Department of Computer Science and Engineering

Indian Institute of Information Technology, KOTA

2015-2019

A Project Report

on

MyVision : Vision for Visually Impaired People

Submitted by :
Puneet Saluja (2015KUCP1019)
Tanmay Sonkusle (2015KUCP1023)
Krishna Sharma (2015KUCP1040)

Supervisor:
Dr. Isha Pathak Tripathi

December 4, 2018

Declaration

We hereby declare that the project work entitled "**MyVision : Vision for Visually Impaired People**" submitted at Indian Institute of Information Technology, Kota, is an authentic record of our work carried out under the supervision of **Dr. Isha Pathak Tripathi**. This project work is submitted in partial fulfillment of the requirements for the award of the degree of Bachelors of Technology in Computer Science and Engineering. The results embodied in this thesis have not been submitted to any other university or institute for the award of any other degree.

Puneet Saluja
(2015KUCP1019)

Tanmay Sonkusle
(2015KUCP1023)

Krishna Sharma
(2015KUCP1040)

Department of Computer Science and Engineering
Indian Institute of Information Technology, Kota
Jaipur, 302017
India

Certificate

This is to certify that the project work entitled "**MyVision : Vision for Visually Impaired People**" which is being submitted by Puneet Saluja, Tanmay Sonkusle and Krishna Sharma in fulfillment for the award of degree of B.Tech in Computer Science and Engineering by the Indian Institute of Information Technology, Kota, is the record of candidates own work carried by them under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of any other degree.

Dr. Isha Pathak Tripathi
Department of Computer Science and Engineering
Indian Institute of Information Technology, Kota
Jaipur, India-302017

Acknowledgements

We are profoundly grateful to **Dr. Isha Pathak Tripathi** for her expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. Her timely and efficient contribution helped us shape our work into its final form and we express our sincerest gratitude for her assistance in any way that we may have asked. We appreciate her guidance in our project that has improved our project many folds, thanks for the comments and advises.

Puneet Saluja
Tanmay Sonkusle
Krishna Sharma

Contents

List of Figures

Abbreviations

CNN : Convolutional Neural Network
R-CNN : Region-based Convolutional Neural Network
SSD : Single Shot Detector
API : Application Programming Interface
SVM : Support Vector Machine
RPN : Region Proposal Network
YOLO : You Look Only Once

Chapter 1

Introduction

1.1 Background

Nowadays smart-phones are playing huge role in our life and the main reason behind this is its diverse functionality and types of services it is providing to us. For a normal person there are many application to interact with like banking, communication, entertainment and many more. But there are many people which are unable to use these services to full extent like people with visual impairment. There are very few application on smart phone that can help these people to perform certain task. The idea behind this project is to create an application that would assist people with visual impairment to analyze their surroundings. This android application will learn contents/objects in the surrounding in real time and will provide voice assistance, about the types of objects nearby. One of the applications and advantages is that the android mobile devices are easily available with everyone but on an advanced level, one can implement this idea to build some gadget that will work in a similar manner. Object detection is the problem of finding and classifying variable number of objects in an image. Task of object detection is very much complex as compared to classification thus to perform this task we have used YOLO (You Only Look Once) algorithm and created a Tensor flow model to use it in an Android Application.

1.2 Motivation and State of Art

Humans learn to recognize and understand their surrounding starting from their birth. Same idea has been utilized by incorporating the intelligence by training into a camera using neural networks and Tensor Flow. This enables to have the same intelligence in cameras, which can be used as an artificial eye and can be used in many areas such as surveillance, detection of objects/things etc.

1.3 Flow Diagram

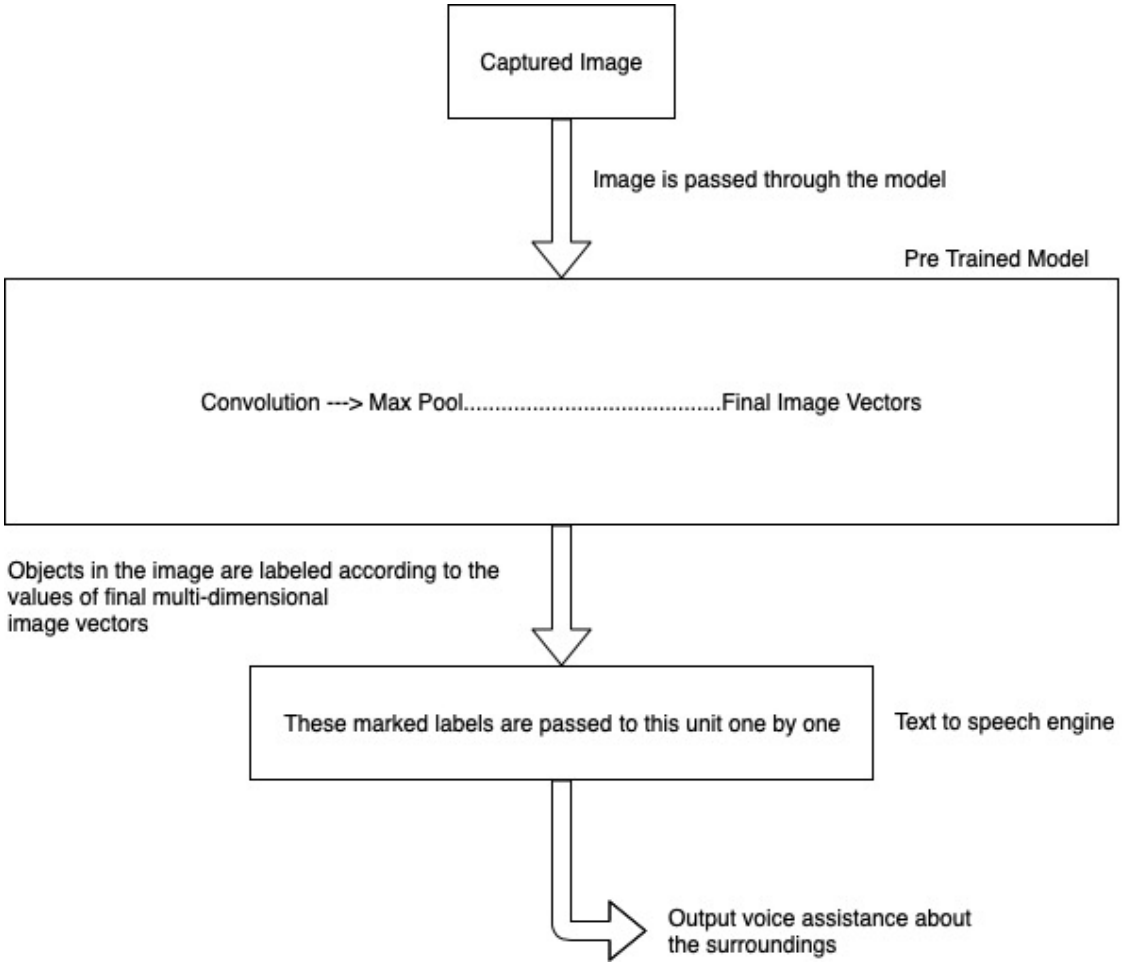

Figure 1.1: Flow Diagram

Android Application uses camera to caputure the images in real time. Then these captured picture will be inputted, to pre-trained Convo-

lutional Neural Network (CNN) and all the detected objects in that picture will get labeled. Those labels will then be passed to Text To Speech engine which by analyzing and processing the text, converts the text into speech.

1.4 Application Overview

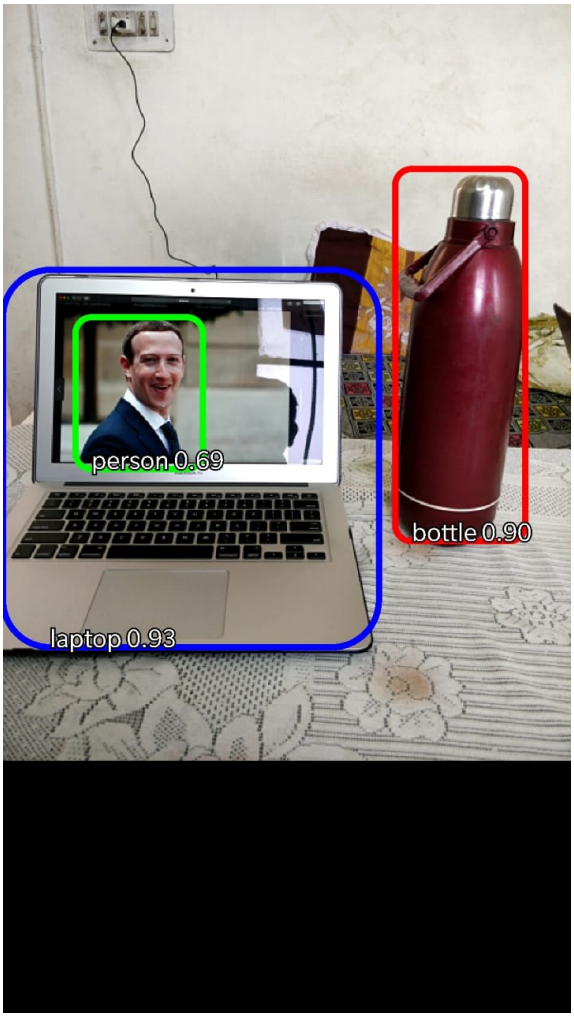

Figure 1.2: Application Preview

We have trained the YOLO model on Pascal VOC image dataset. After the training of the data we got the weights file for the model. The weights file is the converted into to the tensorflow model using the darkflow. Then the tensorflow model is used in the android application to detect the labels in the images and the labels are then passed to the

android text to speech API for voice assistance.

1.5 Challenges

The following challenges were identified while making this project :

- The application should be able to identify multiple objects in a single frame.
- The computational cost of running the model should be less since we have to run this model on a mobile device.
- The application should be able to distinguish between similar objects.
- The application should be able to identify and detect the objects in real time to guide the user.

Chapter 2

Literature Review

2.1 R-CNN

R-CNN[3], or Region-based Convolutional Neural Network, consisted of 3 simple steps:

- Scan the input image for pos sible objects using an algorithm called Selective Search, generating 2000 region proposals
- Run a convolutional neural net (CNN) on top of each of these region proposals
- Take the output of each CNN and feed it into a) an SVM to classify the region and b) a linear regressor to tighten the bounding box of the object, if such an object exists.

So, In this paper au first propose regions, then extract features, and then classify those regions based on their features. In essence, it has turned object detection into an image classification problem. R-CNN was very intuitive, but very slow.

2.2 Fast R-CNN

Fast R-CNN[2] resembled R-CNN in many ways, but improved on its detection speed through two main augmentations:

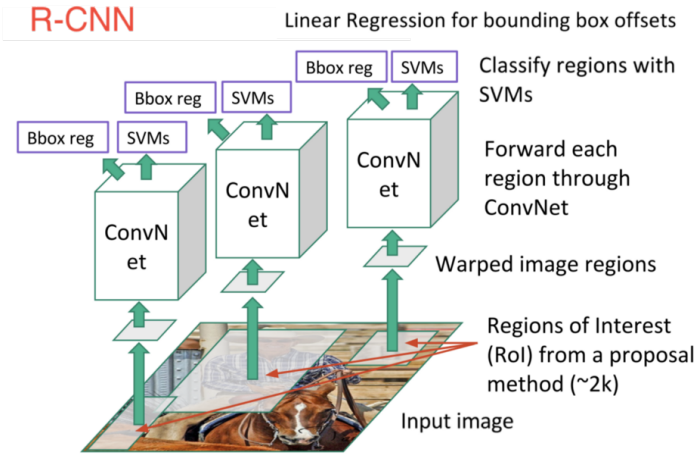

Figure 2.1: R-CNN

- Performing feature extraction over the image before proposing regions, thus only running one CNN over the entire image instead of 2000 CNN's over 2000 overlapping regions
- Replacing the SVM with a softmax layer, thus extending the neural network for predictions instead of creating a new model.

The new model looked something like this:

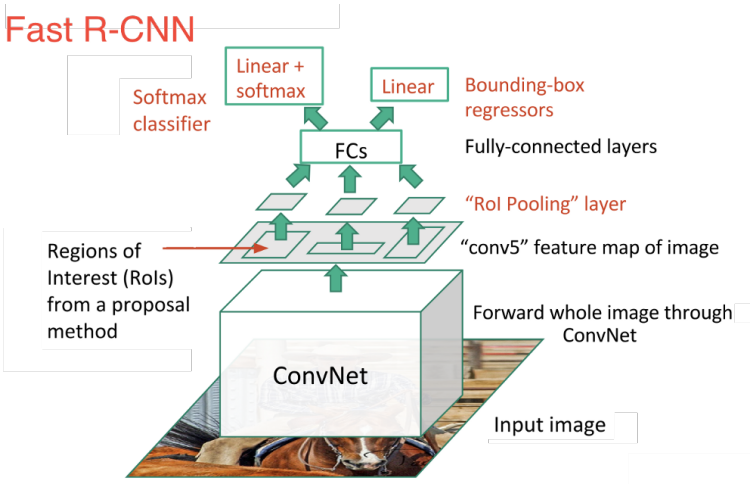

Figure 2.2: Fast R-CNN

As we can see from the image, region proposals are generated based on the last feature map of the network, not from the original image itself. As a result, one CNN can be trained for the entire image.

In addition, instead of training many different SVM's to classify each object class, there is a single softmax layer that outputs the class probabilities directly.

Fast R-CNN performed much better in terms of speed. There was just one big bottleneck remaining: the selective search algorithm for generating region proposals.

2.3 Faster R-CNN

Region proposals detected with the selective search method were still necessary in the previous model, which is computationally expensive. S. Ren and al. (2016) have introduced Region Proposal Network (RPN) to directly generate region proposals, predict bounding boxes and detect objects. The Faster Region-based Convolutional Network (Faster R-CNN)[7] is a combination between the RPN and the Fast R-CNN model.

Here's how the RPN worked:

- At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d).
- For each sliding-window location, it generates multiple possible regions based on k fixed-ratio anchor boxes (default bounding boxes).
- Each region proposal consists of a) an "objectness" score for that region and b) 4 coordinates representing the bounding box of the region.

Once we have our region proposals, we feed them straight into what is essentially a Fast R-CNN. We add a pooling layer, some fully-connected layers, and finally a softmax classification layer and bounding box regressor.

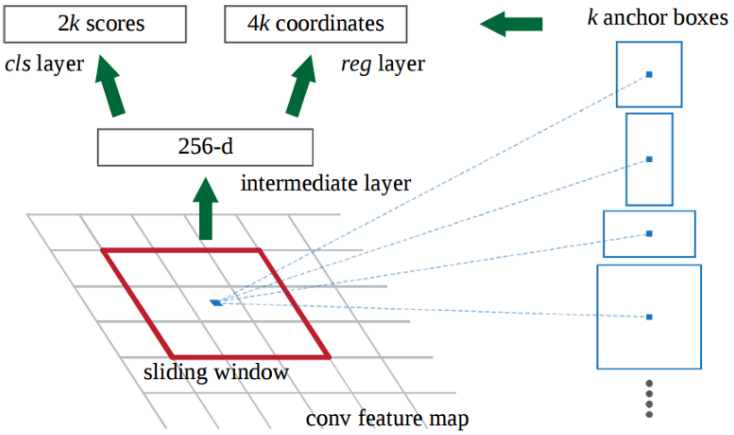

Figure 2.3: RPN

2.4 R-FCN

The Region-based Fully Convolutional Network (R-FCN)[1] released by J. Dai and al. (2016) is a model with only convolutional layers allowing complete backpropagation for training and inference. The authors have merged the two basic steps in a single model to take into account simultaneously the object detection (location invariant) and its position (location variant).

R-FCN works as follows:

- Run a CNN (in this case, ResNet) over the input image.
- Add a fully convolutional layer to generate a score bank of the aforementioned "position-sensitive score maps." There should be k(C+1) score maps, with k representing the number of relative positions to divide an object (e.g. 9 for a 3 by 3 grid) and C+1 representing the number of classes plus the background.
- Run a fully convolutional region proposal network (RPN) to generate regions of interest (RoI's).
- For each RoI, divide it into the same k "bins" or subregions as the score maps.
- For each bin, check the score bank to see if that bin matches the

corresponding position of some object. For example, if I'm on the "upper-left" bin, I will grab the score maps that correspond to the "upper-left" corner of an object and average those values in the RoI region. This process is repeated for each class.

- Once each of the k bins has an "object match" value for each class, average the bins to get a single score per class.
- Classify the RoI with a softmax over the remaining C+1 dimensional vector.

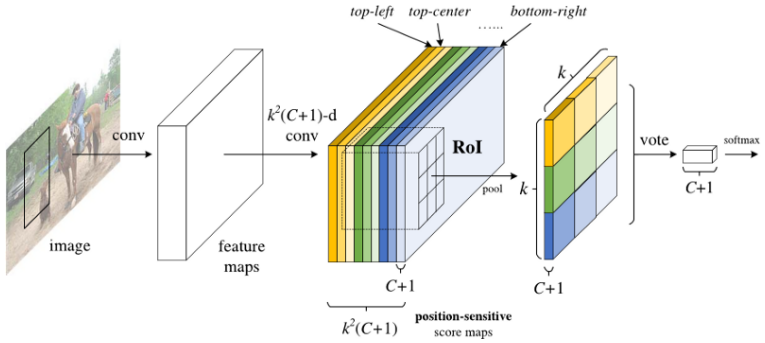

Figure 2.4: R-FCN

With this setup, R-FCN is able to simultaneously address location variance by proposing different object regions, and location invariance by having each region proposal refer back to the same bank of score maps.

2.5 SSD

SSD[4], which stands for Single-Shot Detector. Like R-FCN, it provides enormous speed gains over Faster R-CNN, but does so in a markedly different manner. SSD simultaneously predicts the bounding box and the class as it processes the image.

Concretely, given an input image and a set of ground truth labels, SSD does the following:

- Pass the image through a series of convolutional layers, yielding several sets of feature maps at different scales (e.g. 10x10, then 6x6, then 3x3, etc.)
- For each location in each of these feature maps, use a 3x3 convolutional filter to evaluate a small set of default bounding boxes. These default bounding boxes are essentially equivalent to Faster R-CNN's anchor boxes.
- For each box, simultaneously predict a) the bounding box offset and b) the class probabilities.
- During training, match the ground truth box with these predicted boxes based on IoU. The best predicted box will be labeled a "positive," along with all other boxes that have an IoU with the truth >0.5.

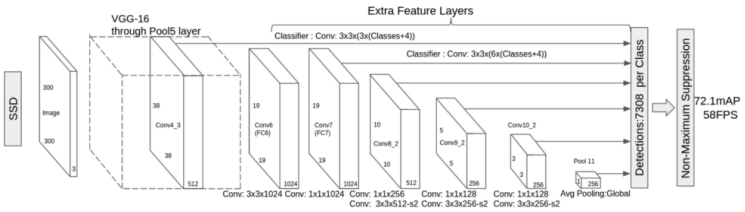

Figure 2.5: SSD

2.6 YOLO

The YOLO[5] model (J. Redmon et al., 2016)) directly predicts bounding boxes and class probabilities with a single network in a single evaluation. The simplicity of the YOLO model allows real-time predictions.

Initially, the model takes an image as input. It divides it into an SxS grid. Each cell of this grid predicts B bounding boxes with a confidence score. This confidence is simply the probability to detect the object multiply by the IoU between the predicted and the ground truth boxes.

The final layer outputs a S*S*(C+B*5) tensor corresponding to the predictions for each cell of the grid. C is the number of estimated prob-

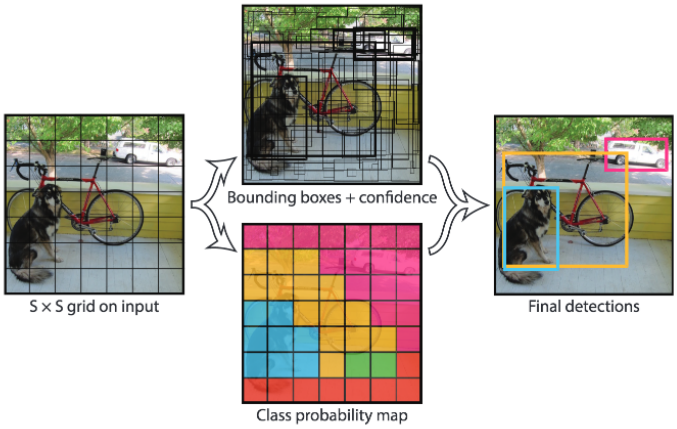

Figure 2.6: YOLO

abilities for each class. B is the fixed number of anchor boxes per cell, each of these boxes being related to 4 coordinates (coordinates of the center of the box, width and height) and a confidence value.

2.7 YOLOv2

The YOLOv2[6] model is focused on improving accuracy while still being a fast detector. Batch normalization is added to prevent overfitting without using dropout. Higher resolution images are accepted as input: the YOLO model uses 448x448 images while the YOLOv2 uses 608x608 images, thus enabling the detection of potentially smaller objects.

The final fully-connected layer of the YOLO model predicting the coordinates of the bounding boxes has been removed to use anchor boxes in the same way as Faster R-CNN. The input image is reduced to a grid of cells, each one containing 5 anchor boxes. YOLOv2 uses 19*19*5=1805 anchor boxes by image instead of 98 boxes for the YOLO model. YOLOv2 predicts correction of the anchor box relative to the location of the grid cell (the range is between 0 and 1) and selects the boxes according to their confidence as the SSD model. The dimensions

of the anchor boxes has been fixed using k-means on the training set of bounding boxes.

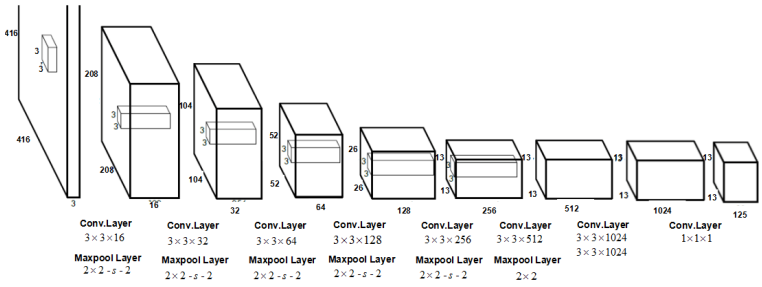

Figure 2.7: YOLOv2

Chapter 3

Object detection

3.1 Object Detection

Figure 3.1: Classification and Detection

Classification tells us what the "main subject" of the image is whereas object detection can find multiple objects, classify them, and locate where they are in the image.
For object detection we need to generate bounding boxes around the objects as well as find the class probabilities of these objects.

3.2 Classification

A classifier takes an image as input and produces a single output, the probability distribution over the classes. But this only gives us a summary of what is in the image as a whole, it doesn't work so well when the image has multiple objects of interest.

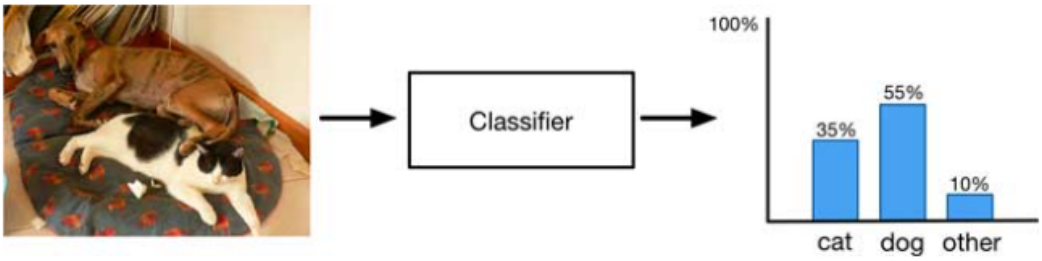

Figure 3.2: Classification

3.3 Detection

An object detection model, on the other hand, will tell you where the individual objects are by predicting a bounding box for each object:

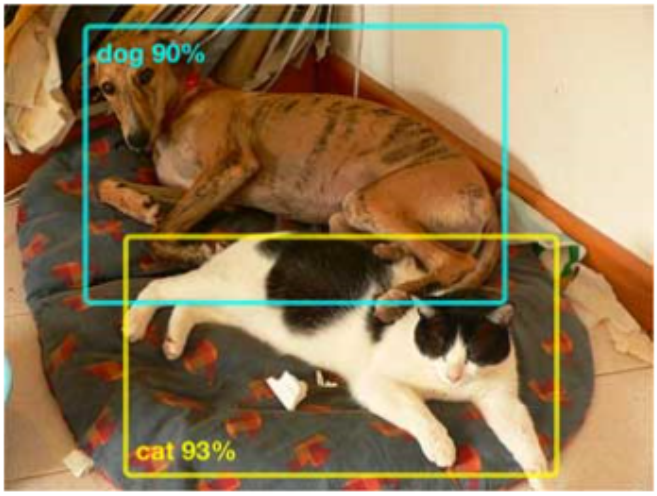

Figure 3.3: Detection

Chapter 4

YOLO Object Detection

4.1 Introduction

Object detection is the computer vision technique for finding objects of interest in an image. This is more advanced than classification, which only tells you what the "main subject" of the image is — whereas object detection can find multiple objects, classify them, and locate where they are in the image. An object detection model predicts bounding boxes, one for each object it finds, as well as classification probabilities for each object.

A high-end model like Faster R-CNN first generates so-called region proposals — areas of the image that potentially contain an object — and then it makes a separate prediction for each of these regions. That works well but it's also quite slow as it requires running the detection and classification portion of the model multiple times.

A one-shot detector, on the other hand, requires only a single pass through the neural network and predicts all the bounding boxes in one go. That is much faster and much more suitable for mobile devices. The most common examples of one-shot object detectors is YOLO.

You only look once (YOLO) is a state-of-the-art, real-time object detection system.It is easy to trade off between speed and accuracy simply by changing the architecture as YOLO has 2 architecture. Tiny YOLO is recommended for mobile devices as it involves less calculation hence it is faster but accuracy is less as compared to simple YOLO.

4.2 Architecture

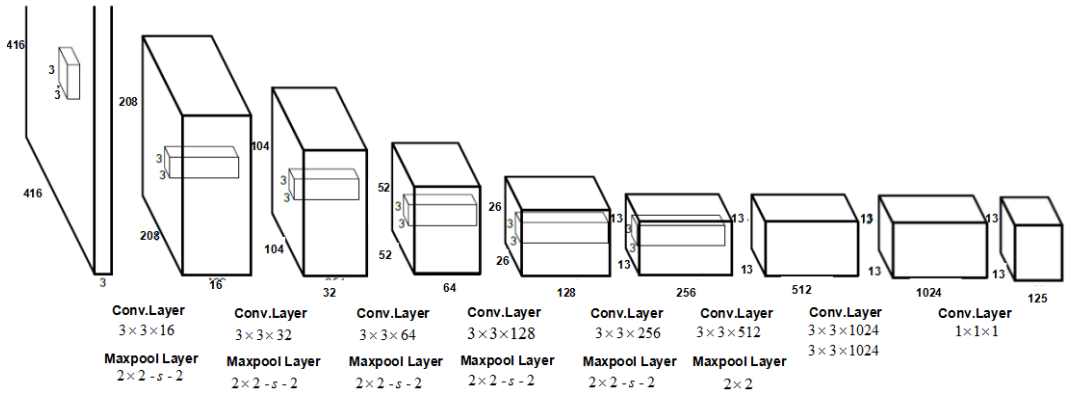

Figure 4.1: Tiny YOLO Architecture

4.3 Grid Cells

YOLO divides the input image into an S*S grid. Each grid cell predicts only one object. For example, the yellow grid cell below tries to predict the "person" object whose center (the blue dot) falls inside the grid cell.

For each grid cell :

- It predicts B boundary boxes and each box has one box confidence score.
- It detects one object only regardless of the number of boxes B.
- It predicts C conditional class probabilities (one per class for the likeliness of the object class).

4.4 Working

Each boundary box contains 5 elements: (x, y, w, h) and a box confidence score. The confidence score reflects how likely the box contains an object (objectness) and how accurate is the boundary box.

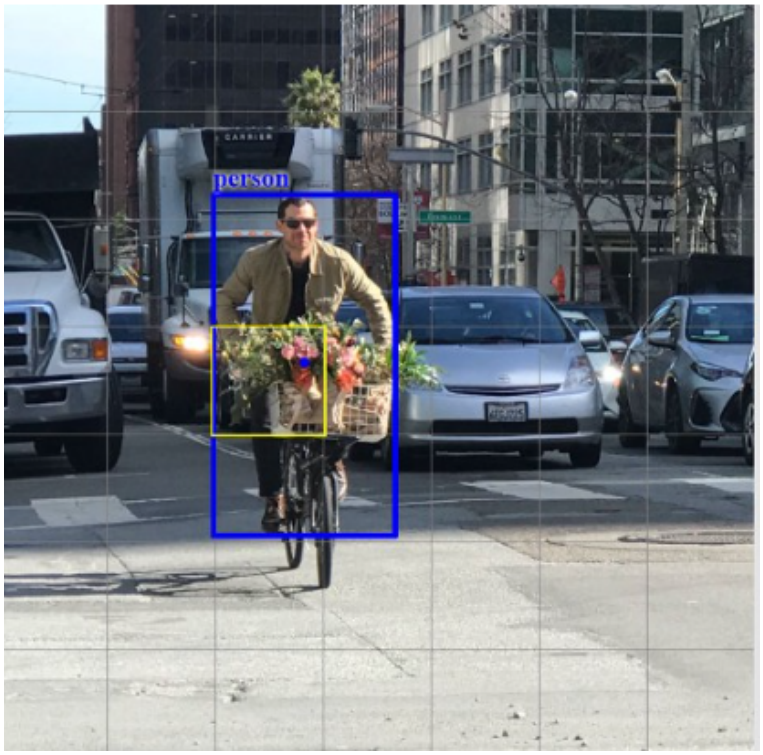

Figure 4.2: Object detection by a grid cell

We normalize the bounding box width w and height h by the image width and height. x and y are offsets to the corresponding cell. Hence, x, y, w and h are all between 0 and 1. Each cell has 20 conditional class probabilities. The conditional class probability is the probability that the detected object belongs to a particular class (one probability per category for each cell). So, YOLO's prediction has a shape of (S, S, 5(B + C)) = (13, 13, 5(5 + 20)) = (13, 13, 125).

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

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 $class_i$

4.5 Non-Maximum Suppression

When nearby cells tries to predict the same object, model might end up predicting multiple bounding boxes around the object.

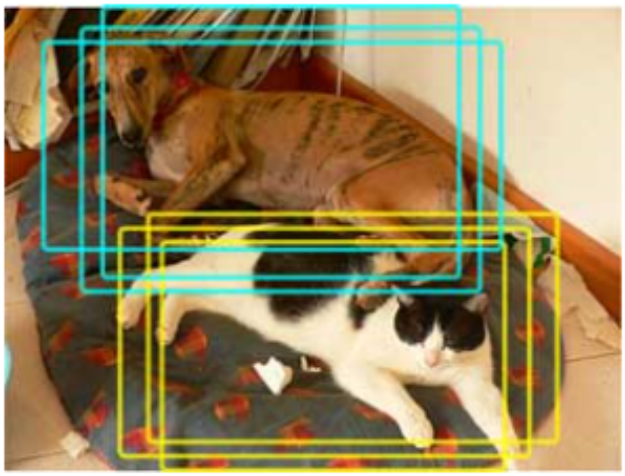

Figure 4.3: Multiple Bounding Box.

To remove these multiple bounding boxes around the same object, we use a post-processing technique non-maximum suppression. Non Maximum Suppression keeps the predictions with the highest confidence scores and removes any other boxes that overlap these by more than a certain threshold.

Here is one of the possible non-maximal suppression implementation:

- Sort the predictions by the confidence scores.
- Start from the top scores, ignore any current prediction if we find any previous predictions that have the same class and IoU > 0.5 with the current prediction.
- Repeat step 2 until all predictions are checked.

4.6 Anchors

The grid is used to specialize detectors to look only at certain spatial locations, and by having several different detectors per grid cell, each of

these object detectors specialize in a certain object shape as well.Since, the model has 13 x 13 grid cells and each cell has 5 detectors, so there are 845 detectors in total. It's hard for a detector to learn how to predict objects that can be located anywhere, it's also hard for a detector to learn to predict objects that can be any shape or size. Therefore, some fixed number of detectors are used.

YOLO chooses the anchors by running k-means clustering on all the bounding boxes from all the training images (with k = 5 so it finds the five most common object shapes). Therefore, YOLO's anchors are specific to the dataset that we're training (and testing) on.

The k-means algorithm finds a way to divide up all data points into clusters. Here the data points are the widths and heights of all the ground-truth bounding boxes in the dataset. If we run k-means on the boxes from the Pascal VOC dataset, we find the following 5 clusters:

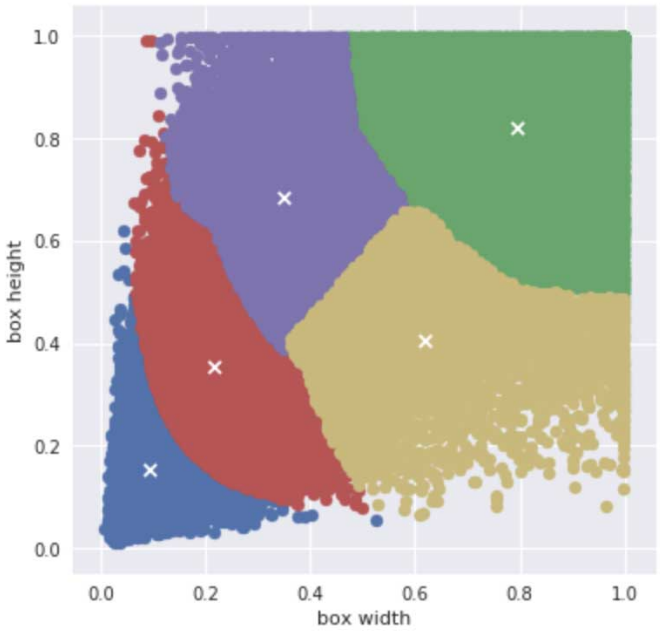

Figure 4.4: k-means on anchor boxes

These clusters represent five "averages" of the different object shapes that are present in this dataset. You can see that k-means found it necessary to group very small objects together in the blue cluster, slightly

larger objects in the red cluster, and very large objects in green. It decided to split medium objects into two groups: one where the bounding boxes are wider than tall (yellow), and one that's taller than wide (purple).

4.7 Loss Function

we want a loss function that encourages the model to predict correct bounding boxes and also the correct classes for these boxes. On the other hand, the model should not predict objects that aren't there. This is a complex task with multiple components. Therefore, our loss consists of several different terms (it's a multi-task loss). Some of these terms are for regression, since they predict real-valued numbers; others are for classification.

For any given detector, there are two possible situations:

1. This detector has no ground-truth associated with it. This is a negative example; it is not supposed to detect any objects (i.e. it should predict a bounding box with confidence score 0).

2. This detector does have a ground-truth box. This is a positive example. The detector is responsible for detecting the object from the ground-truth box.

YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function composes of:

1) The classification loss.
2) The localization loss (errors between the predicted boundary box and the ground truth).
3) The confidence loss (the objectness of the box).

Classification loss

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

where

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

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

Localization loss

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where

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

λ_{coord} increase the weight for the loss in the boundary box coordinates.

We do not want to weight absolute errors in large boxes and small boxes equally. i.e. a 2-pixel error in a large box is the same for a small box. To partially address this, YOLO predicts the square root of the bounding box width and height instead of the width and height. In addition, to put more emphasis on the boundary box accuracy, we multiply the loss by coord (default: 5).

Confidence loss

If an object is detected in the box, the confidence loss (measuring the objectness of the box) is:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where

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

λ_{coord} increase the weight for the loss in the boundary box coordinates.

If an object is not detected in the box, the confidence loss is:

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

where

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

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

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

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor noobj (default: 0.5).

Loss

The final loss adds localization, confidence and classification losses together.

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

Chapter 5

Tools and Utilities

5.1 Dataset

The main goal of our project is to recognize object. To achieve it we have done suvervised learning using the training set of labelled images. We have used the pascal VOC dataset having 20 classes as follows:

Person: Person

Animal: bird, cat, cow, dog, horse, sheep

Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train

Indoor: bottle, chair, dining table, potted plant, sofa, laptop The training data has 11,530 images.

5.2 TensorFlow

A commonly used software for ML tasks is TensorFlow. It is widely adopted as it provides an interface to express common Machine Learning algorithms and executable code of the models. Models created in TensorFlow can be ported to heterogeneous systems with little or no change with devices ranging from mobile phones to distributed servers. TensorFlow was created by and is maintained by Google, and is used internally within the company for Machine Learning purposes. TensorFlow expresses computations as a stateful data flow-graph.

As Machine Learning tasks are computationally expensive, model optimisation is used to improve performance. The minimum hardware requirements of Tensor Flow in terms of Random-Access Memory (RAM) size and CPU speed are low, and the primary bottleneck is the calculation speed of the computations as the desired latency for mobile applications is low.

5.3 Darknet

Darknet is an opensource neural network written in C and CUDA and supports the CPU and GPU computation to train the model. We used the darknet to train the yoloV2 model. We have installed darknet with two dependencies

- OpenCV: To provide a vider variety of supported image types.
- CUDA: For GPU computation (Version 9.1)
- CuDNN: Aditional libraries for CUDA (Version 7.1)

Python (Version 3.5) environment setup.

- Keras
- Tensorflow
- Pandas
- NumPy
- OpenCV
- Statistics
- H5py

We have used NVIDIA GEFORCE GTX 960M (4GB) graphic card for training and testing purposes on the device.
The training of the data generated a .weights file which is used in darkflow to generate the protobuf (.pb) file.

5.4 Darkflow

Darkflow is an open source framework that can be used to train the models.
We have installed Darkflow with following dependencies

- OpenCV3
- Tensorflow (Version 1.0)
- Python (Version 3.5)
- NumPy

Using the .weights file generated by darknet and the config file of YoloV2 we have exported the yoloV2 model to tensorflow.
flow –model cfg/yolo.cfg –load bin/yolo.weights –savepb
The protobuf file is used as a model in android application for testing.

5.5 Application

A bounding box's top left corner's x and y coordinates are found with their height and width. Each bounding box is associated with a confidence value and class.

Recognition class is used to store the detected objects. A list of recognition is given as output by the classify image function. The recognition object includes the id of the object recognized, title of its classified label, confidence associated with that object and the location of the box.

classifyImage function is used to classify the image using the tensorflow output. It classifies the object on image. Parameter tensorFlowOutput is output from the tensorflow, it is 13x13x125 tensor. 125 = (number of classes + x co-ordinate, y co-ordinate, height, width, confidence)*5 because we have 5 boxes for each grid. Parameter labels is string vector with the labels. This function returns the list of recognition objects.

```java
public class BoundingBox {
    private double x;
    private double y;
    private double width;
    private double height;
    private double confidence;
    private double[] classes;

    public double getX() { return x; }

    public void setX(double x) { this.x = x; }

    public double getY() { return y; }

    public void setY(double y) { this.y = y; }

    public double getWidth() { return width; }

    public void setWidth(double width) { this.width = width; }

    public double getHeight() { return height; }

    public void setHeight(double height) { this.height = height; }

    public double getConfidence() { return confidence; }

    public void setConfidence(double confidence) { this.confidence = confidence; }

    public double[] getClasses() { return classes; }

    public void setClasses(double[] classes) { this.classes = classes; }
}
```

Figure 5.1: Bounding Box Class Snippet

```java
public final class Recognition {

    private final Integer id;
    private final String title;
    private final Float confidence;
    private BoxPosition location;

    public Recognition(final Integer id, final String title,
                       final Float confidence, final BoxPosition location) {
        this.id = id;
        this.title = title;
        this.confidence = confidence;
        this.location = location;
    }

    public Integer getId() { return id; }

    public String getTitle() { return title; }

    public Float getConfidence() { return confidence; }

    public BoxPosition getLocation() { return new BoxPosition(location); }

    public void setLocation(BoxPosition location) { this.location = location; }

    @Override
    public String toString() {
        return "Recognition{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", confidence=" + confidence +
                ", location=" + location +
                '}';
    }
}
```

Figure 5.2: Recognition class

```java
public List<Recognition> classifyImage(final float[] tensorFlowOutput, final Vector<String> labels) {
    int numClass = (int) (tensorFlowOutput.length / (Math.pow(SIZE,2) * NUMBER_OF_BOUNDING_BOX) - 5);
    BoundingBox[][][] boundingBoxPerCell = new BoundingBox[SIZE][SIZE][NUMBER_OF_BOUNDING_BOX];
    PriorityQueue<Recognition> priorityQueue = new PriorityQueue<>(MAX_RECOGNIZED_CLASSES, new RecognitionComparator());

    int offset = 0;
    for (int cy=0; cy<SIZE; cy++) {        // SIZE * SIZE cells
        for (int cx=0; cx<SIZE; cx++) {
            for (int b=0; b<NUMBER_OF_BOUNDING_BOX; b++) {   // 5 bounding boxes per each cell
                boundingBoxPerCell[cx][cy][b] = getModel(tensorFlowOutput, cx, cy, b, numClass, offset);
                calculateTopPredictions(boundingBoxPerCell[cx][cy][b], priorityQueue, labels);
                offset = offset + numClass + 5;
            }
        }
    }

    return getRecognition(priorityQueue);
}
```

Figure 5.3: Storing recognitions in list

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

In this project, we have presented a study of different image classification and image detection algorithm. We have used the algorithm YoloV2 to classify and detect the image. Unlike classifier based approaches, Yolo is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. We trained the YoloV2 model on Pascal VOC (2012) data set on laptop and generated a pre-trained model. The model is used in android application using tensorflow API. For classification of the object we have set the threshold confidence value to 60% and successfully classified most of the daily objects with confidence value in between 70% to 80%. We observed that the application find it difficult to classify and detect the objects in low light or when the object is out of focus.

6.2 Application

Face detection Popular applications include face detection and people counting.
People Counting Object detection can be also used for people counting, it is used for analyzing store performance or crowd statistics during festivals.
Vehicle detection Similarly when the object is a vehicle such as a

bicycle or car, object detection with tracking can prove effective in estimating the speed of the object.

Manufacturing Industry Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects.

Security In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quad-copter).

6.3 Future Work

We have shown the implementation of the image classification and detection algorithm on android mobile device. As it can be uncomfortable for a person to use the mobile device in such a way to identify the objects in the surroundings, this can be further implemented on the glasses (like google glass) or a hardware device can be made which can take the images of the surrounding and detect the objects.

Bibliography

[1] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

[2] R. Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[7] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.