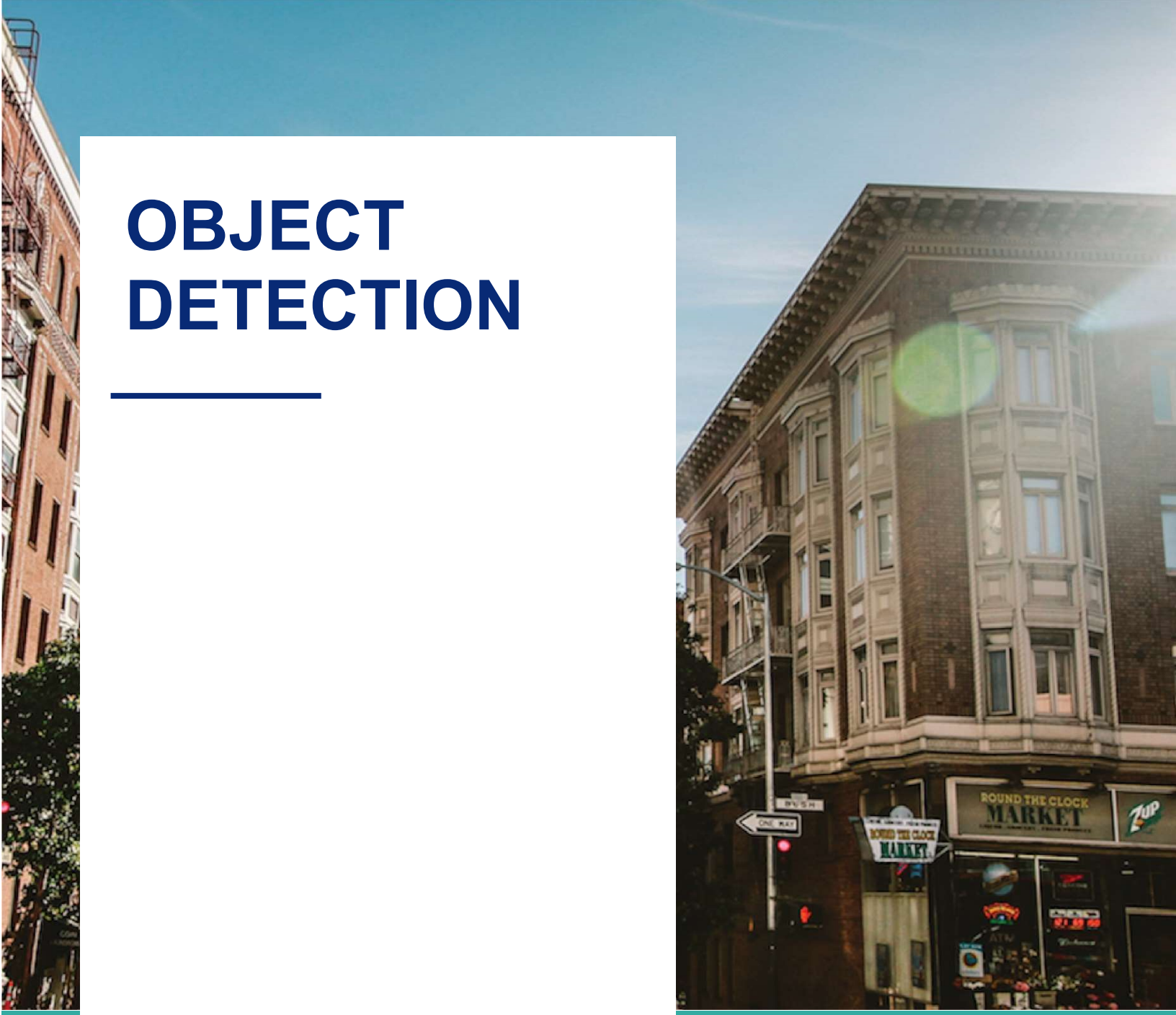
OBJECT DETECTION



PROJECT SYNOPSIS

DJS SYNAPSE

Authored by: Jazib Dawre

20/09/2020

Introduction

Title

Object Detection using YOLOv3 on darknet

Aim

Detect classes of objects from a given picture stream in real time

Overview

Object detection is a computer vision task wherein we have to identify and localize an object within an image frame. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labeling them.

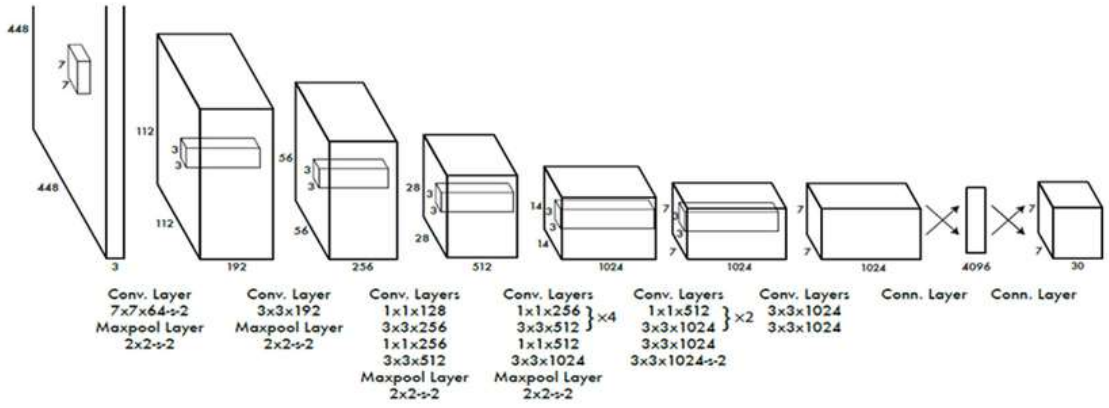
This can be achieved by a variety of algorithms from Region-based Convolutional Neural Networks (R-CNNs) to Single Shot Detectors (SSDs). All these algorithms however more often than not trade speed for accuracy. In this project where we have to perform object detection on a live video stream, an RCNN would require powerful computational power to even operate at around ~10 fps. Even the Faster R-CNN model struggles to reach 20 fps while the SSDs pull off 30-60 fps. Amidst all this, the YOLO algorithm breezes through at 40-90 fps while losing less than 10% accuracy compared to SSDs.

Considering that I don’t want to cook food on my PC and for a respectable frame rate, the YOLO algorithm provides a good enough balance between the two even with the lower accuracy compared to the other models.

Methodology

YOLO

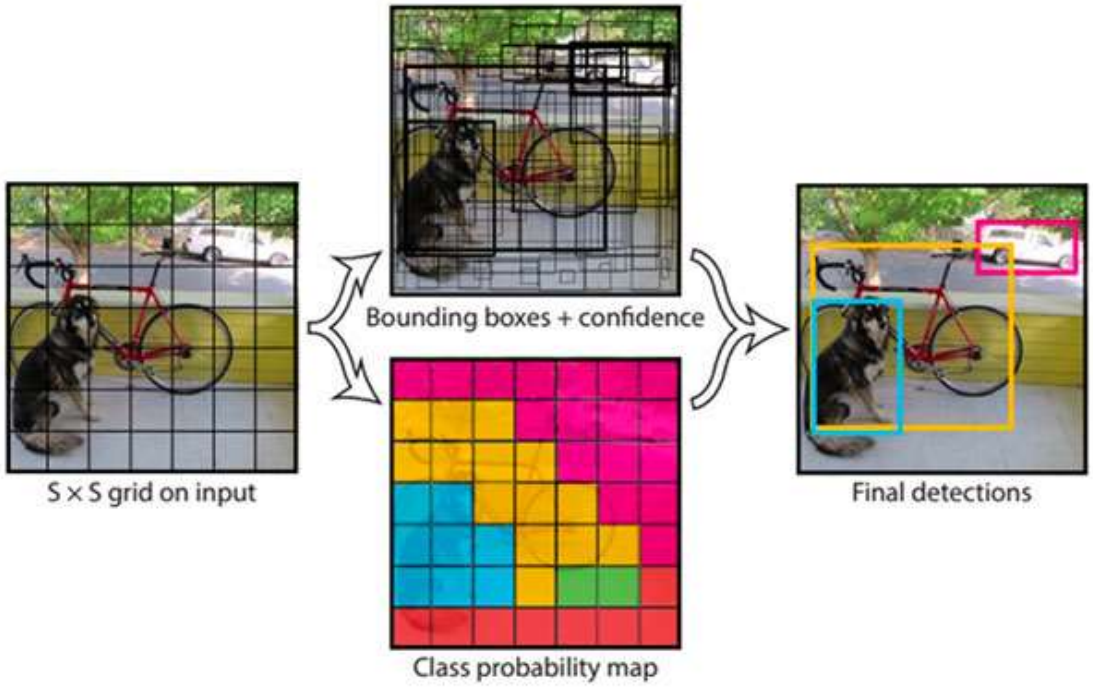
You only look once (YOLO) is a state-of-the-art, real-time object detection system introduced in 2015 paper by Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". Unlike RCNNs which first find "regions of interest" in an image and then run each region through the NN for prediction (which takes an awful lot of time), the YOLO algorithm runs the image only once through the NN to identify the objects. In brief, the algorithm first divides an into image into a grid of small images and assigns the grid cell containing the center of the identified object as the identified object along with the height, width and confidence for making the bounding box.

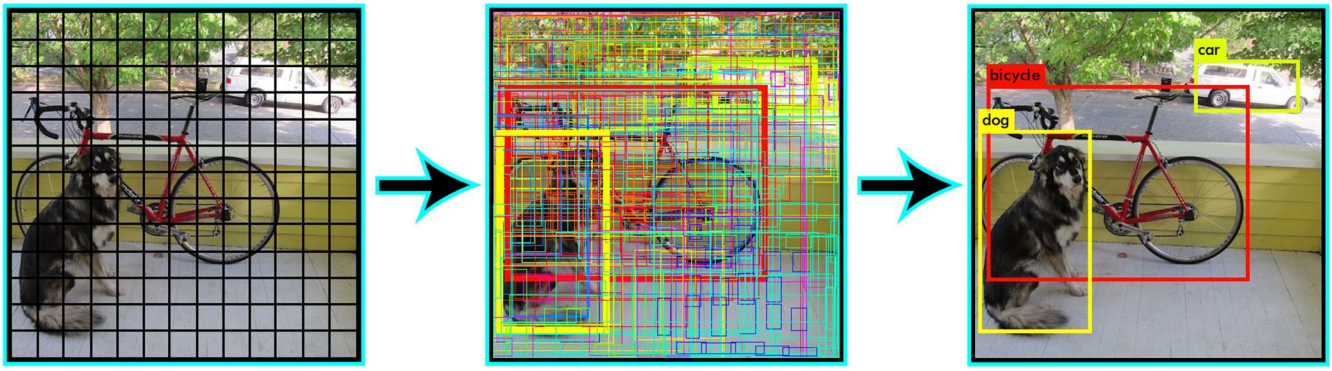


The YOLO architecture

For larger objects in the frame, it may happen that many grid cells may lie in the central region on the image and will be assigned as the object, aka multiple predictions for one object. To counter such cases, the overlap between bounding boxes is calculated, and based on user defined limit (e.g. 80%) all boxes with overlap > 80% for the same object will be discarded keeping only the one with the highest confidence value.

Working

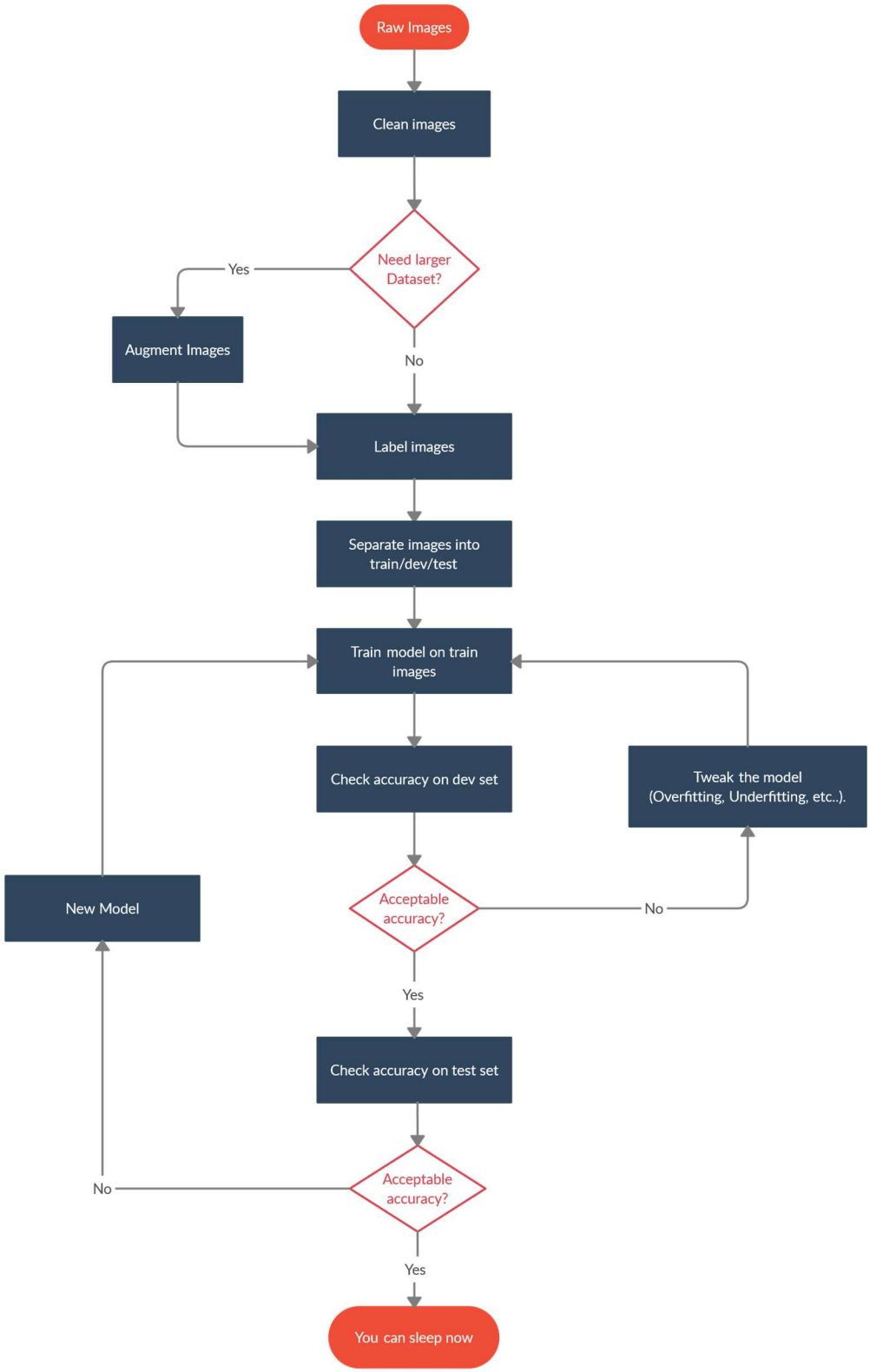




Input image **All bounding boxes** **Prediction**

This is just a high-level explanation of my understanding of the YOLO algorithm. A detailed explanation can be found in the research paper here.

Planning


Raw Images
Clean images
Need larger Dataset?
Yes
Augment Images
No
Label images
Separate images into train/dev/test
Train model on train images
Check accuracy on dev set
Tweak the model (Overfitting, Underfitting, etc..).
Acceptable accuracy?
No
New Model
Yes
Check accuracy on test set
Acceptable accuracy?
No
Yes
You can sleep now

Implementation

Training and Testing

For my implementation, I plan to use the YOLOv4 pre-trained algorithm and further train it on our custom objects. I would use the darknet framework and train the model using google colab because again, no cooking on my pc. Apart from the above attached flowchart that I made to depict the pipeline of training and evaluating the model some model specific steps to implement the YOLO algorithm are as follows:

1. Download the darknet framework (it already has YOLO algorithms)
2. Change the config file (the no of classes, using gpu/cuda, etc.)
3. Download the pretrained weights for the model (optional but will save our time)
4. Prepare the dataset as detailed in the above diagram
5. Start training the model while monitoring the loss and accuracy
6. Check for accuracy on dev set, tweak if required
7. Finally test the model on your test set

Real Time

For real-time execution of YOLO, we will use the python OpenCV library. OpenCV has a function VideoCapture() which takes in a live stream from a camera (e.g. webcam) and breaks it into individual frames of images on which we can run the YOLO algorithm. Darknet has prebuilt support for running YOLO on webcam which outputs a video stream with the bounding boxes. For a manual run, code will look something like this:

```
import cv2

cap = cv2.VideoCapture(0) #capturing the webcam

while(True):
    ret, frame = cap.read() #capture each frame

    #run YOLO on each frame here

#when everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```