License Plate Recognition System

M.Sc. Computer Science
Final Report – April 2020

Supervisor: Dr. Shan Du

Introduction

With the advent of increasing research in various fields of Computer vision, various applications are used in the real world to benefit the society. The logic and story behind Computer vision incorporate a mixture of modern-day technologies such as Machine Learning, Artificial Intelligence, and Big Data. We use various algorithms so that a computer can use gain a high-level understanding from digital images or videos. There are various systems that extract information from an image or a video. Some of the most common examples of Computer Vision applications include Face Detection, Automatic License Plate Recognition, and Medical Imaging Centres. The applications of Computer vision are endless and continuous innovation is taking place in this field. The concept of License plate recognition evolves from the world of machine learning. Machine learning is the ability of machines to take decisions based on the experiences or history of outputs. Formally speaking, Machine learning (ML) is the study of algorithms and mathematical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning is a subset of Artificial Intelligence (AI). AI is the quest of humans to mimic human behavior in machines. A relatively new subset of ML, Deep Learning further enhances the concept in the form of Neural Networks and enables machines to learn based on existing as well as new data.

Computer Vision is a field that is in constant growth and has the same fundamental goal of Machine Learning, to automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. When we further go into various other sub-fields of Computer Vision, we encounter applications such as Pattern Recognition, Face Detection, Video Tracking, Image Restoration etc. Pattern Recognition consists of several applications such as Character Recognition. We will be focusing on License Plate Recognition (LPR). LPR is the ability of machines or cameras to detect the license plate of a vehicle by using the concepts of Optical character recognition, character segmentation. There is a huge need and value for all the research that is taking place in the field of Computer Vision. A decade ago, computer vision was not as hot as today. With each year, the venture capital funding in Computer Vision is tripling. This growth in the field is supported by advances in Computer Hardware, emergence of Deep Learning and increase in number of datasets. Our goal of imitating Human Vision is very far away. Human Vision is a very complex process and we need to understand how it works. Machine Learning and Artificial Intelligence makes sure that the Computer Vision application is implemented with proper optimizations in its frameworks.

The conventional method of performing LPR follows the following algorithm: -

ALGORITHMS

The software requires seven primary algorithms for identifying a license plate:

1. **Plate localization** – responsible for finding and isolating the plate on the picture.

2. **Plate orientation and sizing** – compensates for the skew of the plate and adjusts the dimensions to the required size.

3. **Normalization** – adjusts the brightness and contrast of the image.

4. **Character segmentation** – finds the individual characters on the plates.

5. **Optical character recognition.**

6. **Syntactical/Geometrical analysis** – check characters and positions against country-specific rules.

7. The averaging of the recognized value over multiple fields/images to produce a more reliable or confident result. Especially since any single image may contain a reflected light flare, be partially obscured or other temporary effect.

Due to the development of smart cities, license plate recognition systems are expected to be integrated into intersection monitors, or streetlights, as intelligent video surveillance systems for traffic detection, automatic charging, or crime detection. Recently, license plate recognition systems have been widely used in parking lots. In order to identify license plates easily, the conventional license plate recognition system used in the parking lot has a fixed light source and a shooting angle. The traditional license plate recognition system first performs license plate detection, then performs horizontal and vertical alignment, and then performs character segmentation to cut individual characters. Finally, character recognition is performed using template matching, machine learning methods (e.g. SVM, KNN), or deep learning methods (e.g. CNN, DNN).

Acknowledgments

I would like to thank my supervisor Dr. Shan Du for her suggestions, advice and direction throughout this project’s development. I must also thank my family and friends for my support and encouragement.

Contents

Chapter 1
INTRODUCTION

Vehicle Number Plate Detection aims at detection of the License Plate present on a vehicle and then extracting the contents of that License Plate. A vehicle's license plate is commonly known as 'a number plate'. It is a metal plate that is attached to a vehicle and has the official registration number of a vehicle embossed on it. Number plates are placed at the front and back of the vehicle and help anyone to identify a vehicle. Motor vehicle registration is the registration of a motor vehicle with a government authority, either compulsory or otherwise. The purpose of motor vehicle registration is to establish a link between a vehicle and an owner or user of the vehicle. This link might be used for taxation or crime detection purposes Contents of Number Plates. The registration identifier is a numeric or alphanumeric code that uniquely identifies the vehicle within the issuing authority database. These number plates can be of different color & have different font and font size depending upon the country and other rules.

Some places where this solution can be used:

· *Analysis of city traffic during peak periods*

· *Automation of weigh-in-motion systems*

· *Enhanced vehicle theft prevention*

· *Effective law enforcement*

· *Effective enforcement of traffic rules*

· *Flexible and automatic vehicle entry to and exit from the car park*

· *Management information about car park usage*

· *Improved security for both car park operators and car park users*

And finally, state border control is one of the most important applications of automatic license plate recognition.

. *Automation and simplicity of airport and harbor logistics*

· *Security monitoring of roads, checkpoints, etc.*

· *Vehicle surveillance · Prevention of non-payment at gas stations, drive-in restaurant*

This process can be achieved by splitting the problem into two components. We can use Object Detection algorithm to detect the license plate from an image or video and then we can perform OCR or Optical Character Recognition on the cropped license plates. By splitting the problem into two components, we have made it easier and efficient to perform License Plate Recognition (LPR). Hence, the problem can be divided into two sub-problems as we mentioned above:

1. **Number Plate Detection.** This problem can be tackled using Object Detection approach where we need to train our model using the car/other vehicle images with number plates.

2. **Extracting text from the detected Number Plate**. This problem can be solved using OCR (Optical Character Recognition) which can be helpful in extracting alphanumeric characters from cropped Number Plate images.

It can be assumed that this solution must be embedded with a web application as well where once the camera captures the image of the vehicle, the backend would call the solution and outputting the contents to the user. Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localization can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation).

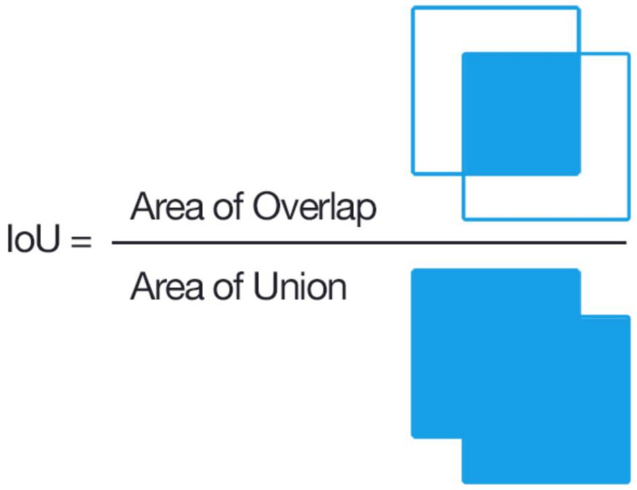
Object detection was studied even before the breakout popularity of CNNs in Computer Vision. While CNNs are capable of automatically extracting more complex and better features, taking a glance at the conventional methods can at worst be a small detour and at best an inspiration. Object detection before Deep Learning was a several step process, starting with edge detection and feature extraction using techniques like SIFT, HOG etc. These images were then compared with existing object templates, usually at multi scale levels, to detect and localize objects present in the image.

Object Detection using bounding boxes

Understanding the Metrics

Intersection over Union (IoU): Bounding box prediction cannot be expected to be precise on the pixel level, and thus a metric needs to be defined for the extend of overlap between 2 bounding boxes.

Intersection over Union *does exactly what it says*. It takes the area of intersection of the 2 bounding boxes involved and divide it with the area of their union. This provides a score, between 0 and 1, representing the quality of overlap between the 2 boxes.

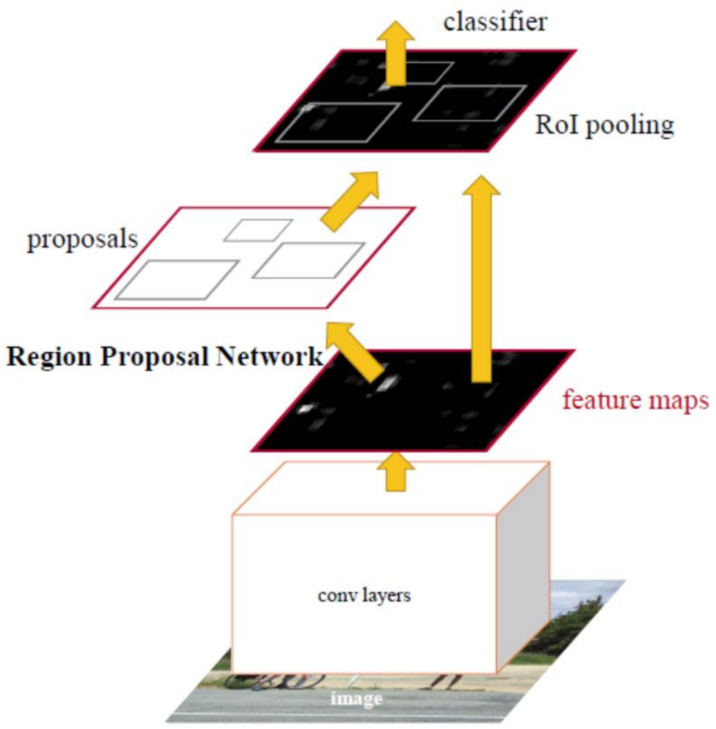


Average Precision and Average Recall: Precision meditates how accurate are our predictions while recall accounts for whether we are able to detect all objects present in the image or not. Average Precision (AP) and Average Recall (AR) are two common metrics used for object detection.

Two-Step Object Detection

Two-Step Object Detection involves algorithms that first identify bounding boxes which may potentially contain objects and then classify each bounding separately.

The first step requires a *Region Proposal Network*, providing several regions which are then passed to common DL based classification architectures. From the hierarchical grouping algorithm in RCNNs (which are extremely slow) to using CNNs and ROI pooling in Fast RCNNs and anchors in Faster RCNNs (thus speeding up the pipeline and training end-to-end), a lot of different methods and variations have been provided to these region proposal networks (RPNs).

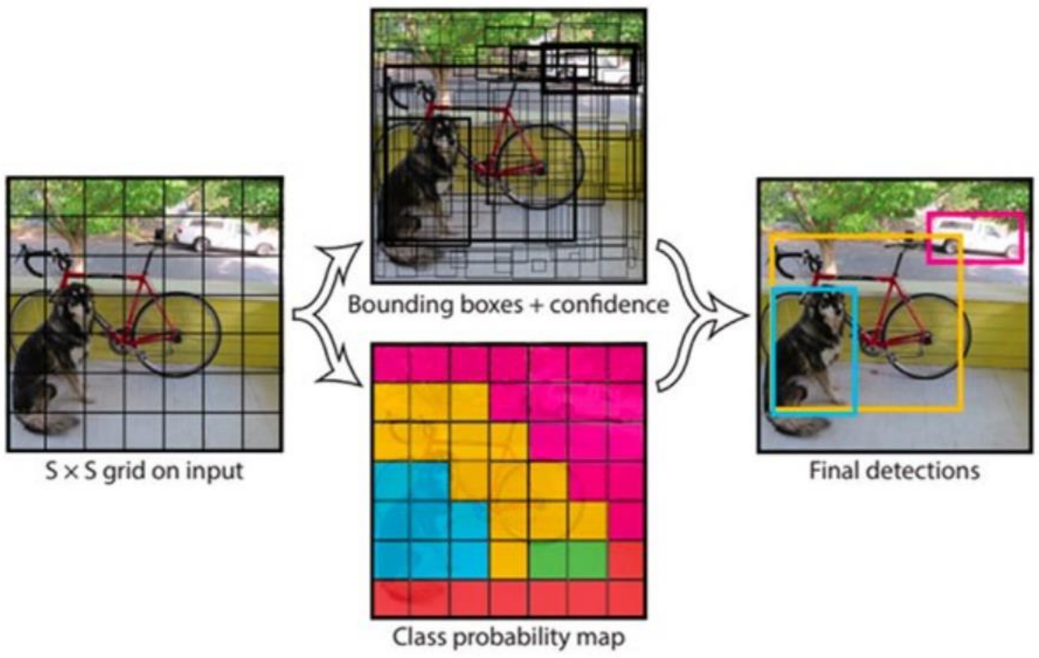


These algorithms are known to perform better than their one-step object detection counterparts but are slower in comparison. With various improvements suggested over the years, the current bottleneck in the latency of Two-Step Object Detection networks is the RPN step.

One-Step Object Detection

With the need of real time object detection, many one-step object detection architectures have been proposed, like YOLO, YOLOv2, YOLOv3, SSD, RetinaNet etc. which try to combine the detection and classification step.

One of the major accomplishments of these algorithms have been introducing the idea of ‘regressing’ the bounding box predictions. When every bounding box is represented easily with a few values (for example, xmin, xmax, ymin and ymax), it becomes easier to combine the detection and classification step and dramatically speed up the pipeline.



For example, YOLO divided the entire image into smaller grid boxes. For each grid cell, it predicts the class probabilities and the x and y coordinates of every bounding box which passes through that grid cell. These modifications allow one-step detectors to run faster and also work on a global level. However, since they do not work on every bounding box separately, this can cause them to perform worse in case of smaller objects or similar object in close vicinity. There have been multiple new architectures introduced to give more importance to lower level features too, thus trying to provide a balance.

Chapter 2

OBJECT DETECTION

It can be challenging for beginners to distinguish between different related computer vision tasks. For example, image classification is straight forward, but the differences between object localization and object detection can be confusing, especially when all three tasks may be just as equally referred to as object recognition.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.

Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs. *Image classification* involves predicting the class of one object in an image. *Object localization* refers to identifying the location of one or more objects in an image and drawing abounding box around their extent. *Object detection* combines these two tasks and localizes and classifies one or more objects in an image.

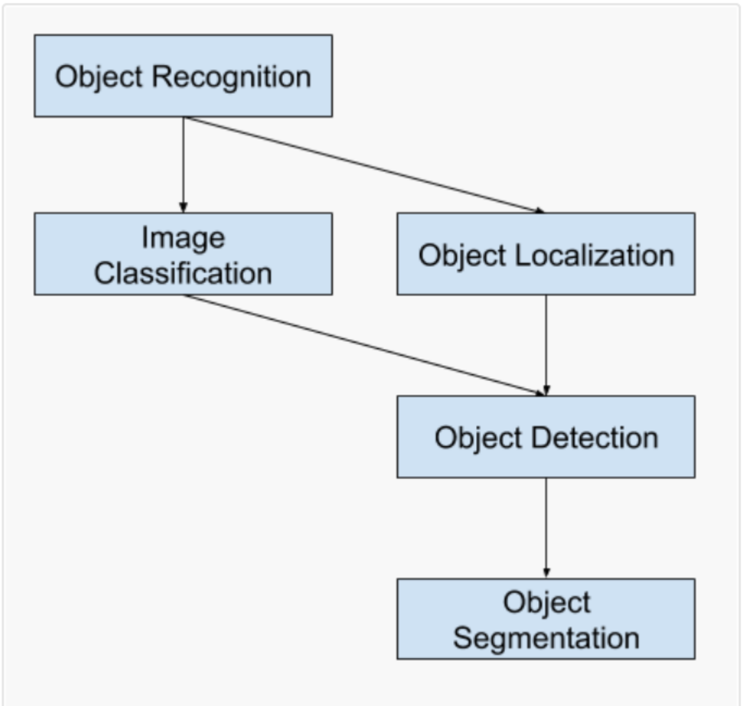
As such, we can distinguish between these three computer vision tasks:

- **Image Classification**: Predict the type or class of an object in an image.
  - *Input*: An image with a single object, such as a photograph.
  - *Output*: A class label (e.g. one or more integers that are mapped to class labels).
- **Object Localization**: Locate the presence of objects in an image and indicate their location with a bounding box.
  - *Input*: An image with one or more objects, such as a photograph.
  - *Output*: One or more bounding boxes (e.g. defined by a point, width, and height).
- **Object Detection**: Locate the presence of objects with a bounding box and types or classes of the located objects in an image.
  - *Input*: An image with one or more objects, such as a photograph.

- *Output*: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

One further extension to this breakdown of computer vision tasks is *object segmentation*, also called "object instance segmentation" or "semantic segmentation," where instances of recognized objects are indicated by highlighting the specific pixels of the object instead of a coarse bounding box.

From this breakdown, we can see that object recognition refers to a suite of challenging computer vision tasks.



Overview of Object Recognition Computer Vision Tasks

This is an annual academic competition with a separate challenge for each of these three problem types, with the intent of fostering independent and separate improvements at each level that can be leveraged more broadly. It is called ImageNet Large Scale Visual Recognition Challenge.

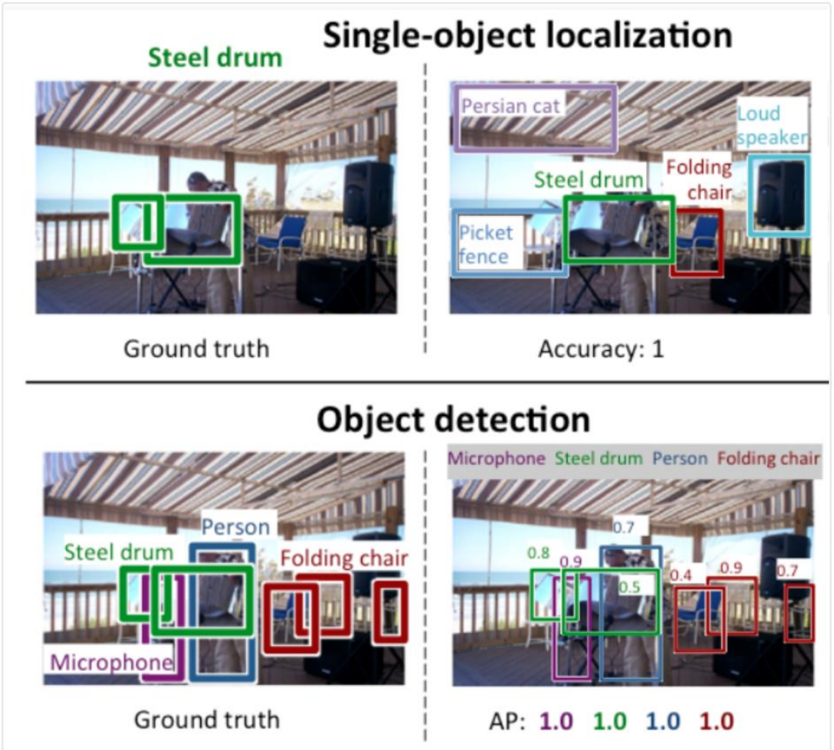
For example, see the list of the three corresponding task types below taken from the 2015 ILSVRC review paper:

- **Image classification**: Algorithms produce a list of object categories present in the image.

- **Single-object localization**: Algorithms produce a list of object categories present in the image, along with an axis-aligned bounding box indicating the position and scale of one instance of each object category.
- **Object detection**: Algorithms produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of every instance of each object category.

We can see that "*Single-object localization*" is a simpler version of the more broadly defined "*Object Localization*," constraining the localization tasks to objects of one type within an image, which we may assume is an easier task.

Below is an example comparing single object localization and object detection, taken from the ILSVRC paper. Note the difference in ground truth expectations in each case.



Comparison Between Single Object Localization and Object Detection.Taken From: ImageNet Large Scale Visual Recognition Challenge.

The performance of a model for image classification is evaluated using the mean classification error across the predicted class labels. The performance of a model for single-object localization is evaluated using the distance between the expected and predicted bounding box for the expected class. Whereas the performance of a model for object recognition is evaluated using the precision and recall across each of the best matching bounding boxes for the known objects in the image.

2.1 R-CNN Model Family

The R-CNN family of methods refers to the R-CNN, which may stand for "Regions with CNN Features" or "Region-Based Convolutional Neural Network," developed by Ross Girshick, et al.

This includes the techniques R-CNN, Fast R-CNN, and Faster-RCNN designed and demonstrated for object localization and object recognition.

2.1.1 R-CNN

The R-CNN was described in the 2014 paper by Ross Girshick, et al. from UC Berkeley titled "Rich feature hierarchies for accurate object detection and semantic segmentation." It may have been one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation. The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset.

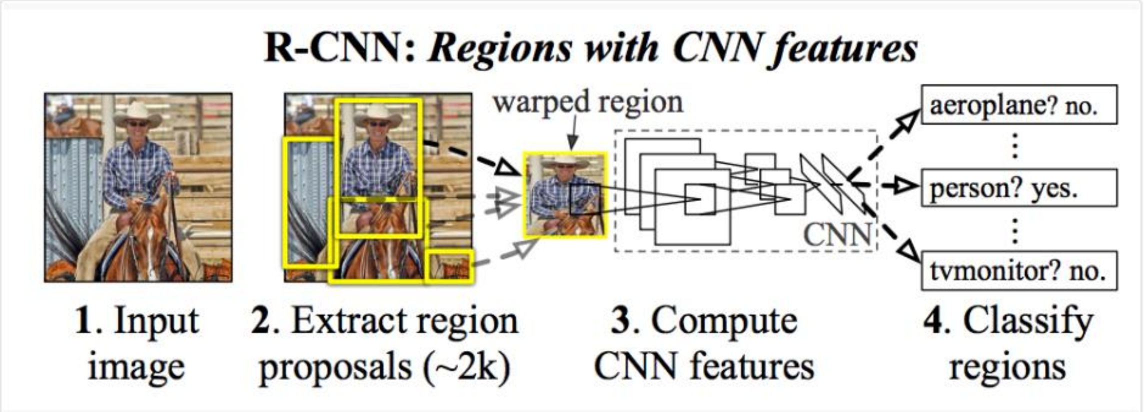
Their proposed R-CNN model is comprised of three modules; they are:

Module 1: Region Proposal. Generate and extract category independent region proposals, e.g. candidate bounding boxes.

Module 2: Feature Extractor. Extract feature from each candidate region, e.g. using a deep convolutional neural network.

Module 3: Classifier. Classify features as one of the known class, e.g. linear SVM classifier model.

The architecture of the model is summarized in the image below, taken from the paper.



Summary of the R-CNN Model ArchitectureTaken from Rich feature hierarchies for accurate object detection and semantic segmentation.

A computer vision technique is used to propose candidate regions or bounding boxes of potential objects in the image called "*selective search*," although the flexibility of the design allows other region proposal algorithms to be used.

The feature extractor used by the model was the AlexNet deep CNN that won the ILSVRC-2012 image classification competition. The output of the CNN was a 4,096-element vector that describes the contents of the image that is fed to a linear SVM for classification, specifically one SVM is trained for each known class. It is a relatively simple and straightforward application of CNNs to the problem of object localization and recognition. A downside of the approach is that it is slow, requiring a CNN-based feature extraction pass on each of the candidate regions generated by the region proposal algorithm. This is a problem as the paper describes the model operating upon approximately 2,000 proposed regions per image at test-time.

Python (Caffe) and MatLab source code for R-CNN as described in the paper was made available in the R-CNN GitHub repository.

2.1.2 Fast R-CNN

Given the great success of R-CNN, Ross Girshick, then at Microsoft Research, proposed an extension to address the speed issues of R-CNN in a 2015 paper titled "Fast R-CNN."

The paper opens with a review of the limitations of R-CNN, which can be summarized as follows:

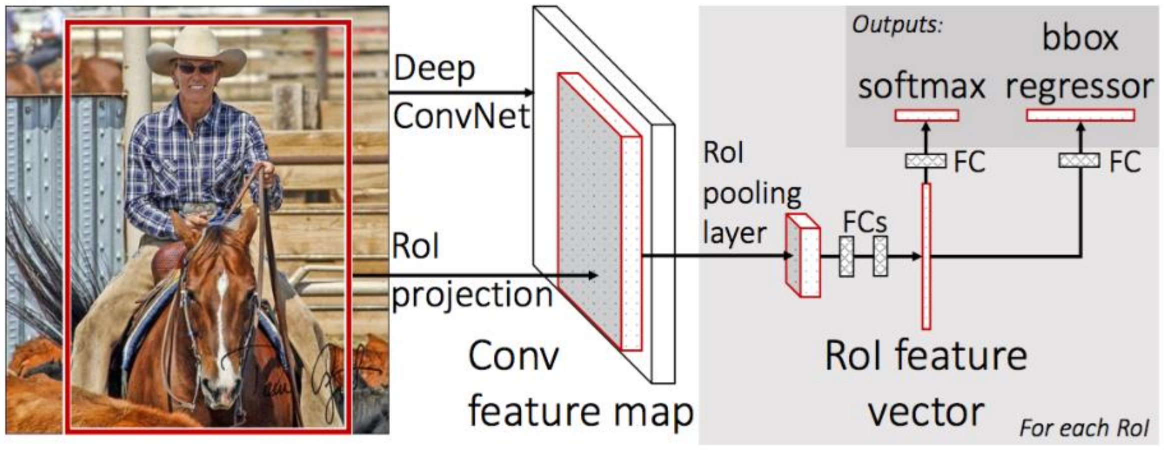
- **Training is a multi-stage pipeline**. Involves the preparation and operation of three separate models.
- **Training is expensive in space and time**. Training a deep CNN on so many region proposals per image is very slow.
- **Object detection is slow**. Make predictions using a deep CNN on so many region proposals is very slow.

A prior work was proposed to speed up the technique called spatial pyramid pooling networks, or SPPnets, in the 2014 paper "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." This did speed up the extraction of features, but essentially used a type of forward pass caching algorithm. Fast R-CNN is proposed as a single model instead of a pipeline to learn and output regions and classifications directly.

The architecture of the model takes the photograph a set of region proposals as input that are passed through a deep convolutional neural network. A pre-trained CNN, such as a VGG-16, is used for feature extraction. The end of the deep CNN is a custom layer called a Region of Interest Pooling Layer, or RoI Pooling, that extracts features specific for a given input candidate region.

The output of the CNN is then interpreted by a fully connected layer then the model bifurcates into two outputs, one for the class prediction via a softmax layer, and another with a linear output for the bounding box. This process is then repeated multiple times for each region of interest in a given image.

The architecture of the model is summarized in the image below, taken from the paper.



Summary of the Fast R-CNN Model Architecture.
Taken from: Fast R-CNN.

The model is significantly faster to train and to make predictions, yet still requires a set of candidate regions to be proposed along with each input image.

Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

2.1.3 Faster R-CNN

The model architecture was further improved for both speed of training and detection by Shaoqing Ren, et al. at Microsoft Research in the 2016 paper titled "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks."

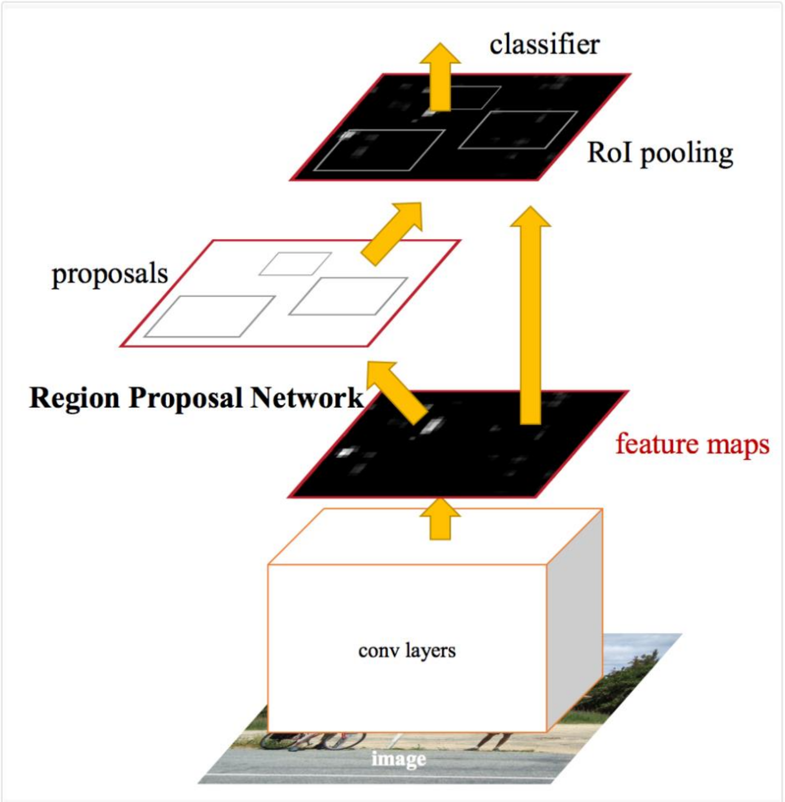
The architecture was the basis for the first-place results achieved on both the ILSVRC-2015 and MS COCO-2015 object recognition and detection competition tasks. The architecture was designed to both propose and refine region proposals as part of the training process, referred to as a Region Proposal Network, or RPN. These regions are then used in concert with a Fast R-CNN model in a single model design. These improvements both reduce the number of region proposals and accelerate the test-time operation of the model to near real-time with then state-of-the-art performance.

Although it is a single unified model, the architecture is comprised of two modules:

- **Module 1: Region Proposal Network**. Convolutional neural network for proposing regions and the type of object to consider in the region.
- **Module 2: Fast R-CNN**. Convolutional neural network for extracting features from the proposed regions and outputting the bounding box and class labels.

Both modules operate on the same output of a deep CNN. The region proposal network acts as an attention mechanism for the Fast R-CNN network, informing the second network of where to look or pay attention.

The architecture of the model is summarized in the image below, taken from the paper.



Summary of the Faster R-CNN Model Architecture.Taken from: Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks.

The RPN works by taking the output of a pre-trained deep CNN, such as VGG-16, and passing a small network over the feature map and outputting multiple region proposals and a class prediction for each. Region proposals are bounding boxes, based on so-called anchor boxes or pre-defined shapes designed to accelerate and improve the proposal of regions. The class prediction is binary, indicating the presence of an object, or not, so-called "*objectness*" of the proposed region.

A procedure of alternating training is used where both sub-networks are trained at the same time, although interleaved. This allows the parameters in the feature detector deep CNN to be tailored or fine-tuned for both tasks at the same time.

At the time of writing, this Faster R-CNN architecture is the pinnacle of the family of models and continues to achieve near state-of-the-art results on object recognition tasks. A further extension adds support for image segmentation, described in the paper 2017 paper "Mask R-CNN."

Python and C++ (Caffe) source code for Fast R-CNN as described in the paper was made available in a GitHub repository.

2.2 YOLO Model Family

Another popular family of object recognition model is referred to collectively as YOLO or "You Only Look Once," developed by Joseph Redmon, et al.

The R-CNN models may be generally more accurate, yet the YOLO family of models are fast, much faster than R-CNN, achieving object detection in real-time.

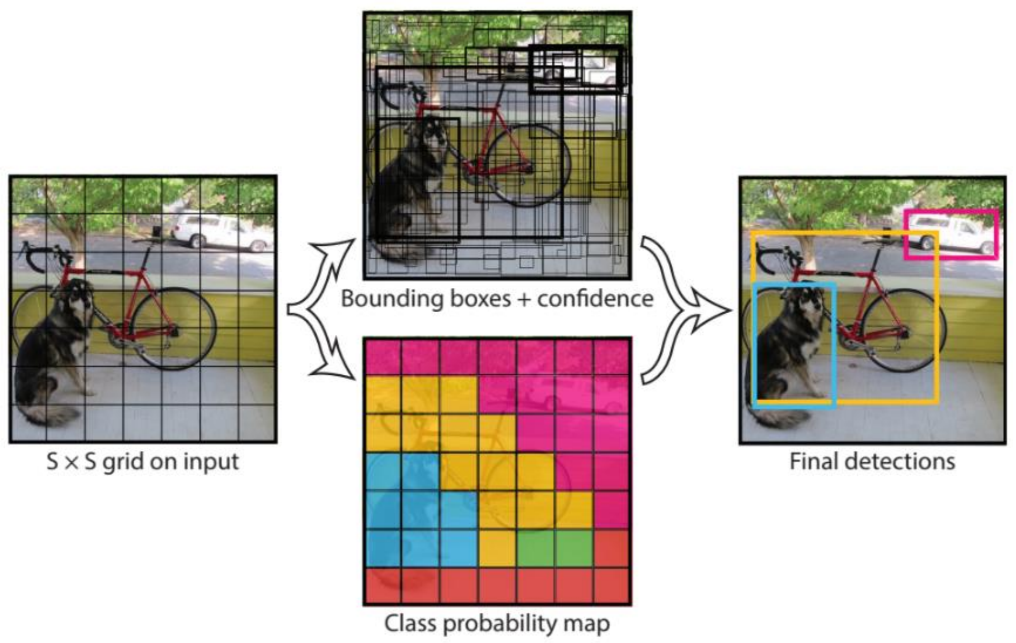
2.2.1 YOLO

The YOLO model was first described by Joseph Redmon, et al. in the 2015 paper titled "You Only Look Once: Unified, Real-Time Object Detection." Note that Ross Girshick, developer of R-CNN, was also an author and contributor to this work, then at Facebook AI Research.

The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model.

The model works by first splitting the input image into a grid of cells, where each cell is responsible for predicting a bounding box if the center of a bounding box falls within it. Each grid cell predicts a bounding box involving the x, y coordinate and the width and height and the confidence. A class prediction is also based on each cell.

For example, an image may be divided into a 7×7 grid and each cell in the grid may predict 2 bounding boxes, resulting in 94 proposed bounding box predictions. The class probabilities map and the bounding boxes with confidences are then combined into a final set of bounding boxes and class labels. The image taken from the paper below summarizes the two outputs of the model.



Summary of Predictions made by YOLO Model.Taken from: You Only Look Once: Unified, Real-Time Object Detection

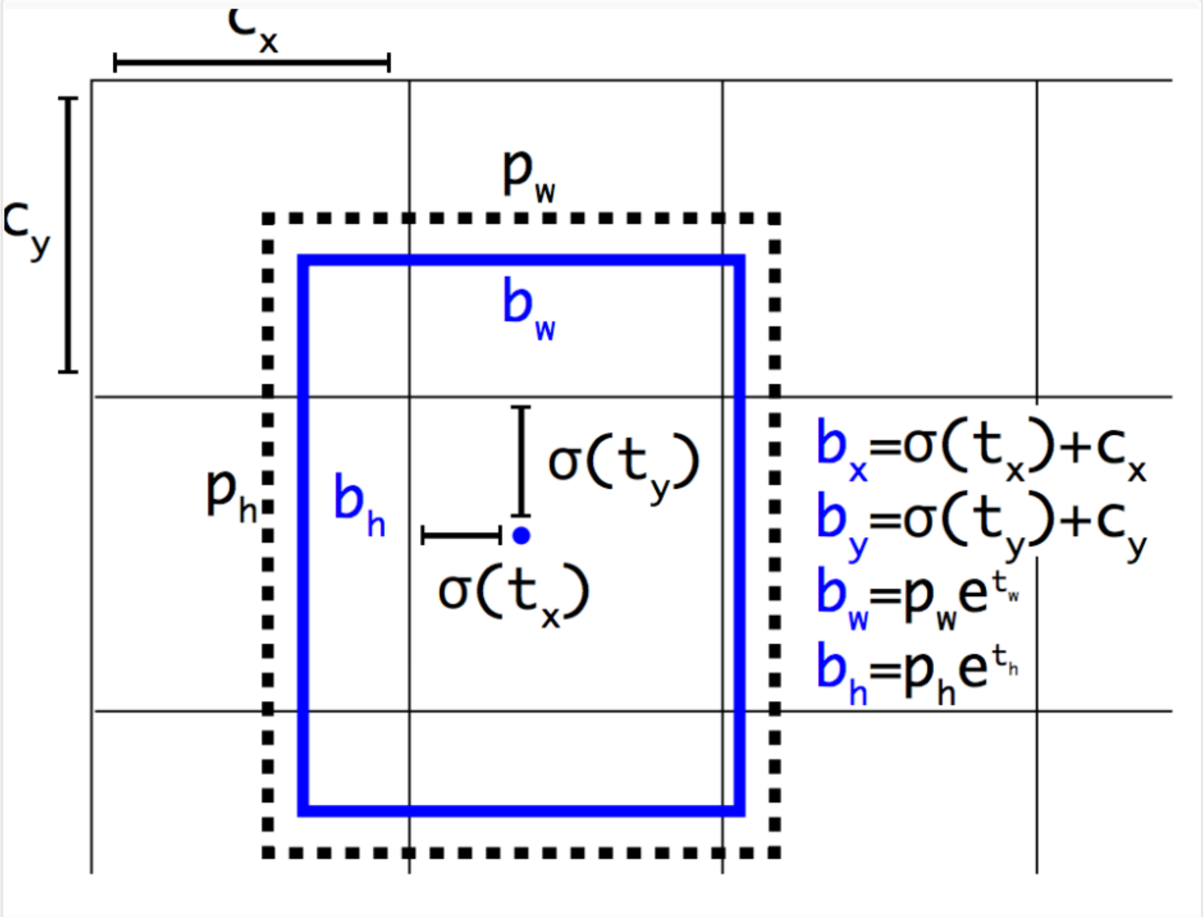
2.2.2 YOLOv2 (YOLO9000) and YOLOv3

The model was updated by Joseph Redmon and Ali Farhadi in an effort to further improve model performance in their 2016 paper titled "YOLO9000: Better, Faster, Stronger."

Although this variation of the model is referred to as YOLO v2, an instance of the model is described that was trained on two object recognition datasets in parallel, capable of predicting 9,000 object classes, hence given the name "YOLO9000."

Several training and architectural changes were made to the model, such as the use of batch normalization and high-resolution input images.

Like Faster R-CNN, YOLOv2 model makes use of anchor boxes, pre-defined bounding boxes with useful shapes and sizes that are tailored during training. The choice of bounding boxes for the image is pre-processed using a k-means analysis on the training dataset.



Example of the Representation Chosen when Predicting Bounding Box Position and ShapeTaken from: YOLO9000: Better, Faster, Stronger

Importantly, the predicted representation of the bounding boxes is changed to allow small changes to have a less dramatic effect on the predictions, resulting in a more stable model. Rather than predicting position and size directly, offsets are predicted for moving and reshaping the pre-defined anchor boxes relative to a grid cell and dampened by a logistic function.

Further improvements to the model were proposed by Joseph Redmon and Ali Farhadi in their 2018 paper titled "YOLOv3: An Incremental Improvement." The improvements were reasonably minor, including a deeper feature detector network and minor representational changes.

Chapter 3
YOLO Model Family

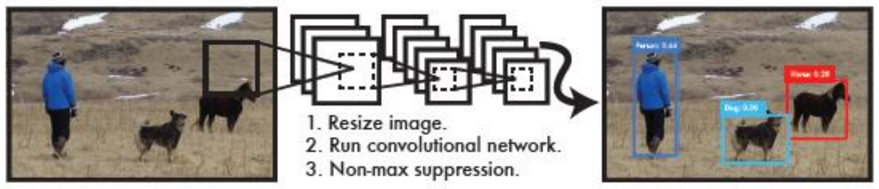
Object Detection has been amongst the hottest streams in Data Science. A lot of models have been explored and gained tremendous success. But the first & foremost that comes to our mind is YOLO i.e. You Look Only Once. Due to its tremendous speed, it has found its application in several live applications. The name itself explains a lot about itself, **that it just goes through the entire image just once**!! Hence, we will be exploring how YOLO works and a comparative study of its different versions.

3.1 YOLOv1

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image.

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene. These complex pipelines are slow and hard to optimize because each individual component must be trained separately. We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see figure below. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.



This unified model has several benefits over traditional methods of object detection. First, 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.

Second, 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. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN. Third, 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.

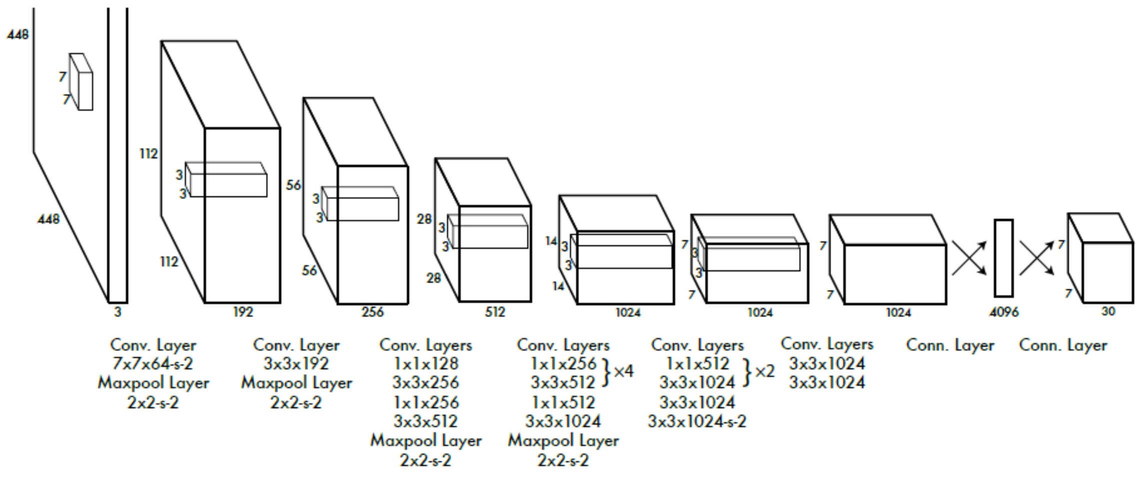
YOLO still lags state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones.

Yolo unifies the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.

It divides the input image into an S x S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and how accurate it thinks the box is that it predicts. Formally we define confidence as $P_r(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: *x, y, w, h,* and *confidence*. The *(x, y)* coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $P_r(\text{Class}_i \mid \text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}}$$

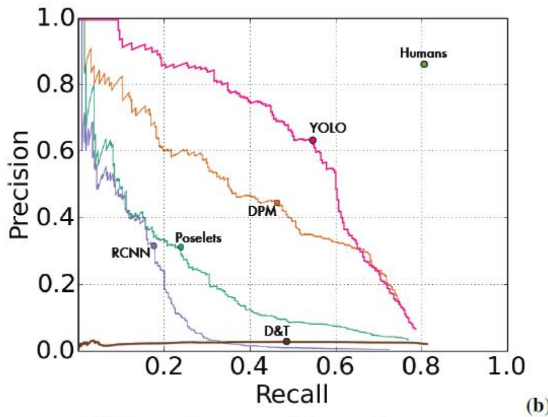
which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.



Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 x 1 convolutional layers reduce the features space from preceding layers.

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. The model struggles with small objects that appear in groups, such as flocks of birds. Since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. It also uses relatively coarse features for predicting bounding boxes since our architecture has multiple down sampling layers from the input image.

Finally, when we train on a loss function that approximates detection performance, the loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. The main source of error is incorrect localizations.



(a) Picasso Dataset precision-recall curves.

	VOC 2007 AP	Picasso AP	Picasso Best F_1	People-Art AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 5: Generalization results on Picasso and People-Art datasets.

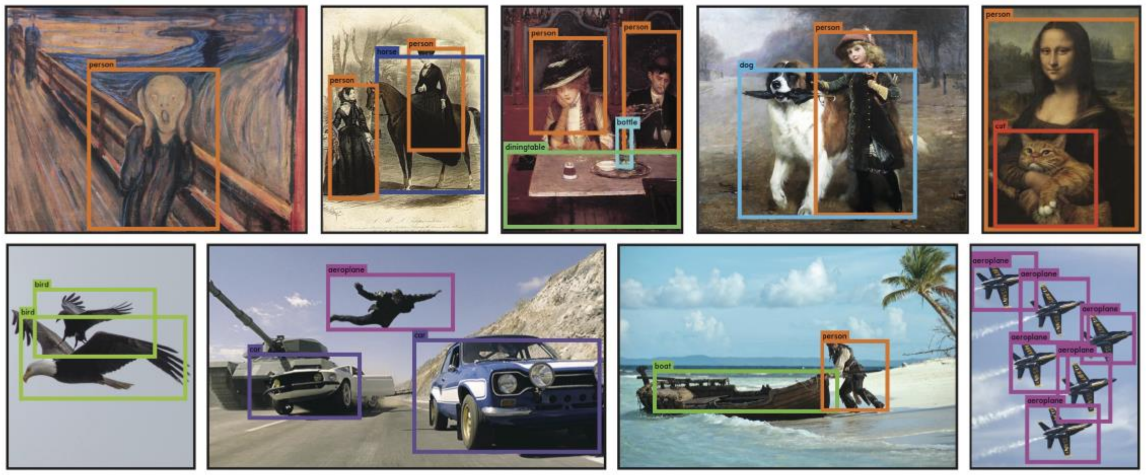


Figure 6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

3.2 YOLOv2 (YOLO9000)

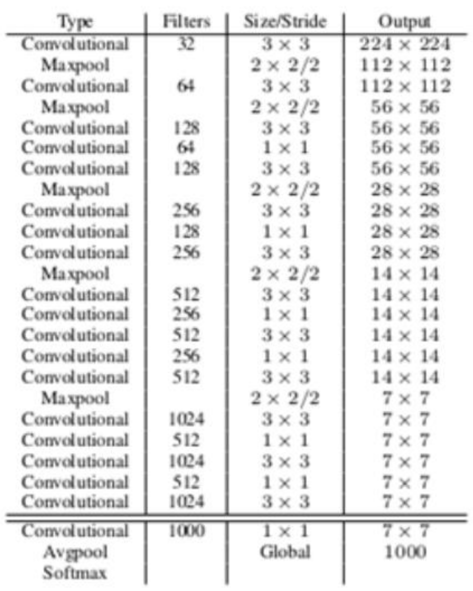
YOLO suffers from a variety of shortcomings relative to state-of-the-art detection systems. Error analysis of YOLO compared to Fast R-CNN shows that YOLO makes a significant number of localization errors. Furthermore, YOLO has relatively low recall compared to region proposal-based methods. Thus, we focus mainly on improving recall and localization while maintaining classification accuracy. Computer vision generally trends towards larger, deeper networks. Better performance often hinges on training larger networks or assembling multiple models together. However, with YOLOv2 we want a more accurate detector that is still fast. Instead of scaling up our network, we simplify the network and then make the representation easier to learn. We pool a variety of ideas from past work with our own novel concepts to improve YOLO's performance.

Batch Normalization - Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization. By adding batch normalization on all the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

High Resolution Classifier - All state-of-the-art detection methods use classifier pre-trained on ImageNet. Starting with AlexNet most classifiers operate on input images smaller than 256 x 256. The original YOLO trains the classifier network at 224 x 224 and increases the resolution to 448 for detection. This means the network must simultaneously switch to learning object detection and adjust the new input resolution. For YOLOv2 we first fine tune the classification network at the full 448_448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high-resolution classification network gives us an increase of almost 4% mAP.

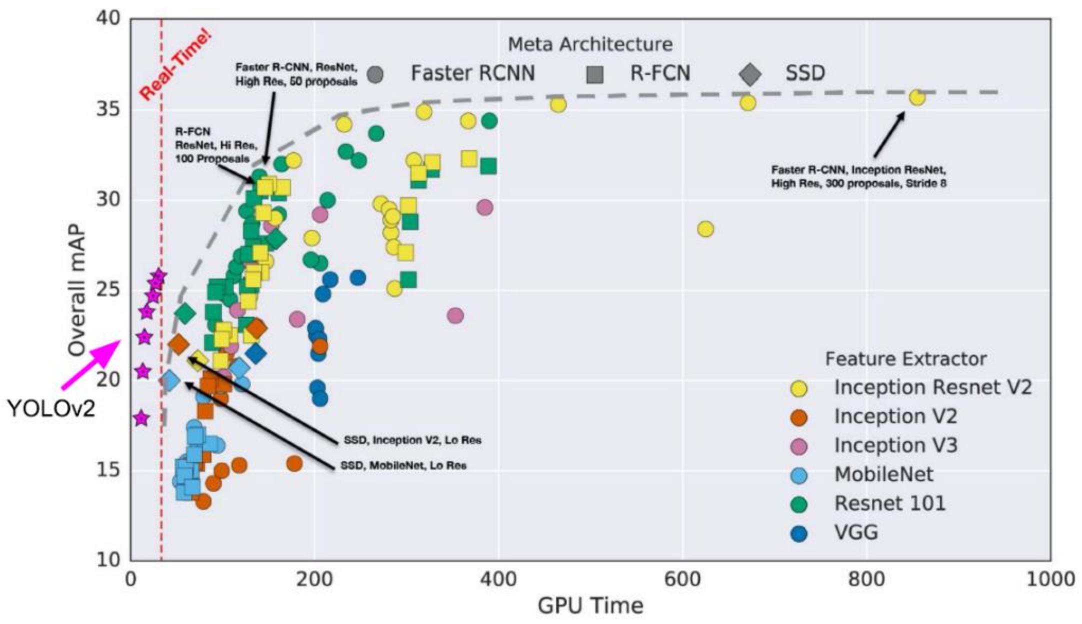
Convolutional with Anchor Boxes -YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Instead of predicting coordinates directly Faster R-CNN predicts bounding boxes using hand-picked priors. Using only convolutional layers the region proposal network (RPN) in Faster R-CNN predicts offsets and confidences for anchor boxes. Since the prediction layer is convolutional, the RPN predicts these offsets at every location in a feature map. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn. We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes.

YOLOv2 is state-of-the-art and faster than other detection systems across a variety of detection datasets. Furthermore, it can be run at a variety of image sizes to provide a smooth tradeoff between speed and accuracy. YOLO9000 is a real-time framework for detection more than 9000 object categories by jointly optimizing detection and classification. We use WordTree to combine data from various sources and our joint optimization technique to train simultaneously on ImageNet and COCO. YOLO9000 is a strong step towards closing the dataset size gap between detection and classification.



Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Layered Architecture of Yolov2

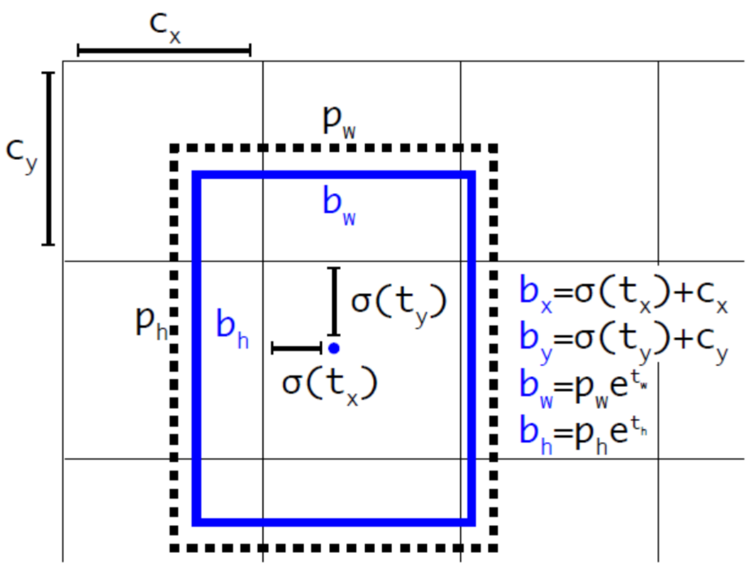


Microsoft COCO Dataset

3.3 YOLOv3

YOLOv3 has few incremental improvements on YOLOv2. For example, a better feature extractor, **DarkNet-53** with shortcut connections as well as a better object detector with **feature map up sampling and concatenation**.

1. **Bounding Box Prediction: -**



Bounding Box Prediction, Predicted Box (Blue), Prior Box (Black Dotted)

- It is the same as YOLOv2.
- **t_x, t_y, t_w, t_h are predicted.**
- During training, sum of squared error loss is used.
- And objectness score is predicted using logistic regression. It is 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. Only one bounding box prior is assigned for each ground truth object.

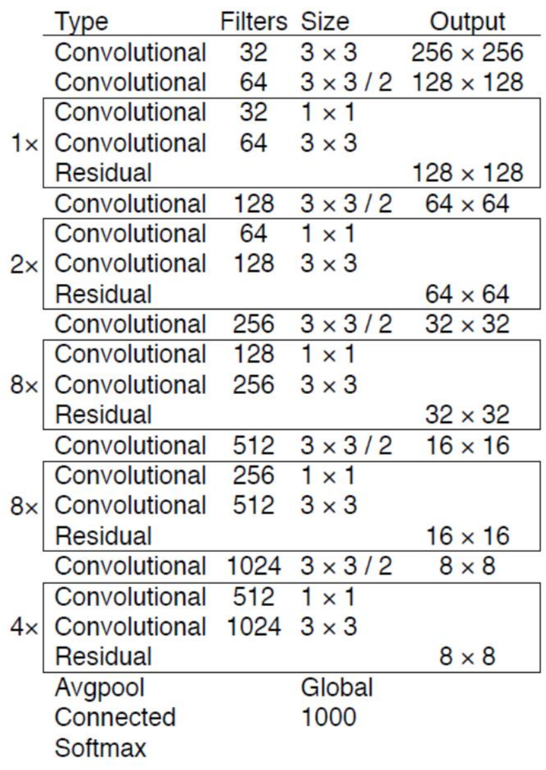
2. Class Prediction: -

SoftMax is not used. Instead, independent logistic classifiers are used, and binary cross-entropy loss is used. Because there may be overlapping labels for multilabel classification such as if the YOLOv3 is moved to other more complex domain such as Open Images Dataset.

3. Prediction Across Scales: -

- 3 different scales are used.
- Features are extracted from these scales like FPN.
- Several convolutional layers are added to the base feature extractor Darknet-53 (which is mentioned in the next section).
- The last of these layers predicts the bounding box, objectness and class predictions.
- On COCO dataset, 3 boxes at each scale. Therefore, the output tensor is N×N×[3×(4+1+80)], i.e. 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.
- Next, the feature map is taken from 2 layers previous and is upsampled by 2×. A feature map is also taken from earlier in the network and merge it with our upsampled features using concatenation. This is the typical encoder-decoder architecture, just like SSD is evolved to DSSD.
- This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map.
- Then, a few more convolutional layers are added to process this combined feature map, and eventually predict a similar tensor, although now twice the size.
- k-means clustering is used here as well to find better bounding box prior. Finally, on COCO dataset, (10×13), (16×30), (33×23), (30×61), (62×45), (59×119), (116×90), (156×198), and (373×326) are used.

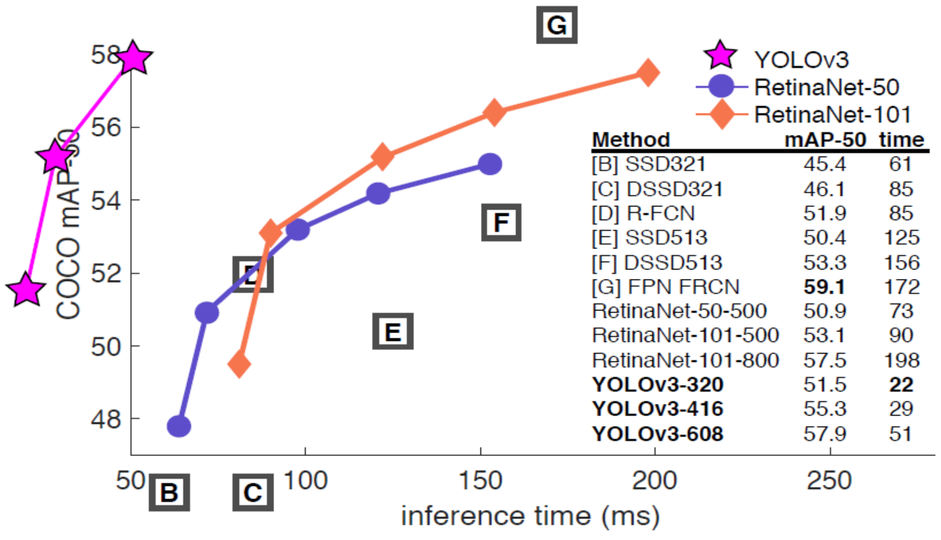
4. Feature Extractor: Darknet – 53



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Darknet-53

- Darknet-19 classification network is used in YOLOv2 for feature extraction.
- Now, in YOLOv3, **a much deeper network Darknet-53** is used, i.e. 53 convolutional layers.
- Both YOLOv2 and YOLOv3 also use Batch Normalization.
- **Shortcut connections** are also used as shown above.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLOv3 is a good detector. It's fast, it's accurate. It's not as great on the COCO average AP between .5 and .95 IOU metric. But it's very good on the old detection metric of .5 IOU. When the duo ran YOLOv3 on Microsoft's COCO Dataset it performed on par with RetinaNet and SSD variants, indicating the model's strength at fitting boxes to objects. However, when the IOU threshold raises the model struggles to align boxes perfectly with objects. Redmon and Farhadi say the model does not work well on average AP between 0.5 and 0.95 IOU metric but performs very well on a threshold metric of 0.5 IOU. It also performs better with small objects than with large objects.

Chapter 4
EVALUATION PARAMETERS

After reading about YOLO and how it works, we need to look at how it is evaluated for its own performance as well as with other methods.

It Intakes an image and divides it in grid of S X S (where S is a natural number). Each pixel in the image can be responsible for a finite number of (5 in our case) bounding box predictions. A pixel is taken responsible for prediction when it is the center of the object detected. Out of all detected boxes, it is taken responsible for the detection of only one object and other detections are rejected.

It predicts **C conditional class probabilities** (one per class for the likeliness of the object class).

*Total detections to be done per image=**S X S((B*5) +C)***

Where

***S X S**= Total number of images yolo divides the Input*

***B*5**=B is the Number of Bounding boxes Detected all over the image(Without any threshold consideration).For each bounding box, 5 elements are detected:*

Detected Objects Centre coordinates(x,y),

Height and Width

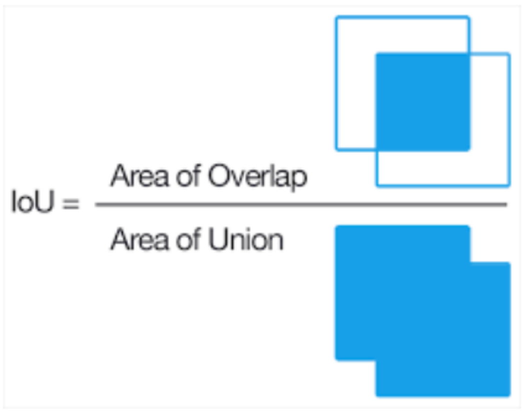
Confidence score.

***C**=Conditional probability for Number of Classes.*

Hence if the image is divided in a 2 x 2 grid, and 10 boxes are predicted with 3 classes(Dog, Cat, Mouse), we will have 2*2(10*5+3) predictions=212 predictions.

- Two thresholds are taken in consideration for selecting final detection

1. **IOU threshold**: The IOU threshold can be better understood using the below image:



Here the two boxes represent Predicted and True object

2. **Confidence threshold**: Threshold for minimum confidence the model has on a detected object (box confidence score)

- Below are certain scores calculated over the detected object.

$$\begin{aligned}\text{box confidence score} &\equiv P_r(object) \cdot IoU \\ \text{conditional class probability} &\equiv P_r(class_i|object) \\ \text{class confidence score} &\equiv P_r(class_i) \cdot IoU \\ &= \text{box confidence score} \times \text{conditional class probability}\end{aligned}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

P(object) is 1 if a box is detected else 0. We must make this clear that IoU isn't the IOU threshold mentioned above but the predicted IoU. Hence the entire process of detection can be summed up as:

- Dividing the image in grid of S X S
- Detecting bounding box
- Calculating various scores (mentioned in the above image)

- Depending on the threshold for confidence and IoU, select some boxes
- **Use Non-Max suppression**: It helps us to avoid duplicate detections of the same object by rejecting multiple predictions for the same object. It takes a list of predicted boxes for the same image and accepts detection with maximum confidence.
- If the detection is over both IOU and Confidence threshold, It is taken as the final prediction

Loss Function: -

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

It has 4 major parts:

- Error in X, Y coordinates of detected object
- Error in Height, width of detected object (root is taken to give less weightage)
- Error in Classification. Lambda has been used to give more weightage to Confidence error.
- Error in confidence of object detected (like log loss)

AP (Average precision): It is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. It sounds complicated but simple as we illustrate it with an example. But before that, we will do a quick recap on precision, recall, and IoU first.

Precision & recall

Precision measures how accurate is your predictions. i.e. the percentage of your predictions are correct.

Recall measures how good you find all the positives. For example, we can find 80% of the possible positive cases in our top K predictions.

Here are their mathematical definitions:

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

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

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

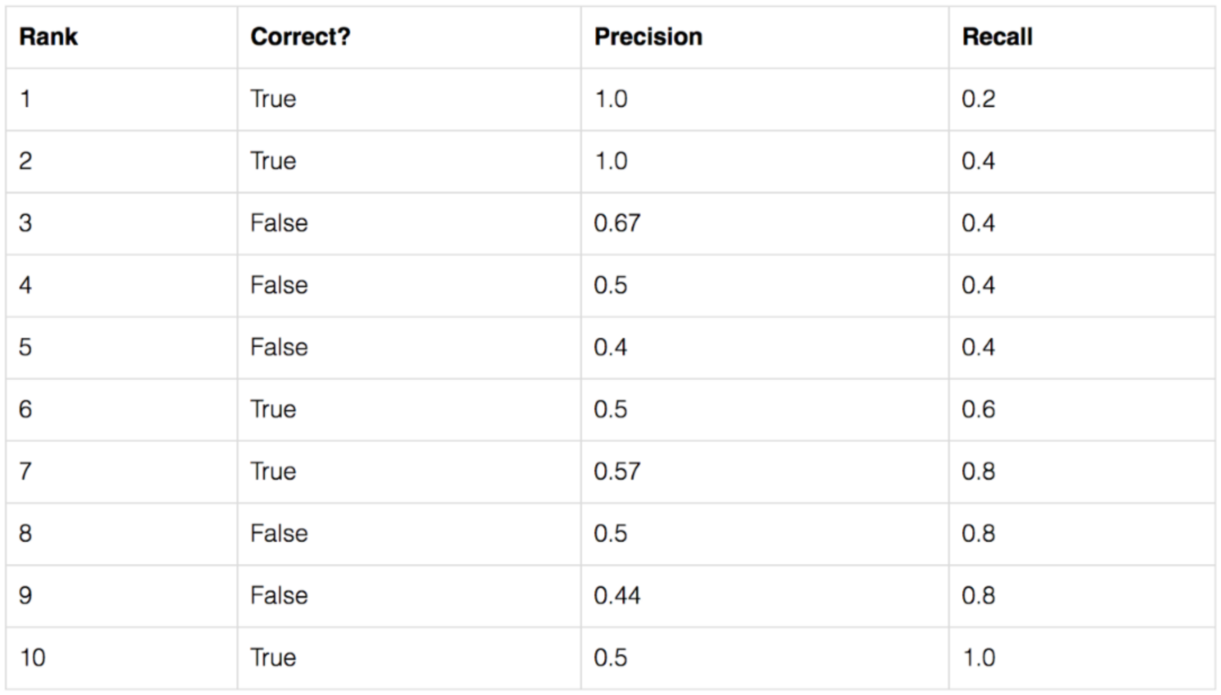
FN = False negative

For example, in the testing for cancer:

$$Precision = \frac{TP}{\text{total positive results}}$$

$$Recall = \frac{TP}{\text{total cancer cases}}$$

Let's create an over-simplified example in demonstrating the calculation of the average precision. In this example, the whole dataset contains 5 apples only. We collect all the predictions made for apples in all the images and rank it in descending order according to the predicted confidence level. The second column indicates whether the prediction is correct or not. In this example, the prediction is correct if IoU ≥ 0.5.



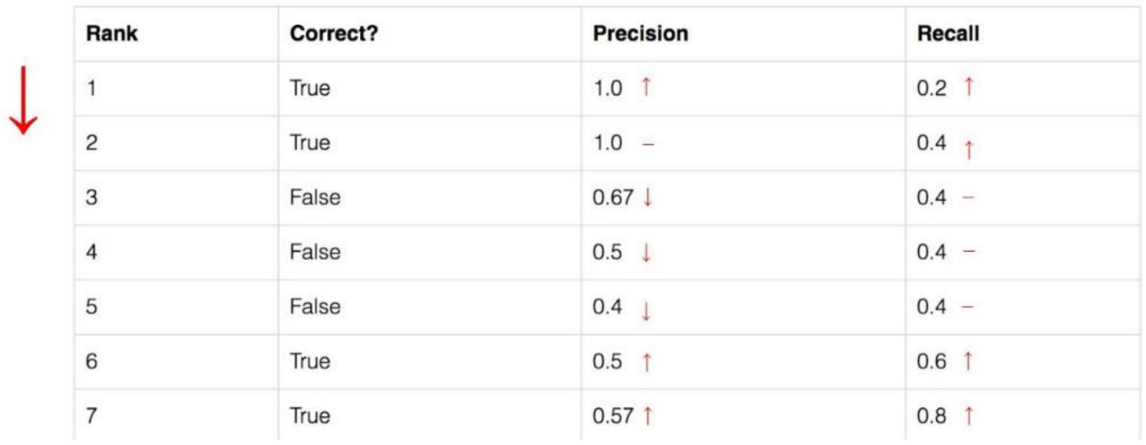
Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Let’s take the row with rank #3 and demonstrate how precision and recall are calculated first.

Precision is the proportion of TP = 2/3 = 0.67.

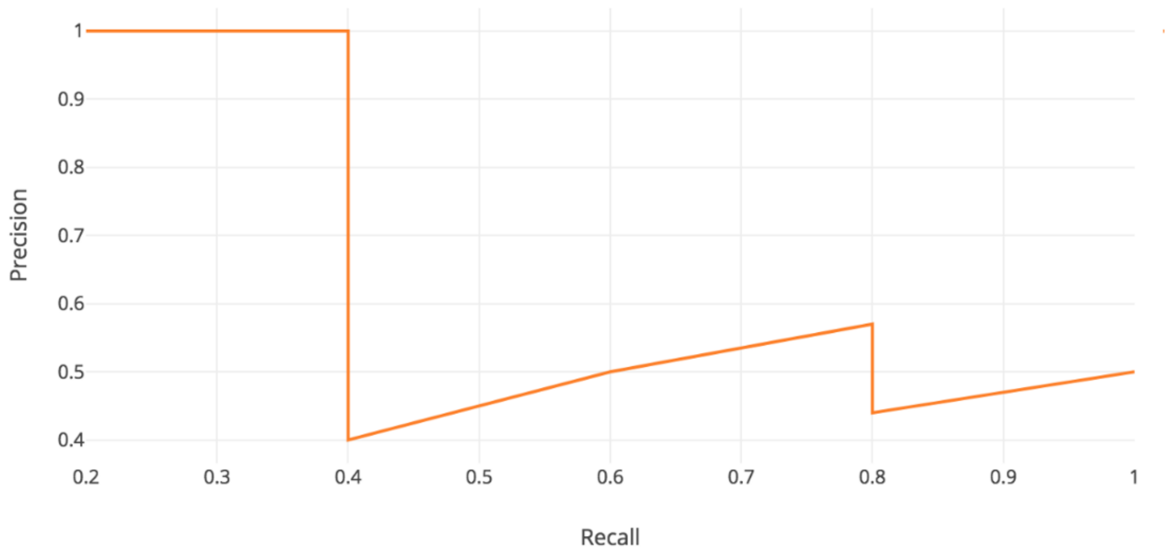
Recall is the proportion of TP out of the possible positives = 2/5 = 0.4.

Recall values increase as we go down the prediction ranking. However, precision has a zigzag pattern — it goes down with false positives and goes up again with true positives.



Rank	Correct?	Precision	Recall
1	True	1.0 ↑	0.2 ↑
2	True	1.0 –	0.4 ↑
3	False	0.67 ↓	0.4 –
4	False	0.5 ↓	0.4 –
5	False	0.4 ↓	0.4 –
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑

Let’s plot the precision against the recall value to see this zig-zag pattern.



Precision-recall curve

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above.

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

Precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1 also. Before calculating AP for the object detection, we often smooth out the zigzag pattern first.

mAP (mean average precision) is the average of AP. In some context, we compute the AP for each class and average them. But in some context, they mean the same thing. For example, under the COCO context, there is no difference between AP and mAP. Here is the direct quote from COCO:

AP is averaged over all categories. Traditionally, this is called "mean average precision" (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from context.

Chapter 5
Methodology

The stated problem can be divided into two sub-problems:

1. **Number Plate Detection.** This problem can be tackled using Object Detection approach where we need to train our model using the car/other vehicle images with number plates.
2. **Extracting text from the detected Number Plate**. This problem can be solved using OCR (Optical Character Recognition) which can be helpful in extracting alphanumeric characters from cropped Number Plate images.

It can be assumed that this solution must be embedded with a web application as well where once the camera captures the image of the vehicle, the backend would call the solution and outputting the contents to the user

Scope

The elements that have been considered in this solution:

1)Number Plate Detection:

● Both frontal and back plates are considered in the solution

● Due to lack of unavailability of annotated Indian car images with number plates, a mix of Tunisian and Indian Cars is used for training purpose

● Rotated images are considered

● Cases, where cars are in some angle, are considered

● Image of any dimension has been considered

● Considering only one detected object for feeding OCR (like if many plates detected, taking only one) per image.

2)Text Extraction from Detected Number Plate:

● The output of Number Plate Detection solution only(no rotated images though)

● Only cropped number plate fed to OCR and not the entire image

● Text with size 12pts is the best extracted.

Out of Scope:

● No random images (cat, dog etc.) for training purpose of the Detection model

● No other vehicle considered other than cars in training (unavailability of the dataset)

● Rotated images (in case of OCR)

● As Tunisian data has been used all through the problem, OCR is able to understand digits but not characters (Arabic language)

● Bad quality images (blur, incomplete etc.)

● No video data

Assumptions

● GPU availability for training

● Though trained & tested over only Car images, It should be able to detect Number Plates in any sort of vehicle (Bike, Truck, etc)

● Image quality is up to mark

● No limitation over Memory and Latency

● Internet Availability (For OCR purpose)

● Python 3+

● Microsoft Vision API and Google Vision API being used is accessible.

● Videos not considered

HIGH LEVEL APPROACH

As mentioned above, the problem has been divided into 2 parts:

● Number Plate Detection

● OCR over extracted Number Plates extracted

The solution has been approached in the following ways:

● Dataset preparation for training (Training/Validation) of Object Detection model

● Setting up Darknet for YOLOv3 on Google Colaboratory (due to the availability of free GPU)

● Once satisfactory results are obtained after changing certain configurations according to the problem, validated the model

● Now, the test dataset was fed, and the objects were detected, and coordinates stored in a JSON file (output of YOLOv3)

● Using the JSON file, extracted the coordinates and cropped the objects accordingly

● Fed the cropped Number Plates to Microsoft Vision API and Google Cloud Vision API which in turn performs OCR over the image.

The problem has been split into two components:

- Object Detection using Yolov3 – Train our model and crop out license plates. This problem can be tackled using Object Detection approach where we need to train our model using the car/other vehicle images with number plates.

- Character Recognition using Google OCR vs Microsoft OCR on the cropped plates. This problem can be solved using OCR (Optical Character Recognition) which can be helpful in extracting alphanumeric characters from cropped Number Plate images.

Technologies Used – Google Colab (GPU), Neural Network Framework (Darknet), OpenCV, Microsoft Azure Cognitive Services, Google Cloud OCR, Pillow library,

Step 1 – Setting up environment: The first step is ready and install all the necessary libraries and pre-requisite applications and datasets to implement this project. We need to have Google Colab setup since we will we need the GPU for training our model. We need to access the darknet repository on GitHub since it will be our neural network framework.

https://github.com/AlexeyAB/darknet

We need to copy and clone it in our google drive so that we can access it through Google Colab.

Step 2 – Preparation of Data Sets: The next step is to prepare our dataset for training our model. Before we start training our model of YoloV3, we need to prepare and annotate our dataset according to standards of the algorithm. Our training dataset has 700 Images which are a mix of Indian and Tunisian number plates. Our testing data set has 140 images that we will test on. Now let us explain the format of the training dataset.

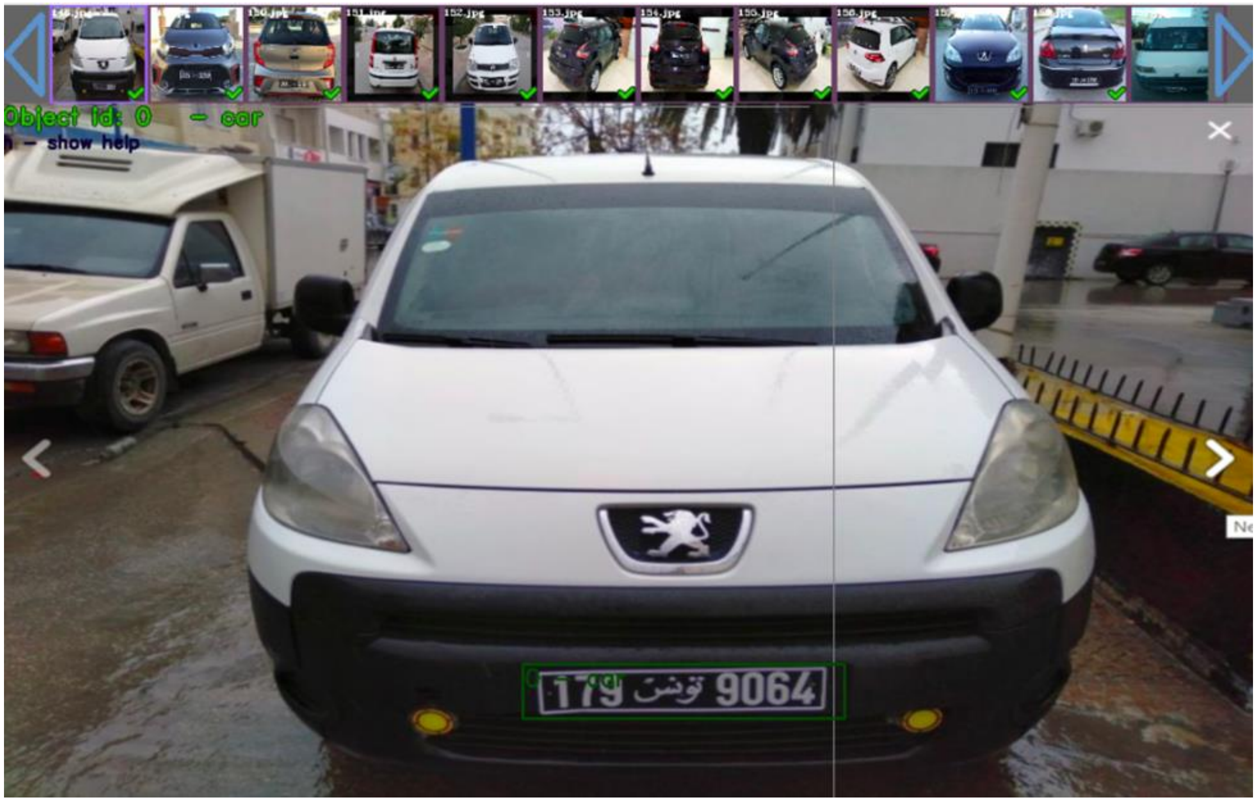
The training dataset consists of 700 images which are labelled as "0.jpg" till "707.jpg". Now these images need to have text files(.txt) of the same name (0.txt) corresponding to their coordinates.

"Label_ID X_CENTER Y_CENTER WIDTH HEIGHT"

Label ID is the numeric ID given to different classes that has to be determined starting from 0 i.e. if 3 classes has to be determined, cats, dogs and monkeys, IDs can be 0,1,2. In our case, the label Id will always be zero since we are only detecting license plates. X_CENTER is the X coordinate of the center of the object that has to be detected/Image_width. Y_CENTER is the Y coordinate of the center of the object that must be detected/Image_height. WIDTH is the width of the object that has to be detected/Image_width. HEIGHT is the height of the object that has to be detected/Image_height. All the values will range from 0-1. Now, to produce such labels/text files, the best tool present is YOLO_Mark from the developers of YOLO itself supporting such an output once the user annotates the image. Now there are two files which will contain information about out dataset.

Obj.data - This holds information about Number_of_classes, Train & valid data location(relative to darknet directory) and backup folder location(where weights after training on yolo are saved).You might not need to create a train.txt as Yolo_Mark creates it for you on its own.

Obj.names - This includes the label names, each label entered in a new line. Remember that the IDs assigned are dependent on the order of label names mentioned in obj.names.



Annotate the images according to your requirement.

So now we have an image dataset with its labels in text files. The folder name is ‘img’ and it will be our training dataset.

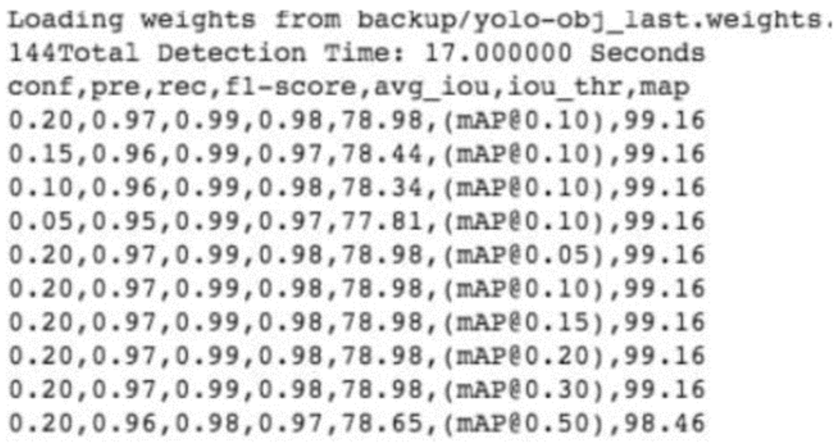
So now that we have our dataset ready, we will start training our model in Google Colab in the next step by finalizing and moving all the necessary files to their respective locations on our Google Drive. Every will till now will be moved to our cloud on Google Drive.

Step 3- Training: The next crucial step is to go to Google Colab and open a new notebook

```
! pip install pydrive
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
local_download_path = os.path.expanduser('~/data')
try:
  os.makedirs(local_download_path)
except: pass
from google.colab import drive
drive.mount('/content/drive/',force_remount=True)

%cd 'drive/My Drive'

!apt-get update > /dev/null
!apt-get upgrade > /dev/null
!apt-get install build-essential > /dev/null
!apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev
libswscale-dev > /dev/null
!apt-get install libopencv-dev > /dev/null
!apt-get install libavcodec-dev libavformat-dev libswscale-d > /dev/null

%cd darknet

!sed -i 's/OPENCV=1/OPENCV=0/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/g' Makefile

!make
!apt install g++-5
!apt install gcc-5
!apt update
!apt upgrade

!./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74
```

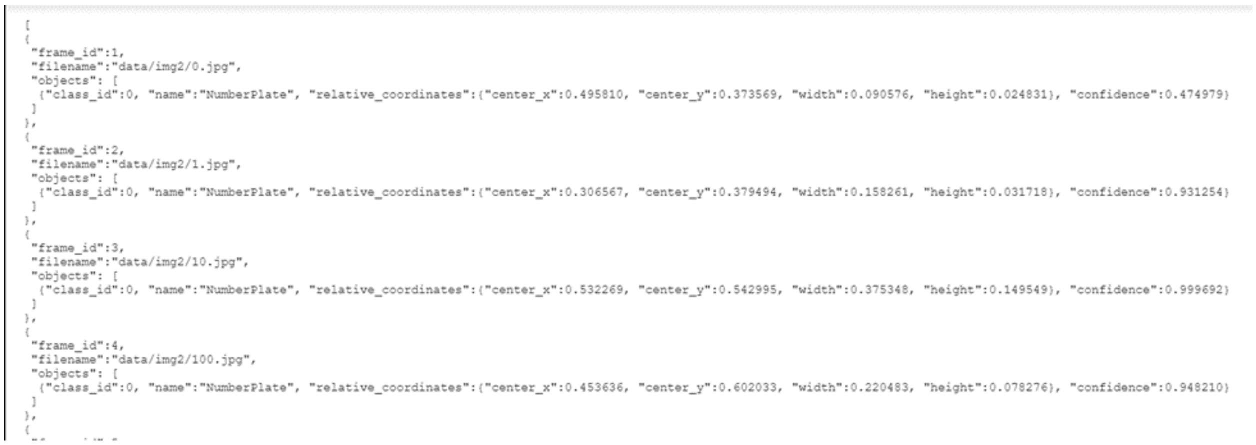
specifically, for training. We mount our google drive in Google Colab by using the commands as you can see above. We install necessary documents and start training our model by using the last command as you can see above. It takes around 12-14 hours for our model to train after which our weights our stored in the backup/ folder in darknet/ as 'yolo-obj_last_weights'. So the result of training our images is the yolo-obj_last_weights file in backup folder that we will use for testing in the next step.

The columns in the below table represents Confidence, Precision, Recall, F1-score, AVG-IOU, Threshold, mAP when validating over same data(confidence & threshold are the parameters to be analyzed) -It can be observed that best results are observed by setting confidence:0.2 & any threshold. Tested images were taken on a confidence=0.2,threshold=0.3



```
Loading weights from backup/yolo-obj_last.weights.
144Total Detection Time: 17.000000 Seconds
conf,pre,rec,f1-score,avg_iou,iou_thr,map
0.20,0.97,0.99,0.98,78.98,(mAP@0.10),99.16
0.15,0.96,0.99,0.97,78.44,(mAP@0.10),99.16
0.10,0.96,0.99,0.98,78.34,(mAP@0.10),99.16
0.05,0.95,0.99,0.97,77.81,(mAP@0.10),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.05),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.10),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.15),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.20),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.30),99.16
0.20,0.96,0.98,0.97,78.65,(mAP@0.50),98.46
```

Step 4 – Testing: After training our model, we will use the weights obtained from the previous step to perform testing on our dataset. The testing dataset consists of 142 images on which we will perform object detection using yolov3. The result of testing is a .json file that stores the coordinates of the license plates of all images. We use these coordinates from the json file to create another dataset which contains the images of the cropped license plates.



```
[
{
 "frame_id":1,
 "filename":"data/img2/0.jpg",
 "objects": [
  {"class_id":0, "name":"NumberPlate", "relative_coordinates":{"center_x":0.495810, "center_y":0.373569, "width":0.090576, "height":0.024831}, "confidence":0.474979}
 ]
},
{
 "frame_id":2,
 "filename":"data/img2/1.jpg",
 "objects": [
  {"class_id":0, "name":"NumberPlate", "relative_coordinates":{"center_x":0.306567, "center_y":0.379494, "width":0.158261, "height":0.031718}, "confidence":0.931254}
 ]
},
{
 "frame_id":3,
 "filename":"data/img2/10.jpg",
 "objects": [
  {"class_id":0, "name":"NumberPlate", "relative_coordinates":{"center_x":0.532269, "center_y":0.542995, "width":0.375348, "height":0.149549}, "confidence":0.999692}
 ]
},
{
 "frame_id":4,
 "filename":"data/img2/100.jpg",
 "objects": [
  {"class_id":0, "name":"NumberPlate", "relative_coordinates":{"center_x":0.453636, "center_y":0.602033, "width":0.220483, "height":0.078276}, "confidence":0.948210}
 ]
},
{
```

As you can see above, the json file has the coordinates of the possible detected license plates. We input these coordinates in program to output the images on which we will eventually perform OCR using a suitable technique. The code for testing is like training expect the training command changes to the testing command.

!./darknet detector test data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_last.weights -ext_output -dont_show -out result.json < data/valid.txt

From the above command we can observe that we use the weights we obtained from the previous step of training our model. The path of our image dataset is given in the 'valid.txt' file which consists of all the images we will performing our object detection on. This second dataset is called a validation dataset.

Step 5 – Cropping license plates: The next step is to use the .json file obtained in the previous step and create a dataset of just the cropped license plates. It can be done using the following code:

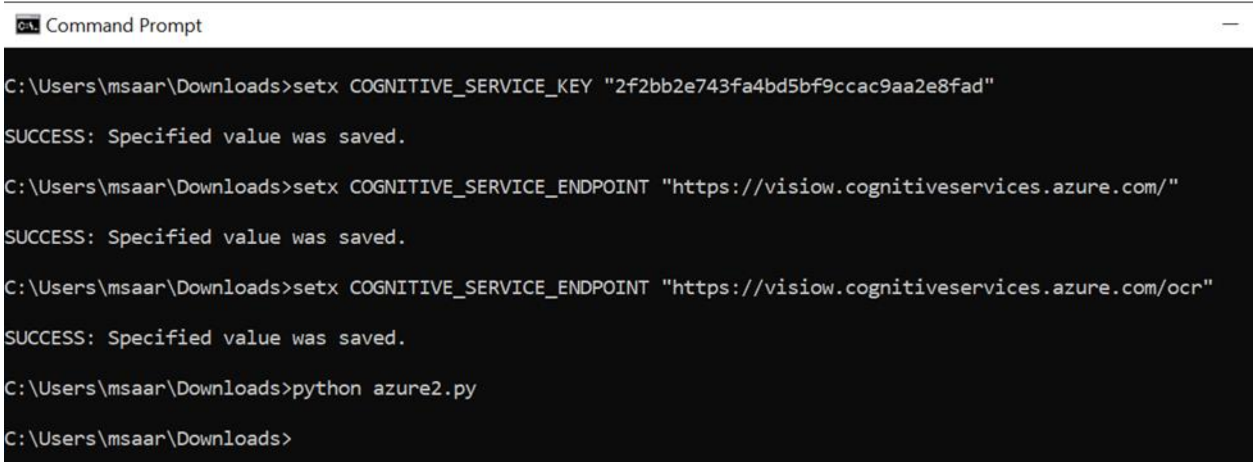
```
import pandas as pd
import json

#loading results of YOLOv3 Detection
with open('result_plates.json') as file:
    data = json.load(file)
df = pd.DataFrame(data)
df

med=[]
def check(x):
    for y in x:
        if y['name']=="NumberPlate":
                    return True
    return False
df['plates']=df['objects'].transform(lambda temp:check(temp))

df=df[df['plates']]

df['filename']=df['filename'].transform(lambda f:f.split('/')[-1])
in_book=list(df['filename'])

%cd img2/
import subprocess
proc=subprocess.Popen('ls', shell=True, stdout=subprocess.PIPE, )
output=proc.communicate()[0]
output=output.decode('utf-8').split('\n')
output=list(output)
p=set(output)-set(in_book)
for x in p:
    %rm $x
med=[]
def check2(x):
    for y in x:
        if y['name']=="NumberPlate":
                    return y
    return False
df['plate_data']=df['objects'].transform(lambda temp:check2(temp))
df=df.drop(['frame_id','objects','plates'],axis=1)
def details(x,flag_2):
        co=x['relative_coordinates']
        wid=float(co['width'])/2.0
        hi=float(co['height'])/2.0
        if flag_2=='y2':
            return float(co['center_y'])+hi
        if flag_2=='x2':
            return float(co['center_x'])+wid
        if flag_2=='x1':
            return float(co['center_x'])-wid
        if flag_2=='y1':
            return float(co['center_y'])-hi
data['x1']=data['plate_data'].transform(lambda f:details(f,'x1'))
data['y1']=data['plate_data'].transform(lambda f:details(f,'y1'))
data['x2']=data['plate_data'].transform(lambda f:details(f,'x2'))
data['y2']=data['plate_data'].transform(lambda f:details(f,'y2'))
from PIL import Image
count=0
for x,y in data.iterrows():
        img = Image.open('/content/drive/My Drive/darknet/img2/'+str(y['filename']))
        dim=img.size
        w,h=dim[0],dim[1]
        x1=y['x1']*w
        x2=y['x2']*w
        y1=y['y1']*h
        y2=y['y2']*h
        area=(x1,y1,x2,y2)
        print(count)
        cropped_img = img.crop(area)
        cropped_img=cropped_img.convert('RGB')
        cropped_img.save("/content/drive/My Drive/darknet/im/"+str(x)+'.jpg')
        count+=1
```

After executing the code above, we will have a dataset that contains images of our cropped license plates. Since this is the part of the whole image, we are interested in we will perform OCR or Optical Character recognition using Google OCR and Microsoft Azure Cognitive Services Vision API.

So far, we have completed one component of our project which is to perform Object detection on images and find out the license plates in our test dataset consisting of 142 images. We have created another dataset on which we will perform OCR or character recognition. All the steps we have performed so far were done on our Google Colab. All the data was being used from our Google Drive to access various folders and files including our neural network framework Darknet. The main motivation to use Google Colab was the free GPU that it provides our training. Since, we have accomplished one component of our project we will be moving to our local machine to perform OCR. Before we do that, we need to download the ‘im’ directory from Google Drive that contains all the cropped license plates on which we will perform character recognition using Google OCR and Microsoft Vision API. The reason we choose them is because they are easily integrable and provide high quality state of the art recognition performances. One of the reasons is also that they offer free trial subscription for a limited period to enable short term testing goals.

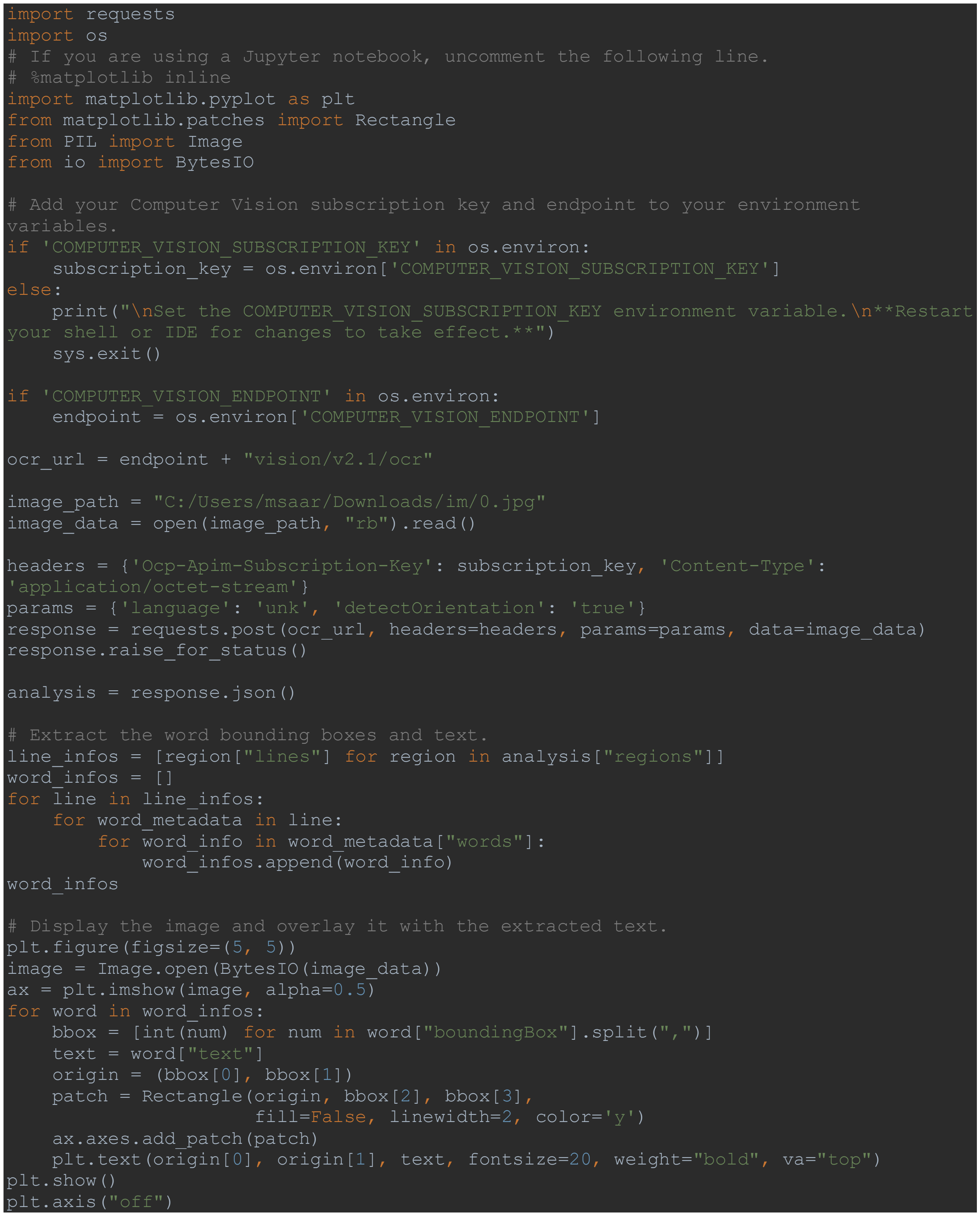
Step 6- OCR (Microsoft Azure OCR): We will use the APIs provided by Microsoft Azure called Computer Vision APIs and call them locally in our code of Python. Before that we need to sign up for the Azure services and get two components that we will use in our code – Subscription key and end points. They act as a certification and a way of authentication for us customers to use the APIs of Azure. After getting hold of these two components we will write code in our local machine in python and call these APIs in our code.

Before we run the code below, we will have to create environment variable for the subscription key and the service end point string named, COMPUTER_VISION SUBSCRIPTION_KEY and COMPUTER_VISION_END_POINT as shown below on the Command prompt of our computer.



```
C:\Users\msaar\Downloads>setx COGNITIVE_SERVICE_KEY "2f2bb2e743fa4bd5bf9ccac9aa2e8fad"

SUCCESS: Specified value was saved.

C:\Users\msaar\Downloads>setx COGNITIVE_SERVICE_ENDPOINT "https://visiow.cognitiveservices.azure.com/"

SUCCESS: Specified value was saved.

C:\Users\msaar\Downloads>setx COGNITIVE_SERVICE_ENDPOINT "https://visiow.cognitiveservices.azure.com/ocr"

SUCCESS: Specified value was saved.

C:\Users\msaar\Downloads>python azure2.py

C:\Users\msaar\Downloads>
```

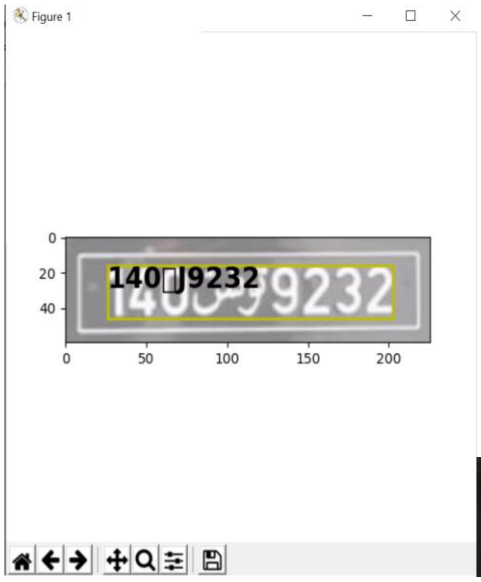
The code was locally run on our machine, but Azure Vision APIs were called from our Python File ‘azure2.py’ as shown be



```
import requests
import os
# If you are using a Jupyter notebook, uncomment the following line.
# %matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO

# Add your Computer Vision subscription key and endpoint to your environment
variables.
if 'COMPUTER_VISION_SUBSCRIPTION_KEY' in os.environ:
    subscription_key = os.environ['COMPUTER_VISION_SUBSCRIPTION_KEY']
else:
    print("\nSet the COMPUTER_VISION_SUBSCRIPTION_KEY environment variable.\n**Restart
your shell or IDE for changes to take effect.**")
    sys.exit()

if 'COMPUTER_VISION_ENDPOINT' in os.environ:
    endpoint = os.environ['COMPUTER_VISION_ENDPOINT']

ocr_url = endpoint + "vision/v2.1/ocr"

image_path = "C:/Users/msaar/Downloads/im/0.jpg"
image_data = open(image_path, "rb").read()

headers = {'Ocp-Apim-Subscription-Key': subscription_key, 'Content-Type':
'application/octet-stream'}
params = {'language': 'unk', 'detectOrientation': 'true'}
response = requests.post(ocr_url, headers=headers, params=params, data=image_data)
response.raise_for_status()

analysis = response.json()

# Extract the word bounding boxes and text.
line_infos = [region["lines"] for region in analysis["regions"]]
word_infos = []
for line in line_infos:
    for word_metadata in line:
        for word_info in word_metadata["words"]:
            word_infos.append(word_info)
word_infos

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(5, 5))
image = Image.open(BytesIO(image_data))
ax = plt.imshow(image, alpha=0.5)
for word in word_infos:
    bbox = [int(num) for num in word["boundingBox"].split(",")]
    text = word["text"]
    origin = (bbox[0], bbox[1])
    patch = Rectangle(origin, bbox[2], bbox[3],
                      fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(origin[0], origin[1], text, fontsize=20, weight="bold", va="top")
plt.show()
plt.axis("off")
```

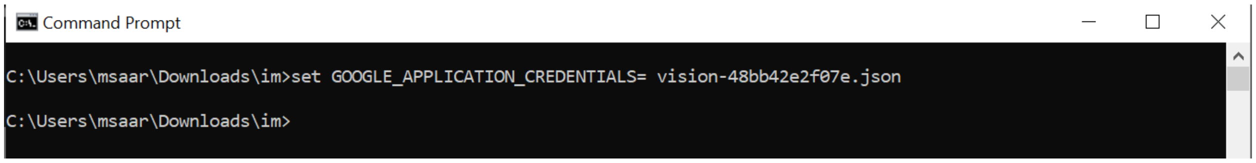
Now we can just change the file name and choose whatever image we want to perform OCR on. In our case, we perform OCR on the cropped license plates and observe the results.

For example, we had ‘9.jpg’ in our folder that we perform OCR on using Azure Vision APIs and got the following result.



Hence, we test this for all images in the 'im' directory with cropped license plates and note down the results. This is how we perform OCR using Microsoft Azure Vision API.

Step 7 – (Google OCR): We also used Microsoft Azure's competitor Google to use its Vision API service to observe the differences in performance in both by using our cropped license plate image dataset and reading studies of their comparison on some more vast and bigger benchmark datasets. As previously, we need to set up our free account in Google Cloud to use the Cloud Vision API provided by Google to perform OCR. We also need to obtain Subscription keys and End points for authentication and certification for when we finally use the APIs. We have to add credentials to our project and then download a .JSON key that we will use for authentication. We also need to enable Cloud Vision API for that project that we are working on in our Google Cloud. The contents of the .Json file contain information such as keys, endpoints necessary for authentication. The file is downloaded and put into the working directory where our cropped license plates are present. So, after we get the authentication key, we will run load the .json file into our environmental variable. Now, before we run our code to perform OCR on those cropped images we must authenticate and give adequate information to our OS that we can use the Google Cloud Vision APIs. To do that, we add the downloaded .json file which contains important information such as keys and end points which help us authenticate Google Cloud APIs.

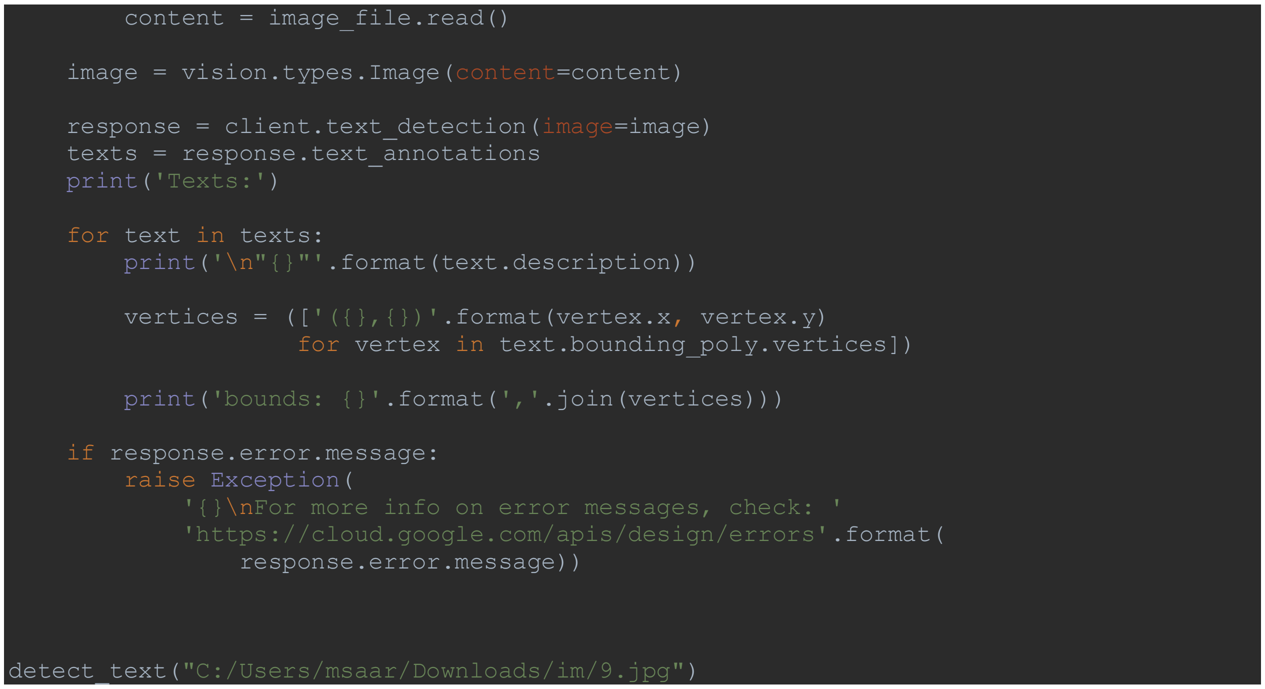


```
C:\Users\msaar\Downloads\im>set GOOGLE_APPLICATION_CREDENTIALS= vision-48bb42e2f07e.json

C:\Users\msaar\Downloads\im>
```

After we have successfully imported the key into our system, we will move into the directory 'im' where our images are ready for OCR. After that, we run our code and get the desired results.

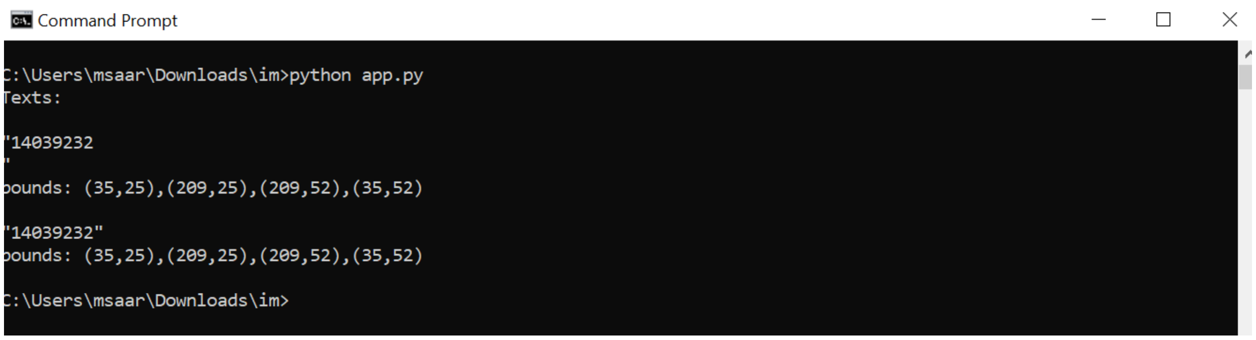
The code was locally run on our machine but Google Cloud Vision APIs were called using our Python file 'app.py' as shown below: -

```
def detect_text(path):
    """Detects text in the file."""
    from google.cloud import vision
    import io
    client = vision.ImageAnnotatorClient()

    with io.open(path, 'rb') as image_file:
```



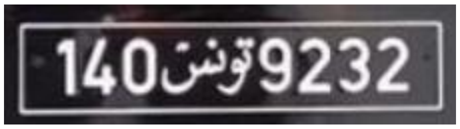
```
        content = image_file.read()

    image = vision.types.Image(content=content)

    response = client.text_detection(image=image)
    texts = response.text_annotations
    print('Texts:')

    for text in texts:
        print('\n"{}"'.format(text.description))

        vertices = (['({},{})'.format(vertex.x, vertex.y)
                    for vertex in text.bounding_poly.vertices])

        print('bounds: {}'.format(','.join(vertices)))

    if response.error.message:
        raise Exception(
            '{}\nFor more info on error messages, check: '
            'https://cloud.google.com/apis/design/errors'.format(
                response.error.message))



detect_text("C:/Users/msaar/Downloads/im/9.jpg")
```

We have used the same image as the one we used when we used Azure API for OCR and we get the following result.



Command Prompt

```
C:\Users\msaar\Downloads\im>python app.py
Texts:

"14039232
"
bounds: (35,25),(209,25),(209,52),(35,52)

"14039232"
bounds: (35,25),(209,25),(209,52),(35,52)

C:\Users\msaar\Downloads\im>
```



As we can see the image above is the same one as before and get interesting results. Hence, we complete testing all the images in our directory by using the Google OCR APIs and observe the results and compare them with the ones we got from using Azure APIs.

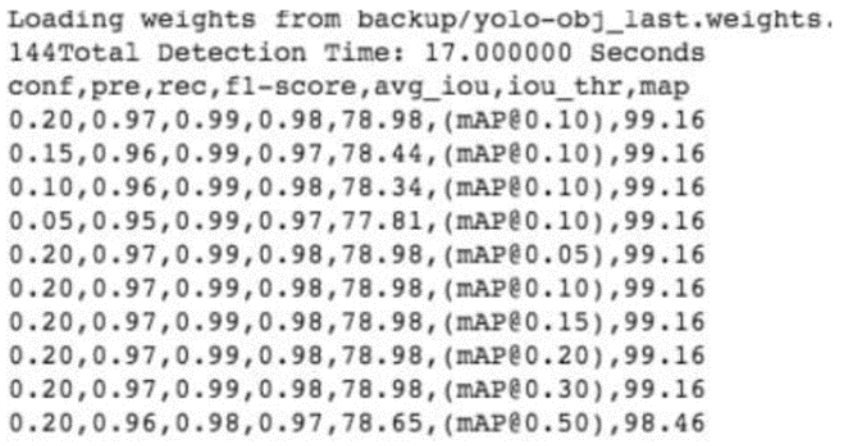
Chapter 6

Experimental Results

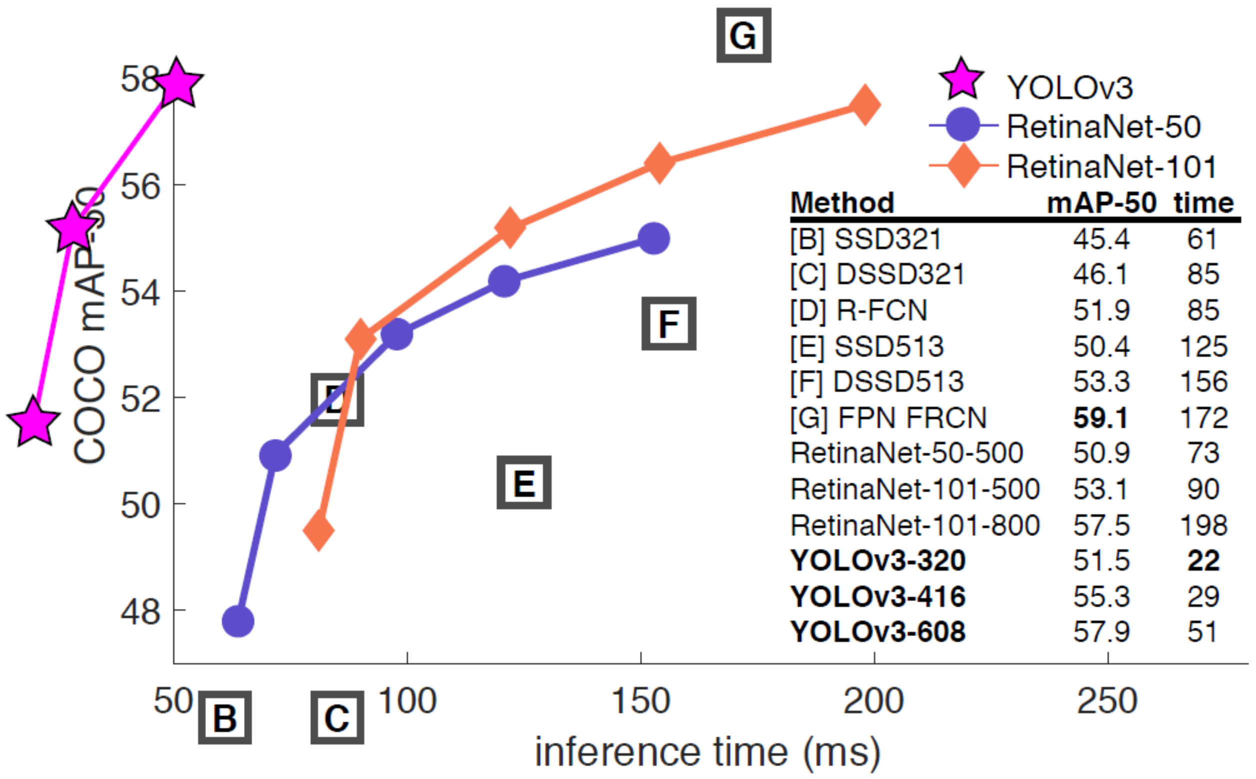
$$\textbf{Final Accuracy} = \frac{\textbf{Characters successfully recognized}}{\textbf{Total characters to recognize}} * \textbf{100}$$

- Yolov3 displayed the highest performance among all object detection algorithms so far with a *mAP* of 99.16%.
- In comparison to other object detection algorithms, Yolov3 displays higher accuracy, lowest inference or processing time.
- Accuracy of **Microsoft Vision API – 92%**
- Accuracy of **Google Cloud Vision API – 95%**
- The performance of OCR applications is very competitive, and it also depends on various factors such as angle, clarity of the image or video.

The columns in the below table represents Confidence, Precision, Recall, F1-score, AVG-IOU, Threshold, mAP when validating over same data(confidence & threshold are the parameters to be analyzed) -It can be observed that best results are observed by setting confidence:0.2 & any threshold. Tested images were taken on a confidence=0.2, threshold=0.3



```
Loading weights from backup/yolo-obj_last.weights.
144Total Detection Time: 17.000000 Seconds
conf,pre,rec,f1-score,avg_iou,iou_thr,map
0.20,0.97,0.99,0.98,78.98,(mAP@0.10),99.16
0.15,0.96,0.99,0.97,78.44,(mAP@0.10),99.16
0.10,0.96,0.99,0.98,78.34,(mAP@0.10),99.16
0.05,0.95,0.99,0.97,77.81,(mAP@0.10),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.05),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.10),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.15),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.20),99.16
0.20,0.97,0.99,0.98,78.98,(mAP@0.30),99.16
0.20,0.96,0.98,0.97,78.65,(mAP@0.50),98.46
```



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

To give you an idea on how we came up with the accuracy for Google OCR as well as Microsoft Azure OCR, we will show you how we used the formula above to come up with those accuracies.

As we discussed in our methodology, we calculated accuracy for each image from our data set and then calculated the accumulated accuracy which is the final accuracy for that OCR. Below, we will show you the calculations for each OCR:

MICROSOFT AZURE OCR

Microsoft Azure is another cloud computing service. Azure provides several computer vision services. They are wrapped as different APIs — Computer Vision API for general purpose CV tasks, Face API for face detection and recognition, Content Moderator for filtering, and some more which are yet in Preview status.

The Computer Vision API allows classifying the image content by providing a comprehensive list of tags and attempting to build a natural language description of the scene. Also, the API can recognize the celebrities and landmarks.

Another feature is Optical Character Recognition (OCR) of printed text and as a preview. The OCR for the handwritten texts is also available, yet only for the English language.

The Face API is used to detect the faces in the images and retrieve the bounding rectangles and facial features like emotional state, gender, age, facial hair, smile score and facial landmarks. One more feature is Face Recognition which helps to understand who the person is matching against a database. This feature may be useful for security. Another one is like face finding, which finds a list of faces which look like the input face.

The Content Moderator could be used for video and image filtering. The unwanted content is filtered using machine learning based classifiers and optical character recognition.

The Video Indexer and Custom Vision Service are yet available as a preview. The Video Indexer is used for insight extraction from the videos. It is capable of sentiment analysis, keyword and metadata extraction, and people detection. The Custom Vision Service allows creating fine-tuned computer vision models for a specific use case. This service is capable of incremental learning — your model will improve over time with each image supplied.

Total number of images in our dataset: - 140 images
Total number of characters recognized: - 984 characters
Total number of characters successfully recognized: - 905 characters

TOTAL ACCURACY: - 905/984 * 100 = 91.9%

GOOGLE VISION OCR

Google is one of the most renowned companies in the fields of AI and machine learning. It provides lots of cloud computing services as APIs for computer vision. The Vision API helps your application to understand what is in the image, classifying the content into the known categories and providing the labels.

It is also capable of detecting landmarks — e.g. buildings, monuments, natural structures, or logos, performing character recognition that supports a wide variety of languages. The facial detection allows detecting a face with the person's emotion and headwear. Unfortunately, there is no support for facial recognition. On top of that, you may use the API to search for the similar images on the web and to filter explicit or violent content.

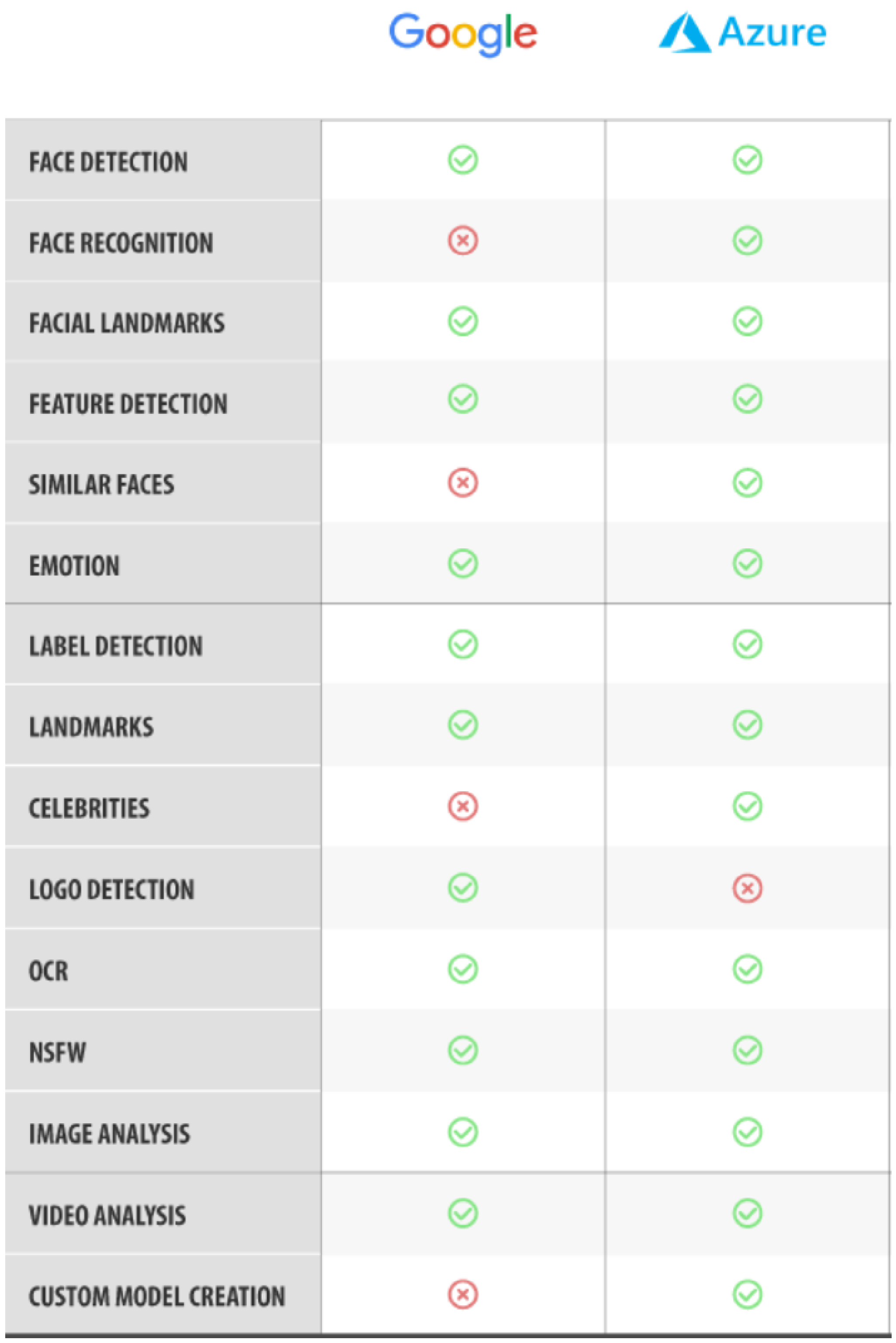
Google also assures the Video Intelligence to perform video analysis, classification, and labeling. This allows searching through the videos based on the extracted metadata. It is also possible to detect the change of the scene and filter the explicit content.

All these features are available through a REST API for easy integration.

Total number of images in our dataset: - 140 images
Total number of characters recognized: - 984 characters
Total number of characters successfully recognized: - 934 characters

TOTAL ACCURACY: - 934/984 * 100 = 94.9%

So, from the above calculations, we can conclude that for our dataset the accuracy was better for Google Vision OCR than Microsoft Azure OCR. The reason for its better performance can be analyzed by either two ways, analyzing the inner-working and design of the character recognition algorithms or testing and doing another comparison between Google OCR and Microsoft OCR to further prove our argument that Google OCR has a better algorithm for character recognition. Now, neither Google or Microsoft provide how their algorithm works and what deep learning algorithm they use to train their model. The primary reason for that is because just like their products Google and Microsoft have many algorithms that have a competitive market to them. Hence, by obvious reasons they don't want to reveal the design and make it open-source. So, the other option left is to analyze and use these OCR applications against various other data sets and see what results we get from that. We can find studies on this already done to validate our argument.



	Google	Azure
FACE DETECTION	✓	✓
FACE RECOGNITION	✗	✓
FACIAL LANDMARKS	✓	✓
FEATURE DETECTION	✓	✓
SIMILAR FACES	✗	✓
EMOTION	✓	✓
LABEL DETECTION	✓	✓
LANDMARKS	✓	✓
CELEBRITIES	✗	✓
LOGO DETECTION	✓	✗
OCR	✓	✓
NSFW	✓	✓
IMAGE ANALYSIS	✓	✓
VIDEO ANALYSIS	✓	✓
CUSTOM MODEL CREATION	✗	✓

Functionality Comparison

According to this study, Google is a notch better than Azure OCR as well as Amazon's AWS OCR library. They used an API integrator to compare them. While all three do a good job when it comes to default text detection they used the Cognitive API Integrator to **compare the responses of these 3 major cognitive API providers** on 3 parameters for the English language:-

- Different orientation
- Different fonts
- Reverse order text

The brief summary of their study is as follows:

- Google does a great job at detecting vertical text irrespective of the top down or bottom up orientation
- Google and Azure both give reverse order text (upside down text) a good shot whereas AWS is never able to decipher it.
- AWS does a great job detecting texts written in different fonts.
- Azure needs handwritten mode on in order to detect different fonts.

Chapter 7

Conclusions

- YOLO v3 has more layers (106 layers) as compared to YOLOv2(30 layers) & YOLOv1(26 layers)
- YOLOv3 predicts smaller objects better (3 scaled versions of the same image passed) as compared to other versions.
- 9 anchor boxes taken than YOLOv2(3 anchors) and YOLOv1(No anchors) helping in better detection.
- Improved loss function for YOLOv3.
- Multilabel detections as compared to multiclass (YOLOv2 & YOLOv1).
- The performance of OCR applications is very competitive, and it also depends on various factors such as angle, clarity of the image or video. Hence, any modern OCR application will perform up to a similar level.
- Google OCR performs better than Microsoft Azure OCR cloud APIs for our dataset.
- Although Google OCR performs better in most scenarios than other OCR APIs, the performance of an OCR Api depends on more than one factor. Every customer has different opinion about 'best' technology. Lots of them think free tool is one of the best. The following characteristics should be used to determine if the OCR is best or not: -
    - **Multi-language supported**
    - **Accuracy**
    - **Scalability**
    - **Bulk image uploading**
    - **Reasonable Price**
    - **And reliable service.**

Chapter 8

REFERNCES

[1]. C. Lin and Y. Li, "A License Plate Recognition System for Severe Tilt Angles Using Mask R-CNN," 2019 International Conference on Advanced Mechatronic Systems (ICAMechS), Kusatsu, Shiga, Japan, 2019, pp. 229-234.

[2]. https://towardsdatascience.com/object-detection-simplified-e07aa3830954

[3]. https://machinelearningmastery.com/object-recognition-with-deep-learning/

[4]. Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[5]. Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[6]. Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).

[7]. https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6

[8]. https://syncedreview.com/2018/03/27/the-yolov3-object-detection-network-is-fast/

[9]. https://medium.com/datadriveninvestor/all-about-yolo-object-detection-and-its-3-versions-paper-summary-and-codes-2742d24f56e

[10] H. Rajput, T. Som and S. Kar, "An Automated Vehicle License Plate Recognition System," in Computer, vol. 48, no. 8, pp. 56-61, Aug. 2015.

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7185290&isnumber=7185273

[11] H. Seibel, S. Goldenstein and A. Rocha, "Eyes on the Target: Super-Resolution and License-Plate Recognition in Low-Quality Surveillance Videos," in IEEE Access, vol. 5, pp. 20020-20035, 2017.

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8016340&isnumber=7859429