Multi-class Object Detection in Images

A Project Report

Presented to

Professor Mark Stamp

Department of Computer Science

San Jose State University

In Fulfillment

Of the Requirements for the Class

CS 271

By

Atharva Khadilkar and Mrunal Zambre

November 2022

Table of Contents

I. INTRODUCTION

A long-standing goal of machine learning has been to create a system capable of recognizing a wide variety of objects in a cluttered environment. Object detection is a computer vision technology that locates items in images or videos. To produce meaningful results, object detection algorithms often use machine learning or deep learning.

Multiclass object detection has many real world applications, in autonomous vehicles[1], in autonomous systems for drug packaging [2]. Object detection is a critical part of advanced driver assistance systems (ADAS), which allow cars to detect driving lanes and detect pedestrians to improve road safety. Object detection is also useful in video surveillance and picture retrieval systems [1].

II. PROJECT OBJECTIVE

The objective of this project is to detect 5 different classes in a given image. These classes are - car, truck, pedestrian, traffic light and bicyclist. The task of object detection will be done using 2 different machine learning algorithms; SVM and YOLO algorithm. The project aims to create a bounding box around objects in a given image and predict the object class. After implementing both approaches, the goal is to compare the accuracies and the time taken for each approach to detect objects.

III. DATASET

The dataset that was used for this project was the ‘Self-driving cars’ dataset on Kaggle [3]. This dataset contains 22,241 images, divided into train and test splits. The images contain multiple objects from the given 5 classes. There are also 2 label_train.csv and label_test.csv files, which contain coordinates for each class object in each image. Using these .csv files, we have

cropped out the various objects according to their classes and used these cropped images to train the SVM classifier later.

Fig 1: Example image from image dataset

frame	xmin	xmax	ymin	ymax	class_id
1478019952686311006.jpg	237	251	143	155	1
1478019952686311006.jpg	437	454	120	186	3
1478019953180167674.jpg	218	231	146	158	1
1478019953689774621.jpg	171	182	141	154	2
1478019953689774621.jpg	179	191	144	155	1
1478019953689774621.jpg	206	220	145	156	1
1478019953689774621.jpg	385	420	122	152	1
1478019953689774621.jpg	411	462	124	148	1
1478019954186238236.jpg	165	177	140	154	2

Fig 2: Sample from labels_train.csv file

IV. MACHINE LEARNING METHODS

We have used 2 machine learning methods to implement object detection, namely, Support Vector Machine (SVM) and YOLO Algorithm using Convolutional Neural Networks.

A. Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for classification problems. We plot each data point as a point in n-dimensional space (here, n is the total number of features used) with the coordinate value being the value of the feature. Then, we find a hyperplane (decision boundary) that differentiates between the classes very well, which enables us to perform classification.

To perform multi-class object detection in images, we had to implement multiple SVMs. Since we had 5 classes to detect, we trained 5 different SVMs, one for each class. In order to do this, we had to pre-process the images and perform feature extraction methods. In the end, we had 5 binary classifiers, which predicted whether the input image belongs to a particular class or not.

i.) Pre-processing the data

Since our dataset contained images containing multiple classes, we had to first split the data according to the 5 classes. In the labels.csv files, coordinates of each object in each image, alongwith the class label were given. According to these coordinates, we cropped the images so that each image contained only that class object. These images were padded and resized to a constant 96x24 size to maintain consistency. Then, for each classifier, we prepared 2 training and testing data subsets - one with the cropped images containing that class, and the other with images not containing that class.

ii.) Feature extraction

We used a technique called Histogram of Oriented Gradients (HOG) for feature extraction in the images. Since images of a class of objects vary so much in color, structural cues

like shape give a more robust representation of the object. Gradients in specific directions can capture some notion of shape.

The idea of HOG is that instead of using each individual gradient direction of each individual pixel of an image, we group the pixels into small cells. For each cell, we compute all the gradient directions and group them into a number of orientation bins. We sum up the gradient magnitude in each sample. Then, stronger gradients contribute more weight to their bins, and effects of small random orientations due to noise are reduced. This histogram gives us a picture of the dominant orientation of that cell. Doing this for all cells gives us a representation of the structure of the image. The HOG features keep the representation of an object distinct but also allow for some variations in shape [4].

We implemented HOG feature extraction using the *hog* function from *skimage.feature* library. Fig 3 shows an example of the HOG features that were extracted from a sample image of the class 'car'.

Fig 3: HOG features for a sample image

iii.) Training the classifiers

The next step was to train the 5 different SVM classifiers. We used the SVC function from *sklearn* library for this. Given the large amount of training data, it took around 10 minutes for the SVMs to train. The test accuracies we got for each SVM are:

Accuracy for SVM_car: 93.0618

Accuracy for SVM_truck: 93.398

Accuracy for SVM_pedestrian: 95.6257

Accuracy for SVM_bicyclist: 96.8197

Accuracy for SVM_light: 96.0867

iv.) Object detection

In order to detect the objects and classify them in the test images, we had to perform sliding window and heatmap methods.

In the sliding window method, we first split the images by 96x64 boxes. This is intentionally the same size as our input images for the classifiers. After that, we classified all the boxes. If a box is positive (it includes that class), it is marked as a potential region, otherwise it is not considered further. However, if we consider that all the regions we classify as positive are true, there would be many false detections. [4]

In order to solve this problem, we used heatmaps. In heatmap, we first created a black image (all pixel values are zero) with the shape of the original image. And if we detected a car in a region, we added some value to the region's pixel value in the black image. Then, we considered only those regions with more heat value as true positives and drew the bounding boxes around them.

iv.) Testing and Results

With SVM, it is not possible to detect all classes simultaneously in any given image. So, we predicted one class per image at a time. Each prediction task took around 5 to 10 seconds.

Fig 4: Prediction using SVM_car

Fig 5: Predicted objects using SVM_truck

However, the classifiers were not very accurate in detecting the objects. The bounding boxes it drew did contain the classes we were trying to predict, but it also predicted many false positives. It also took time to perform the object detection and classification. In a real-world scenario, we cannot afford to use this much time. Additionally, we had to train 5 different SVMs for each class, which increased computing time.

B. YOLO Algorithm

YOLO (You only look once) is a CNN based algorithm which is mainly used for object detection within images. The main concept behind the YOLO algorithm lies within its name itself, we only look once through the image to predict all the different classes. Compared to classifiers before YOLO which work on classification one at a time by creating bounding boxes before classification, YOLO proposes an end-to-end neural network for creation of bounding boxes and classification all at once.

The YOLO performs the bounding box creation and classification of objects within the bounding box by making use of Intersection over Union (IOU). YOLO divides the given image into a N grids, each having a dimension D*D. Each of the N divided grids is then used to localize

and determine the object it contains.

Since the computation is done within each grid, it is computationally cheaper to localize and detect objects for specific grids. Each bounding box has a probability associated with it along with the object which it classifies. Given the nature of our grid splits we can have multiple bounding boxes for a single object within the given image. Hence this creates a problem of having multiple bounding boxes for a single object. YOLO deals with this problem using Non Max Suppression (NMS). As the name suggests, YOLO suppresses all the bounding boxes with lesser probabilities associated with it. After this it decides a single bounding box by suppressing the bounding boxes with less IOU.

$$IOU = area\ of\ overlap / area\ of\ union$$

IOU gives us the ratio of area of overlap over area of union which tells how good the current estimate of the bounding box is. So we calculate the area of all the remaining bounding boxes and choose the bounding box having the highest overlap with it. It gives a pretty good estimate of the bounding box associated with the object in the image.

i.) Architecture

For our project we have used YOLOv5, an architecture developed by Ultralytics[5]. YOLOv5 architecture can be primarily divided into 3 main parts; Backbone, Neck and head. The backbone which learns the basic feature representations is made up of a neural network known as DarkNet-53. Darknet-53 is a convolutional Neural Network which was used as a backbone for YOLOv3 [6]. DarkNet-53 is an improvement over its predecessor DarkNet-19 due to its inclusion of residual connections and more layers. The Neck of YOLOv5 consists of a Spatial pixel pair feature (SPPF) layer to mix and combine the features learnt by backbone and further pass on to the head of the network for prediction. The head of the architecture is similar to the head used by previous YOLO architectures such as YOLOv4 and YOLOv3. The head contains 53 more layers similar to DarkNet-53 for the creation of bounding boxes and prediction of classes.

Following is a high level architecture of the YOLO architecture provided by Ultralytics [5].

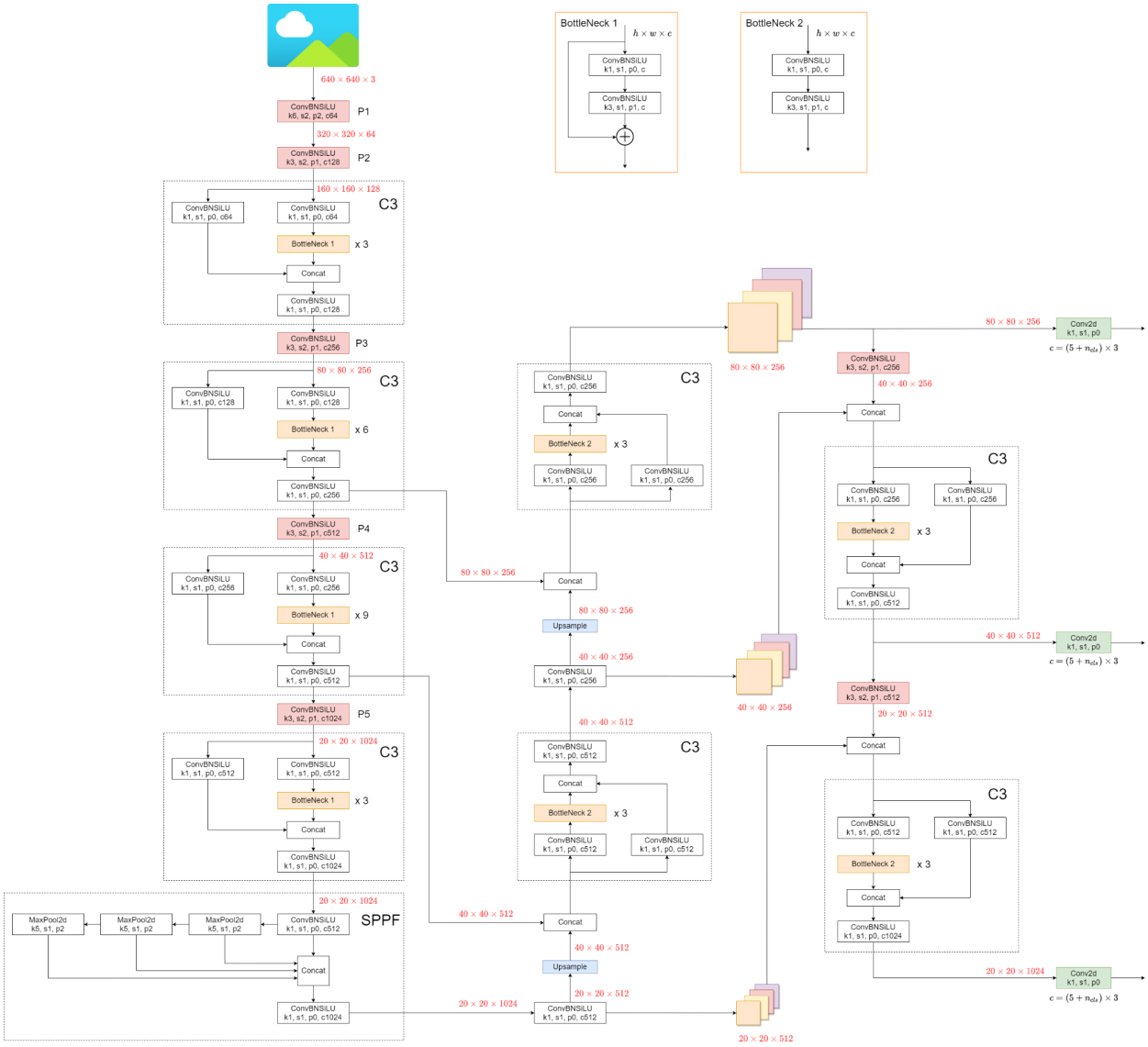

Fig 6: YOLOv5 architecture

ii.) Pre-processing

The YOLOv5 requires the input data to be preprocessed in a specific format. The architecture takes images and labels as inputs, with each image having a .txt file associated with it which has the dimensions of the bounding box for each class within the image. The documentation provided along with YOLOv5 mentions the required input formats for YOLOv5 [7].

The existing data consisted of Images along with a CSV file containing the information for the bounding boxes for each of the frames contained within images. We extracted information

for each image from the CSV file and converted it into the required format (*c,x,y,w,h*) for each object. Where *c* represents the class of the object enclosed within the bounding box, *x* and *y* denote the centerpoint of the bounding box and *w* and *h* denote the width and height of the bounding box. We then copy the converted data into a .txt file for each of the images present in the dataset. The following image shows a sample text file and the contents within.

```
0    0.5083333333333333    0.49666666666666665    0.029166666666666667    0.04
2    0.928125    0.51    0.035416666666666666    0.22
```

Fig 7: Bounding box file format

The image corresponding to this txt file will have two objects of classes 0-car and 2-pedestrian. Since the YOLOv5 uses .yaml files instead of .cfg files to load its configuration which specifies the architecture along with other parameters such as classes we had to create a new configuration file for our specific purposes where the number of classes was 3 instead of the default 80.

iii.) Training

For the training part, we trained the network with the input images retrieved from the dataset along with the generated .txt files for each image. The training was done for 10 epochs for 18000 images out of the 22000 images in the dataset. The following graphs show the change in losses over each epoch.

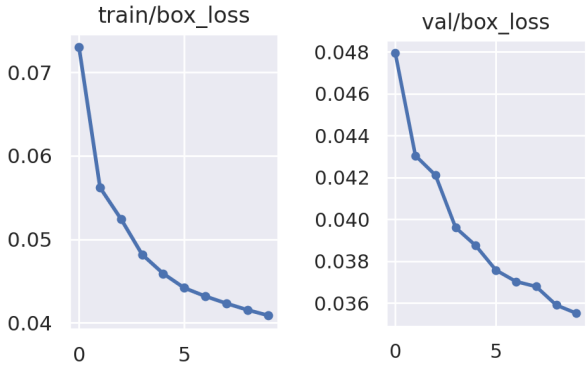

Fig 8: box_loss vs epoch

The box_loss determines the loss per each bounding box within the image. From the images it is clear that the model is doing a good job of predicting the bounding box around the classes.

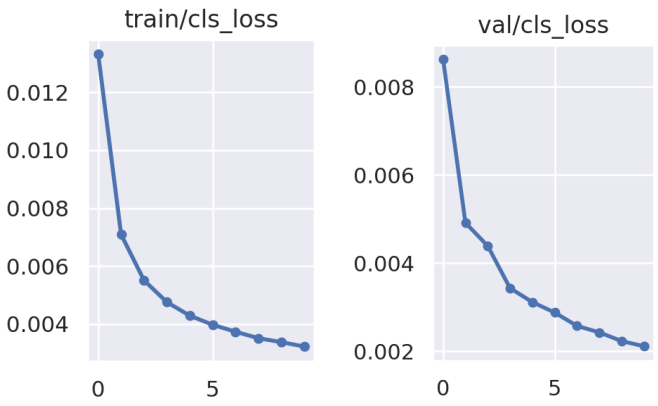

Fig 9: cls_loss vs epoch

The cls_loss determines the loss while classifying each class within the bounding box. We can see here that the model is fitting well for the given data and should do a consistent job of predicting the classes for the bounding boxes.

iv.) Testing and Results

When the model is used to predict the classes along with their bounding boxes it does a fairly good job of both. Following are sample output images classified using YOLOv5.

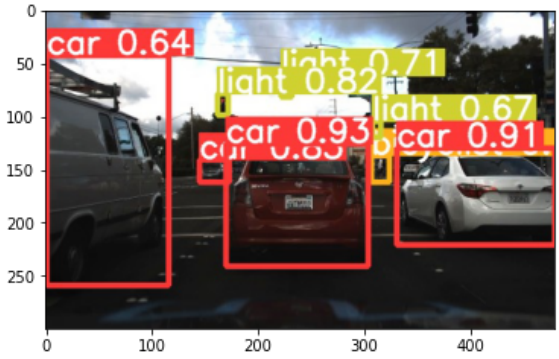

Fig 10. Prediction using YOLO

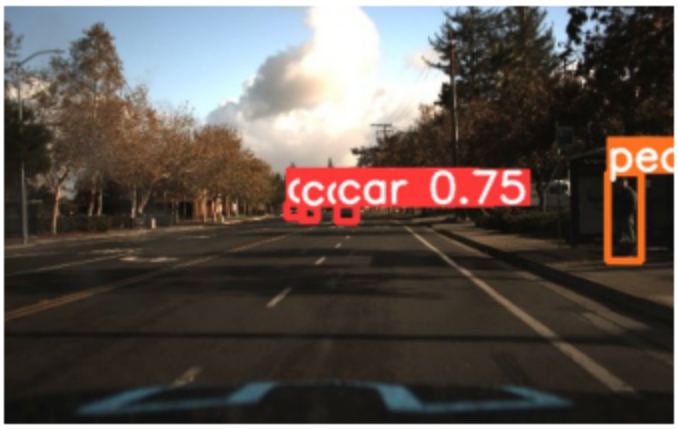

Fig 11. Prediction using YOLO

As visible the classifier does a good job of predicting the different classes in the given image datasets.

Along with proper classification, the main reason why YOLO is better than other traditional methods for multiclass classification is the speed for inferences. For the classification purposes each image had an average processing speed of 9ms. Which rounds up to around 111fps.

V. RESULTS

As we can see, the YOLO algorithm is much more accurate and faster than SVM. Where SVM took 7 seconds to predict the objects, YOLO was able to do it in 9ms. The bounding boxes predicted and the classification of objects within the bounding boxes done by YOLO were much more accurate. Considering the amount of frames which can be processed per second, YOLO can be used for multiclass classification in live videos as the max number of frames in videos is usually limited to 30.

VI. CONCLUSION

From this project, we can conclude that the YOLO algorithm works great for multi-class object detection in images. To use a classic machine learning technique like SVM, we require considerable computational resources and time, and we may still get a low accuracy. This helps us understand how efficient the YOLO algorithm is in terms of processing speed. Given its efficiency, the YOLO algorithm can be used in real time for multiclass classification. This could be widely used in real life applications such as live monitoring and vehicle tracking for autonomous vehicles.

REFERENCES

[1] Mehreen Saeed (2022, January). *Object detection in computer vision: A guide - blog*. AI Exchange. (n.d.). https://exchange.scale.com/public/blogs/object-detection-in-computer-vision-a-guide

[2] Boesch, G. (2022, February 11). *Applications of computer vision in the pharmaceutical industry*. viso.ai. Retrieved November 29, 2022, from https://viso.ai/applications/computer-vision-in-the-pharmaceutical-industry/

[3] https://www.kaggle.com/datasets/alincijov/self-driving-cars?resource=download

[4] Mithi. (2020, April 19). *Vehicle detection with hog and linear SVM*. Medium. https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a

[5] Ultralytics. (n.d.). *Ultralytics/yolov5: Yolov5 🚀 in PyTorch > ONNX > CoreML > TFLite*. GitHub. from https://github.com/ultralytics/yolov5

[6] Redmon, J. (n.d.). Yolo: Real-time object detection. from https://pjreddie.com/darknet/yolo

[7] Solawetz, J. (2020, September 29). *How to train a custom object detection model with Yolo V5*. Medium. from https://towardsdatascience.com/how-to-train-a-custom-object-detection-model-with-yolo-v5-917e9ce13208