REVIEW EXERCISES

You are advised to solve these questions:

1. What is the concept of anchor boxes and non-max suppression?
2. How are bounding boxes important for object detection?
3. How are R-CNN, Fast R-CNN, and Faster R-CNN different and what are the improvements?

4. How does Transfer Learning improve the Neural Network solution?

5. Download the Open Images 2019 dataset from `www.kaggle.com/c/open-images-2019-object-detection` and use it to create a solution using YOLO.

6. Download the chess dataset from `https://public.roboflow.com/object-detection/chess-full` and use it to locate the chess pieces based on the networks used in the chapter.

7. Get the racoon dataset from `https://public.roboflow.com/object-detection/raccoon` and use it to create an object detection solution.

8. Get the COCO dataset from `https://cocodataset.org/#home` and use it to compare the performance using different networks.

9. Download the Vehicles-OpenImages dataset from `https://public.roboflow.com/object-detection/vehicles-openimages` and create an object detection solution.

5.16.1 Further readings

1. Explore the paper "The Object Detection Based on Deep Learning": `https://ieeexplore.ieee.org/document/8110383`.

2. Explore the paper "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications": `https://arxiv.org/pdf/1704.04861v1.pdf`.

3. Explore the paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks": `https://arxiv.org/pdf/1801.04381v4.pdf`.
4. Explore the paper "Searching for MobileNetV3": `https://arxiv.org/pdf/1905.02244v5.pdf`.