Object Detection and Obstacle avoidance

Dr. Mustafa shible

Mahmoud Salah 20200722

Intro:

Object detection and obstacle avoidance are critical components of computer vision and robotics systems. These technologies empower machines to perceive and interact with their environments, enabling a wide range of applications, from autonomous vehicles navigating busy streets to industrial robots safely working alongside humans. In a similar way we can use these methods as a replacement for human disabilities and help not only robots and autonomous vehicles, but also impaired people navigate a world made for visual creatures.

In the realm of computer vision, object detection refers to the process of identifying and locating objects within an image or video frame it's called localization. It involves pinpointing the objects' positions by drawing bounding boxes around them and often categorizing the objects into predefined classes. Object detection forms the foundation for various applications, such as surveillance, face recognition, and content-based image retrieval.

Obstacle avoidance, on the other hand, is a specialized subset of object detection tailored for navigation and robotics. It focuses on identifying obstacles or objects in the path of a moving agent, like a drone or a mobile robot, and determining how to navigate around them safely. This capability is particularly crucial for autonomous systems, ensuring they can navigate complex and dynamic environments while avoiding collisions.

In this context, we'll delve into the methodologies and technologies that underpin object detection and obstacle avoidance, exploring approaches ranging from classical computer vision techniques to cutting-edge deep learning models like Faster R-CNN, YOLO, and SSD. Understanding these methods is essential for engineers and researchers working on the development of intelligent systems capable of perceiving and interacting with the world around them.

Unlike **Image Classification** where we assume there's only one object present in the image in object detection we expect multi-objects on a single frame and the task is to identify what and where each object is, **Object Detection is usually modeled as a classification problem** where we take windows of fixed sizes from input image at all the possible locations and feed them to the above mentioned image classifier. Each window is fed to the classifier which predicts the class of the object in the window(or background if none is present). Hence, we know both the class and location of the objects in the image.

But how do we know that our window will even contain a valid object, objects in a frame can be of varying sizes for example an airplane close on the ground would be huge part of the frame while a far away flying plane would be very small which in turn would require different varying window size the normal way is to actually brute force this approach into scanning the image multiple times with different sizes in a pyramid like way either bottom-up or top-bottom approach

Figure 2 Small sized object

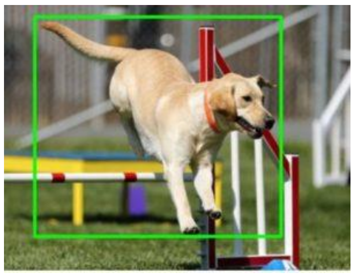


Figure 1 Big sized object

As you can see that the object can be of varying sizes. To solve this problem an image pyramid is created by scaling the image.
Idea is that we resize the image at multiple scales and we count on the fact that our chosen window size will completely contain the object in one of these resized images. Most commonly, the image is down sampled (size is reduced) until certain condition typically a minimum size is reached. On each of these images, a fixed size window detector is run. It's common to have as many as 64 levels on such pyramids. Now, all these windows are fed to a classifier to detect the object of interest. This will help us solve the problem of size and location.

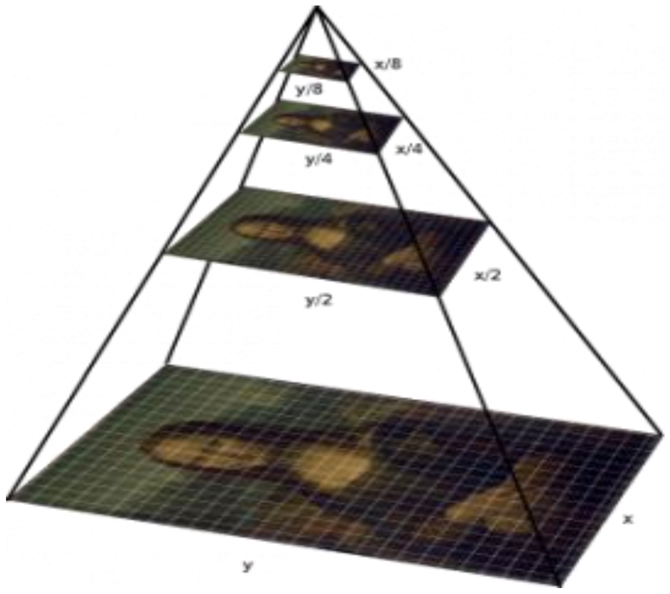


Figure 3 Down Sampling of an Image

Different approaches and methods have been developed over the years from simple pattern recognition to our most recent deep learning and CNN neural networks like:

- **HOG (Histogram of Oriented Gradients)**
- **R-CNN (Region based convolution neural network)**
- **SPP-net (Spatial pyramid pooling)**
- **Fast R-CNN**
- **Faster R-CNN**
- **YOLO**
- **SSD (Single Shot Detector) (the one used in the demo app)**
- **EfficientDet**

Algorithms:

1- HOG:

In a groundbreaking paper in the history of computer vision, **Navneet Dalal and Bill Triggs** introduced Histogram of Oriented Gradients(HOG) features in 2005. Hog features are computationally inexpensive and are good for many real-world problems. On each window obtained from running the sliding window on the pyramid, we calculate Hog Features which are fed to an SVM(Support vector machine) to create classifiers. HOG works by extracting local gradient information from an image. It identifies regions of an image where the intensity and texture change significantly, which are often indicative of object boundaries and shapes which is a much better approach than the brute force method of scanning down sampled images we introduced earlier. it calculates the gradient magnitude and orientation for each pixel in the image. The gradient represents the rate of intensity change at each pixel. HOG is commonly used in object detection tasks, such as pedestrian detection in surveillance systems and face detection in computer vision applications. It's particularly effective when combined with a classifier that can discriminate between objects and non-objects based on the extracted features. While HOG has been surpassed by deep learning-based approaches in many scenarios, it remains a valuable tool for certain object detection tasks, especially when computational resources are limited.

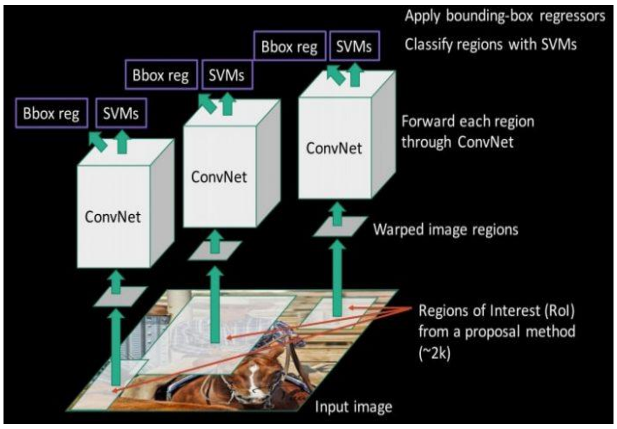
2- R-CNN:

Since we had modeled object detection into a classification problem, success depends on the accuracy of classification. After the rise of deep learning, the obvious idea was to replace HOG based classifiers with a more accurate convolutional neural network based classifier. However, there was one problem. CNNs were too slow and computationally very expensive. It was impossible to run CNNs on so many patches generated by sliding window detector. R-CNN solves this problem by using an object proposal algorithm called **Selective Search** which reduces the number of bounding boxes that are fed to the classifier to close to 2000 region proposals. Selective search uses local cues like texture, intensity, color and/or a measure of insideness etc to generate all the possible locations of the object. Now, we can feed these boxes to our CNN based classifier. Remember, fully connected part of CNN takes a fixed sized input so, we resize(without preserving aspect ratio) all the generated boxes to a fixed size (224×224 for VGG) and feed to the CNN part. Hence, there are 3 important parts of R-CNN:

I. Run Selective Search to generate probable objects.

II. Feed these patches to CNN, followed by SVM to predict the class of each patch.

III. Optimize patches by training bounding box regression separately.

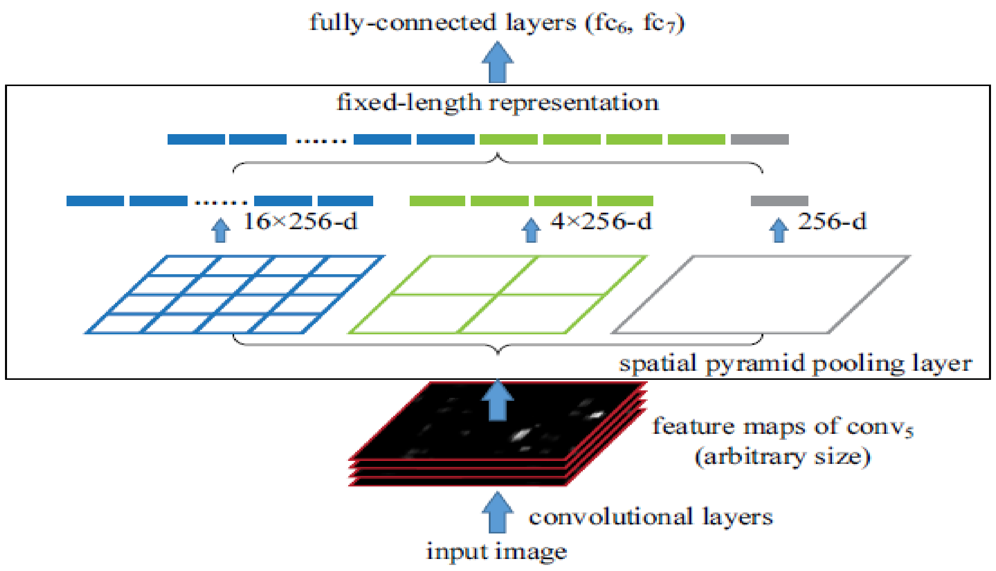

Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Forward each region through ConvNet
ConvNet
ConvNet
ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

3- Spatial pyramid pooling (SPP-net):

Still, **RCNN was very slow**. Because running CNN on 2000 region proposals generated by Selective search takes a lot of time. SPP-Net tried to fix this. **With SPP-net, we calculate the CNN representation for entire image only once and can use that to calculate the CNN representation for each patch generated by Selective Search**. This can be done by performing a pooling type of operation on JUST that section of the feature maps of last conv layer that corresponds to the region. The rectangular section of conv layer corresponding to a region can be calculated by projecting the region on conv layer by taking into account the downsampling happening in the intermediate layers(simply dividing the coordinates by 16 in case of VGG).

There was one more challenge: we need to generate the fixed size of input for the fully connected layers of the CNN so, SPP introduces one more trick. **It uses spatial pooling after the last convolutional layer as opposed to traditionally used max-pooling**. SPP layer divides a region of any arbitrary size into a constant number of bins and max pool is performed on each of the bins. Since the number of bins remains the same, a constant size vector is produced as demonstrated in the figure below.

However, there was one big drawback with SPP net, it was not trivial to perform back-propagation through spatial pooling layer. Hence, the network only fine-tuned the fully connected part of the network. SPP-Net paved the way for more popular Fast RCNN which we will see next.


fully-connected layers (fc_6, fc_7)
fixed-length representation
16×256-d
4×256-d
256-d
spatial pyramid pooling layer
feature maps of $conv_5$ (arbitrary size)
convolutional layers
input image

4- Fast R-CNN:

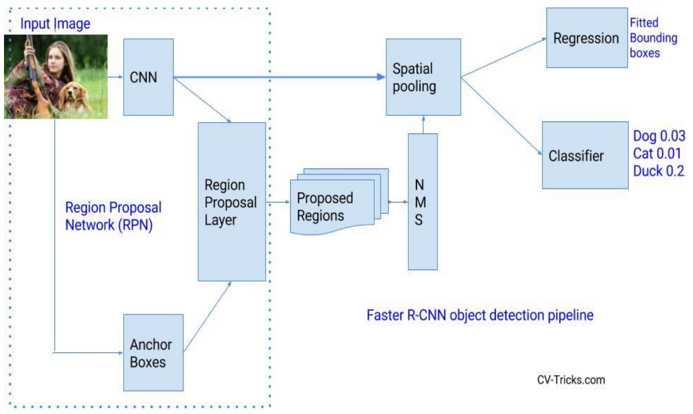
Fast RCNN uses the ideas from SPP-net and RCNN and fixes the key problem in SPP-net i.e. **they made it possible to train end-to-end**. To propagate the gradients through spatial pooling, It uses a simple back-propagation calculation which is very similar to max-pooling gradient calculation with the exception that pooling regions overlap and therefore a cell can have gradients pumping in from multiple regions.

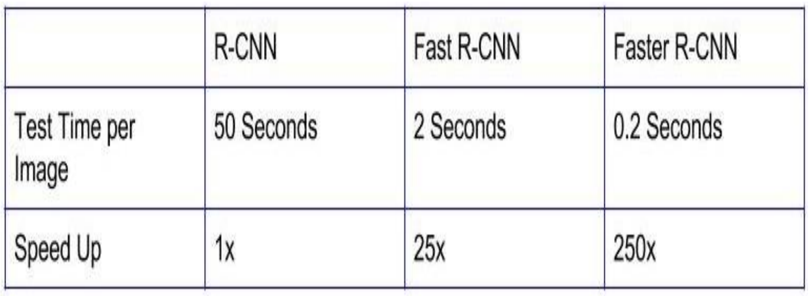
One more thing that Fast RCNN did that they added the bounding box regression to the neural network training itself. So, now the network had two heads, classification head, and bounding box regression head. This multitask objective is a salient feature of Fast-rcnn as it no longer requires training of the network independently for classification and localization. These two changes reduce the overall training time and increase the accuracy in comparison to SPP net because of the end to end learning of CNN.

5. Faster R-CNN:

So, what did Faster RCNN improve? Well, it's faster. And How does it achieve that? Slowest part in Fast RCNN was **Selective Search** or **Edge boxes**. Faster RCNN replaces selective search with a very small convolutional network called **R**egion **P**roposal **N**etwork to generate regions of Interests

To handle the variations in aspect ratio and scale of objects, Faster R-CNN introduces the idea of **anchor boxes**. At each location, the original paper uses 3 kinds of anchor boxes for scale **128x 128, 256×256 and 512×512**. Similarly, for aspect ratio, **it uses three aspect ratios 1:1, 2:1 and 1:2**. So, In total at each location, we have 9 boxes on which RPN predicts the probability of it being background or foreground. We apply bounding box regression to improve the anchor boxes at each location. So, RPN gives out bounding boxes of various sizes with the corresponding probabilities of each class. The varying sizes of bounding boxes can be passed further by apply Spatial Pooling just like Fast-RCNN. The remaining network is similar to Fast-RCNN. Faster-RCNN is 10 times faster than Fast-RCNN with similar accuracy of datasets like VOC-2007. That's why Faster-RCNN has been one of the most accurate object detection algorithms. Here is a quick comparison between various versions of RCNN.





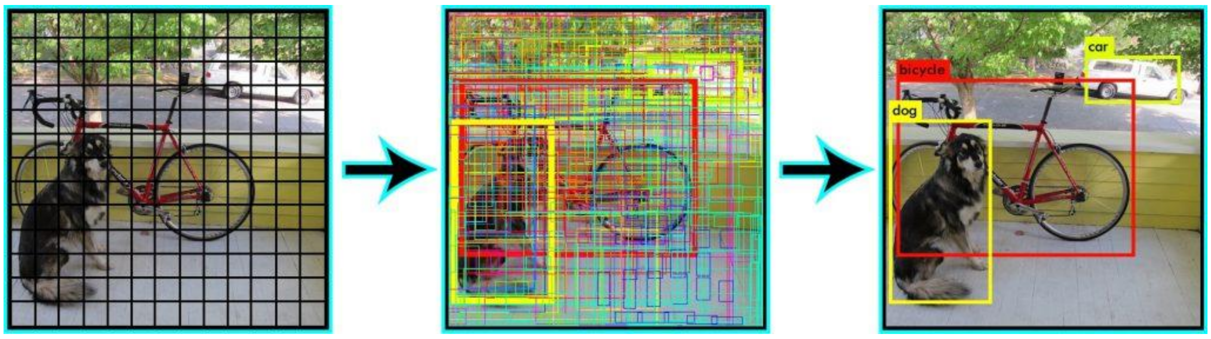
	R-CNN	Fast R-CNN	Faster R-CNN
Test Time per Image	50 Seconds	2 Seconds	0.2 Seconds
Speed Up	1x	25x	250x

6. YOLO(You only Look Once):

So far, all the methods discussed handled detection as a classification problem by building a pipeline where first object proposals are generated and then these proposals are send to classification/regression heads. However, there are a few methods that pose detection as a regression problem. Two of the most popular ones are YOLO and SSD. These detectors are also called single shot detectors.

YOLO, detection is a simple regression problem which takes an input image and learns the class probabilities and bounding box coordinates, it divides each image into a grid of S x S and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object(regardless of class). YOLO also predicts the classification score for each box for every class in training. You can combine both the classes to calculate the probability of each class being present in a predicted box.

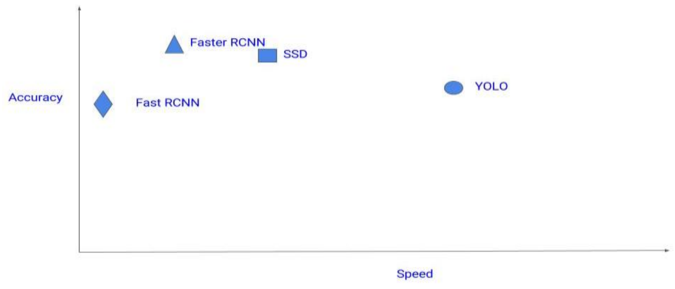
So, total SxSxN boxes are predicted. However, most of these boxes have low confidence scores and if we set a threshold say 30% confidence, we can remove most of them as shown in the example below.



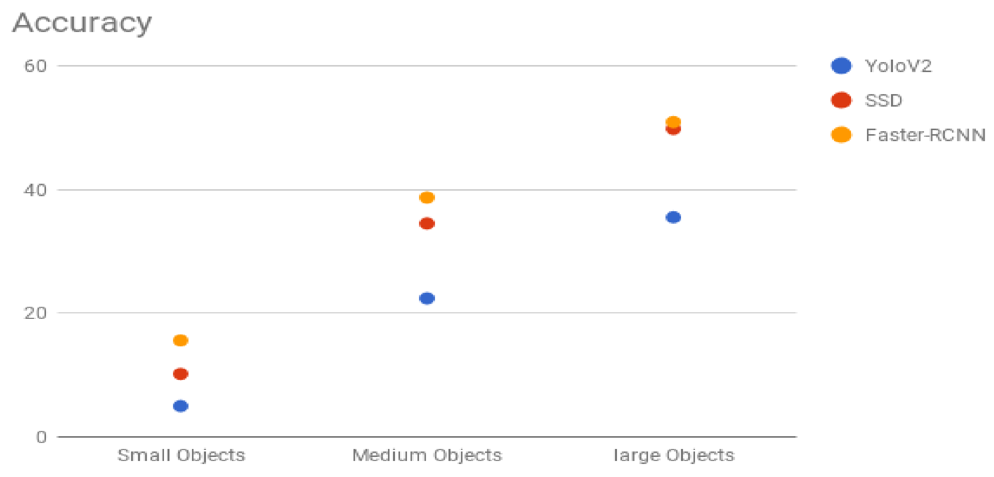
7. Single Shot Detector(SSD):

Single **S**hot **D**etector achieves a good balance between speed and accuracy. SSD runs a convolutional network on input image only once and calculates a feature map. Now, we run a small 3×3 sized convolutional kernel on this feature map to predict the bounding boxes and classification probability. SSD also uses anchor boxes at various aspect ratio like Faster-RCNN and learns the off-set rather than learning the box. In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it can detect objects of various scales.

That’s a lot of algorithms. Which one should you use? Currently, Faster-RCNN is the choice if you are fanatic about the accuracy numbers. However, if you are strapped for computation(probably running it on Nvidia Jetsons), SSD is a better recommendation. Finally, if accuracy is not too much of a concern but you want to go super fast, YOLO will be the way to go. First of all a visual understanding of speed vs accuracy trade-off:



SSD seems to be a good choice as we can run it on a video and the accuracy trade-off is very little. However, it is not that simple, at below chart, At large sizes, SSD seems to perform similarly to Faster-RCNN. However, look at the accuracy numbers when the object size is small, the gap widens.



Choice of a right object detection method is crucial and depends on the problem you are trying to solve and the set-up. Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many industrial applications. Hopefully, this post gave you an intuition and understanding behind each of the popular algorithms for object detection.

Datasets:

1. **COCO (Common Objects in Context):**
   - COCO is one of the most widely used object detection datasets. It contains a large number of images with objects from 80 different classes, including people, animals,
2. **PASCAL VOC (Visual Object Classes):**
   - The PASCAL VOC dataset includes images from various categories, such as animals, vehicles, and indoor objects. It is often used for benchmarking object detection models.
3. **ImageNet Object Detection Dataset:**
   - ImageNet, originally known for image classification, also provides a dataset with object detection annotations. It includes a wide range of objects from various categories.
4. **Open Images Dataset:**
   - Open Images is a large-scale dataset containing millions of images annotated with object bounding boxes. It covers a wide variety of objects and scenes.
5. **KITTI Vision Benchmark Suite:**
   - KITTI is often used for tasks related to autonomous driving and robotics. It includes annotated images and videos with object detection annotations, particularly for vehicles, pedestrians, and other objects encountered in traffic scenes.
6. **Cityscapes Dataset:**
   - Cityscapes is designed for urban scene understanding and includes high-quality pixel-level annotations for object detection tasks related to urban environments.

When choosing a dataset for the algorithm the size and type of objects need to be in consideration retrained models trained on general datasets like COCO or ImageNet can serve as good starting points for transfer learning, allowing you to fine-tune the models for your specific application. Custom datasets are essential when your application requires object detection for domain-specific objects or scenes not well-represented in existing datasets.

Tools:

Android Studio:

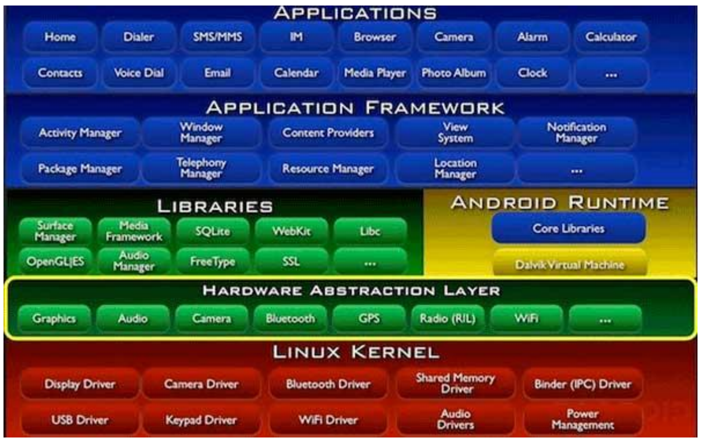
Android Studio is a powerful and indispensable tool in the world of mobile app development. It's the official integrated development environment (IDE) for creating Android applications, offering a comprehensive suite of features and tools that streamline the entire app development process. Whether you're an experienced developer or just beginning your journey into Android app development, Android Studio provides a user-friendly and efficient environment for designing, coding, testing, and deploying Android apps. From its rich code editor to its sophisticated emulator and debugging capabilities, Android Studio empowers developers to transform their creative ideas into functional, polished, and feature-rich Android applications that can be enjoyed by users worldwide. In this ever-evolving landscape of mobile technology, Android Studio stands as a cornerstone, enabling developers to craft innovative and engaging experiences for Android devices.

Native Android Development (Java and Kotlin):

Native Android development is the cornerstone of crafting high-performance, feature-rich, and platform-specific mobile applications for the Android operating system. Unlike cross-platform solutions that target multiple platforms with a single codebase, native Android development focuses on creating apps using the Android platform's native programming languages and tools. This approach, often centered around Java or Kotlin, allows developers to harness the full potential of the Android ecosystem, including access to device hardware, seamless integration with platform-specific features, and a superior user experience.

Native Android development provides a deep level of control over every aspect of an app's behavior and appearance, enabling developers to fine-tune the user interface, optimize performance, and ensure a consistent and polished look and feel that aligns with Android's design principles. With access to the Android Software Development Kit (SDK) and extensive libraries, developers have the tools to implement a wide range of functionalities, from sophisticated animations and custom views to in-depth hardware interactions and sensor integrations.

This approach also facilitates compatibility with the latest Android updates and features, ensuring that your app can take advantage of the newest capabilities and reach the broadest possible audience of Android users. Whether you're building consumer-facing apps, enterprise solutions, or innovative mobile experiences, native Android development empowers you to create applications that seamlessly blend with the Android ecosystem, delivering the best possible user experience on Android devices of all shapes and size

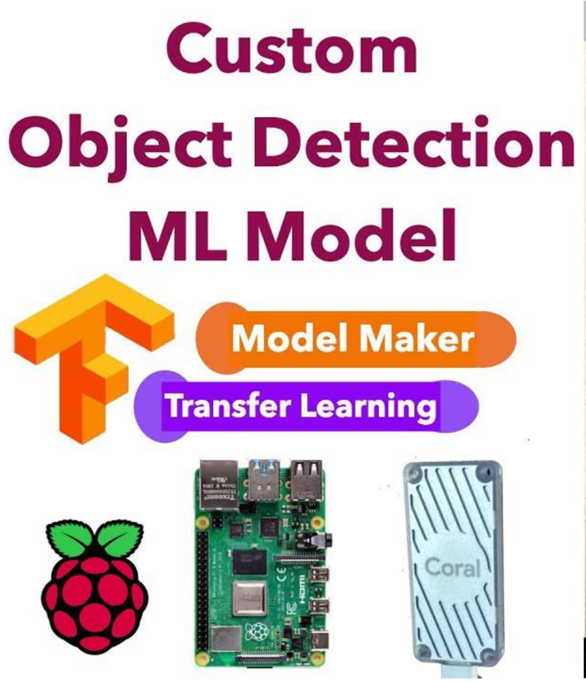


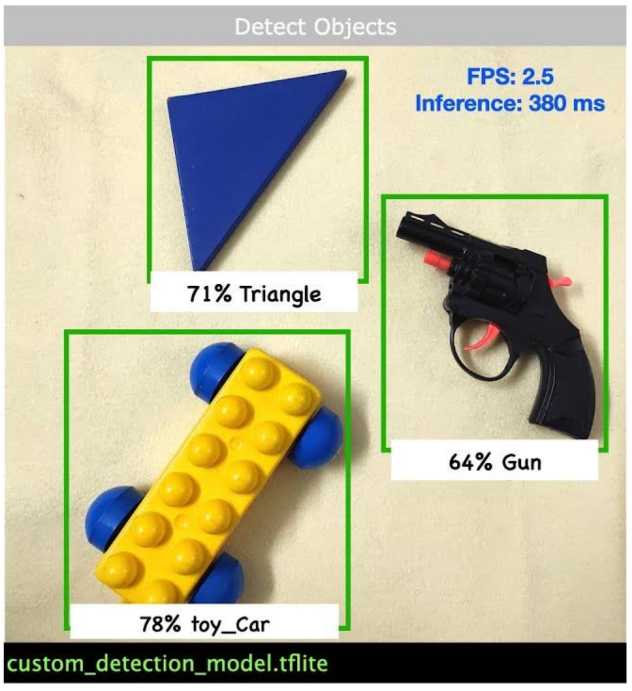
Tensorflow lite:

TensorFlow Lite, often abbreviated as TFLite, represents a remarkable evolution in the world of machine learning and artificial intelligence. It's a lightweight and efficient framework developed by Google that's tailored for deploying machine learning models on resource-constrained devices, such as smartphones, IoT devices, and embedded systems. TensorFlow Lite brings the power of TensorFlow, Google's robust open-source machine learning library, to the edge, enabling developers to integrate machine learning directly into their mobile and embedded applications.

This cutting-edge technology allows you to take advantage of pre-trained machine learning models or build your own custom models using TensorFlow, and then optimize and deploy them efficiently to run locally on devices. By doing so, TensorFlow Lite enables real-time and on-device inferencing, ensuring that your applications can make intelligent decisions without relying on a constant internet connection or cloud resources.

Whether you're interested in creating AI-powered mobile apps, enhancing the capabilities of IoT devices, or developing intelligent robotics applications, TensorFlow Lite provides the tools and flexibility needed to bring the benefits of machine learning to the edge. It empowers developers to design smarter, more responsive, and privacy-friendly applications that can operate seamlessly on a wide range of Android and iOS devices, as well as various embedded platforms. In the ever-expanding landscape of artificial intelligence, TensorFlow Lite stands as a pivotal technology that bridges the gap between powerful machine learning models and the devices that make our daily lives smarter and more efficient.




Detect Objects
FPS: 2.5
Inference: 380 ms
71% Triangle
64% Gun
78% toy_Car
custom_detection_model.tflite