CS 682 COMPUTER VISION

EMERGENCY VEHICLES CLASSIFICATION

SIDDI AVINASH CHENMILLA
MURTUZA SHAREEF
SHUBHAM PUDI
SUNCHIT SEHGAL

TABLE OF CONTENTS

Abstract

Due to its close relationship with video understanding and image analysis, object detection has attracted much attention over the recent years. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance stagnates with more complex structures which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in deep learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures.

YOLO (You-only-Look-Once) is an object detection system targeted for real-time processing which outperforms other methods when generalizing from natural images to other networks. The approach has been a better option to balance the trade-offs between faster real-time processing and near accurate results as compared to other approaches in object detection.

Our project aims to use the existing YOLO architecture to differentiate between Emergency and Non-Emergency vehicles, thereby, classifying them further into Police Vehicles, Ambulances and Fire-Trucks.

Introduction & Related Work

We often see that emergency vehicles face difficulty in passing through heavy traffic. So, identifying emergency vehicles in a traffic by differentiating a vehicle into an emergency and non-emergency category can be an important component in traffic monitoring - as being on time is critical for these services. In this project, we will attempt to build a computer vision model that can classify vehicles as emergency and non emergency vehicles further classifying them into Police Cars, Ambulances and Fire Trucks .

General Approach

The main idea is to build a solution that can reliably not just identify the emergency vehicles, but also classify them into one of the 4 classes – Police, ambulance, fire truck, non-emergency vehicles and also draw a bounding box around the object. The impetus was also on making it portable enough to run on computationally cheaper hardware –to be able to run it on traffic monitoring software. We decided on training our classifier on two CNN architectures : VGG Net and Tiny YOLO

Technical Approach

Transfer Learning

Simply put - using the architecture of a popular pre-trained model to solve our problem.

The advantages for going with transfer learning are :

- Less time to train a model
- Improved performance in terms of accuracy.
- Resistant to overfit
- Generalized model as the output

Transfer Learning using VGG Net

This is the first model – we tried to use. At a high level - the following steps are involved:

1. Preprocessing:

   VGG-16 requires the inputs to be in the order of -1 to +1, so the preprocessing step had to normalize the data to cater to this requirement
2. Load the pre-trained weights:

   Load all the weights of the model that was pre-trained on the MNIST dataset. It has an input layer, multiple convolution and pooling layers and finally flattens out into a fully connected layer with 1000 neurons in the last stage supporting 1000 different outputs of the MNIST dataset.
3. Exclude the last 4 layers:

   This step makes sure that we load all the layers into the dataset - excluding the last 4 layers of the neural network (which contain the softmax layer catering to 1000 outputs etc
4. Build a neural network:

   This involves a sequence of steps including
   - Input layer that can hold the existing weights
   - Fully connected layer with 1024 neurons and ReLU activation func
   - Fully connected layer with 2 neurons and softmax activation.
5. Train the neural network:

   Now using the data we created - train the network for the input data and see how well the classification works.

Though this model worked well in differentiating the emergency vs non-emergency vehicles – it was failing to classify the other types of emergency vehicles into police, ambulance, fire truck etc., probably given that the data we had was contributing to that.
Another limitation was that the model was very heavy and running it took a lot of time which is against one of the requirements we started with.

As we did not have enough images of each of the classes of emergency vehicles, we used the concept of transfer learning i.e. we trained out model based on the weight of already trained model.

YOLO - You Only Look Once

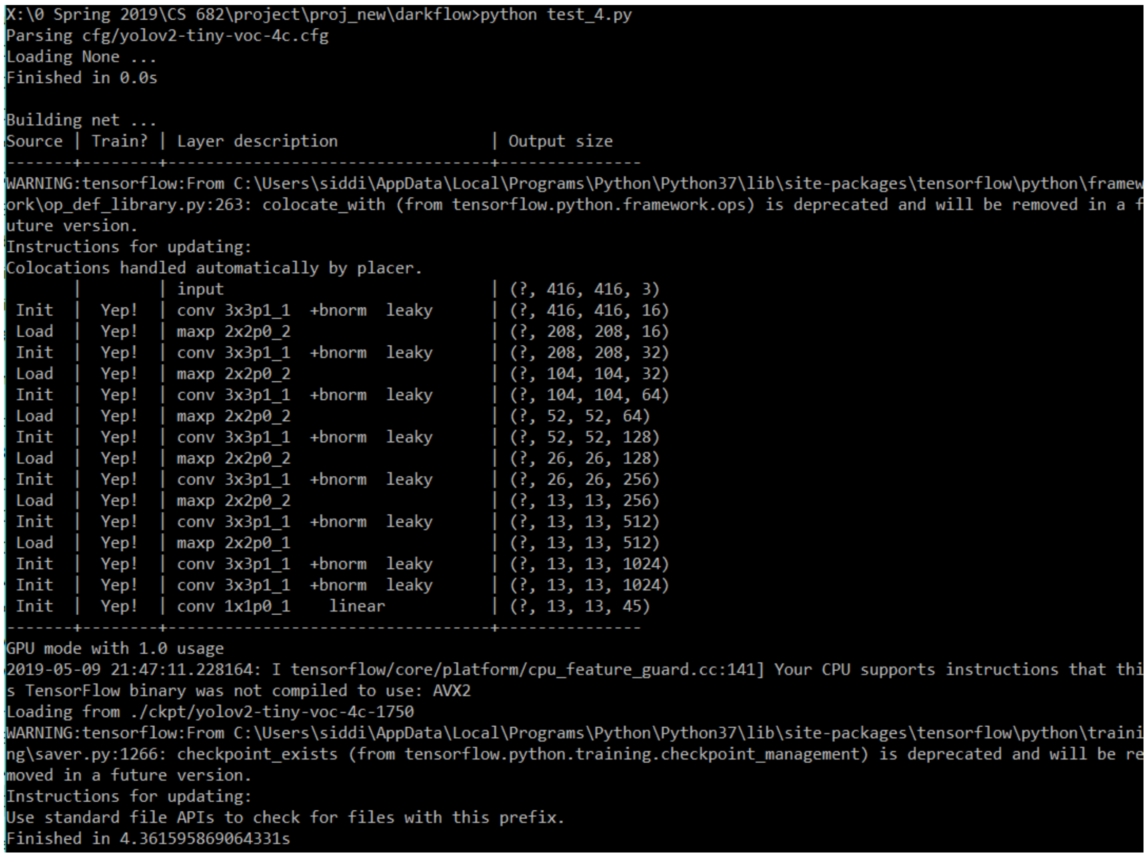
Introduction

- YOLO is a state of the art real time object detection algorithm. It detects objects in real time.
- YOLO has 24 convolutional layers followed by 2 fully connected layers.
- For this project we are using a version of YOLOv2 called Tiny YOLO.
- Tiny YOLO has 9 convolutional layers. It is much faster compared to normal YOLO model but less accurate comparatively.
- As we did not have enough images of each of the classes of emergency vehicles, we used the concept of transfer learning.
- We trained out model based on the weights of already trained model. We have used the weights trained by tiny yolo model on the VOC dataset.
- We used YOLO model from **darkflow** repo. YOLO has been originally written on C language. We found a repository on GitHub which converted the YOLO data into python.

Training

- The Training set consists of image files and annotations files.
- The annotation is an xml file which consists of image name (same as the file name), the height and width of the image, the class of the image, the height and width of the bounding box.
- We had to create all the annotations ourselves as we did not find any dataset which had said annotations for different emergency classes.
- We tried to automate the process as much as we can. We draw bounding boxes on the images using matplot library and saving it onto an xml file directly through a python script.
- We trained the model on 4 different classes:
    - Police: 1032 images.
    - Ambulance: 694 images.
    - Fire Truck: 1001 images.
    - Non-Emergency: 1010 images.

Architecture of Tiny YOLO:

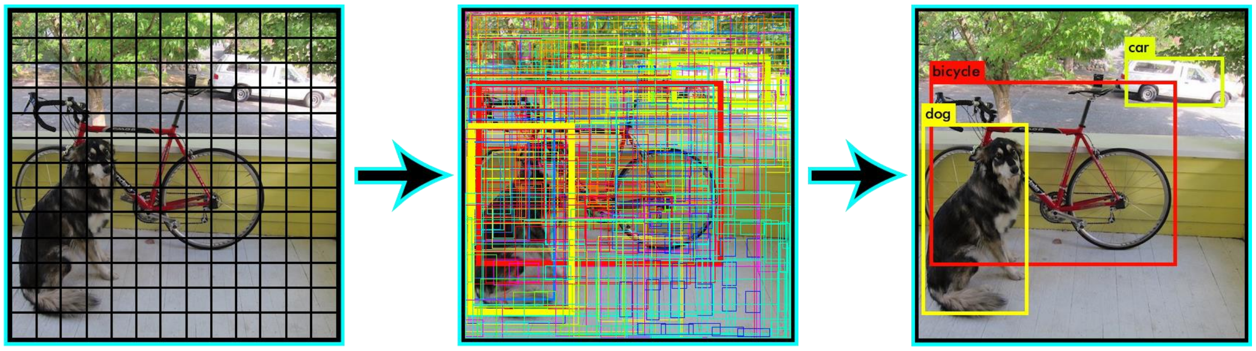


```
X:\0 Spring 2019\CS 682\project\proj_new\darkflow>python test_4.py
Parsing cfg/yolov2-tiny-voc-4c.cfg
Loading None ...
Finished in 0.0s

Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
WARNING:tensorflow:From C:\Users\siddi\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\framew
ork\op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a f
uture version.
Instructions for updating:
Colocations handled automatically by placer.
       |        | input                            | (?, 416, 416, 3)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 416, 416, 16)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 208, 208, 16)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 208, 208, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 104, 104, 32)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 104, 104, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 52, 52, 64)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 52, 52, 128)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 26, 26, 128)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 26, 26, 256)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 13, 13, 256)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 13, 13, 512)
 Load  |  Yep!  | maxp 2x2p0_1                     | (?, 13, 13, 512)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 13, 13, 1024)
 Init  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 13, 13, 1024)
 Init  |  Yep!  | conv 1x1p0_1    linear           | (?, 13, 13, 45)
-------+--------+----------------------------------+---------------
GPU mode with 1.0 usage
2019-05-09 21:47:11.228164: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
Loading from ./ckpt/yolov2-tiny-voc-4c-1750
WARNING:tensorflow:From C:\Users\siddi\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\traini
ng\saver.py:1266: checkpoint_exists (from tensorflow.python.training.checkpoint_management) is deprecated and will be re
moved in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
Finished in 4.361595869064331s
```

YOLO Procedure

- For our discussion, we crop our original photo. YOLO divides the input image into an **S**×**S** grid. Each grid cell predicts only **one** object. Each grid cell predicts a fixed number of boundary boxes.
- 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.



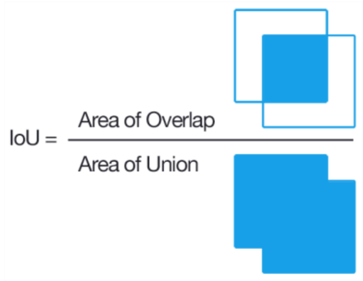
- Each of the grid has the following information(vector):

$$Y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Pc is the probability of whether there is an interesting thing in the background.
bx and by is the center point of the image.
bh and bw is the height and width of the bounding box.
c1, c2 and c3 are the classes.
Of course, all the values will be filled with unknown if Pc itself is zero.

Intersection over Union

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be responsible for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth.



Loss Function

- YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function comprises of:
    - the classification loss.
    - the localization loss (errors between the predicted boundary box and the ground truth).
    - the confidence loss (the objectness of the box).

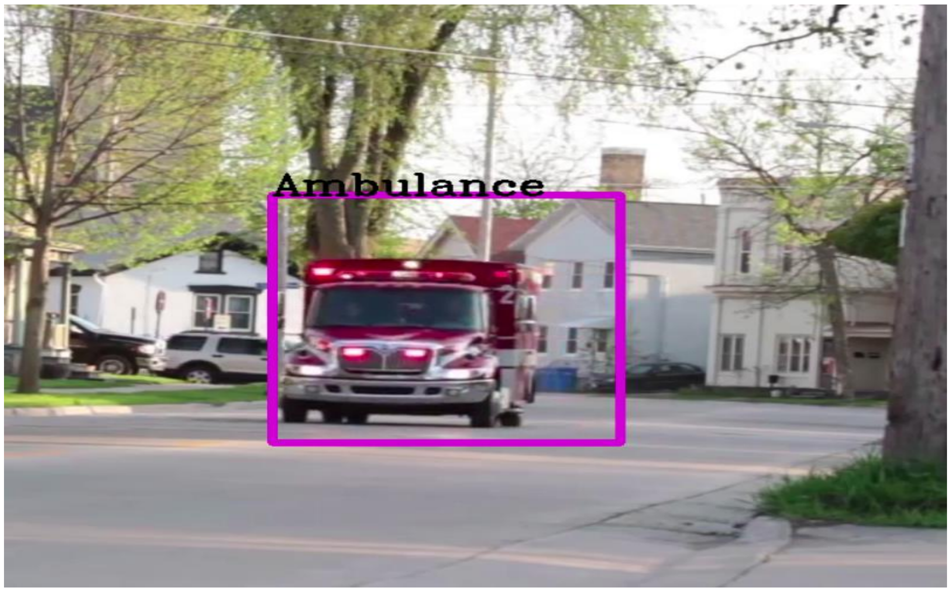
Non Maximal Suppression

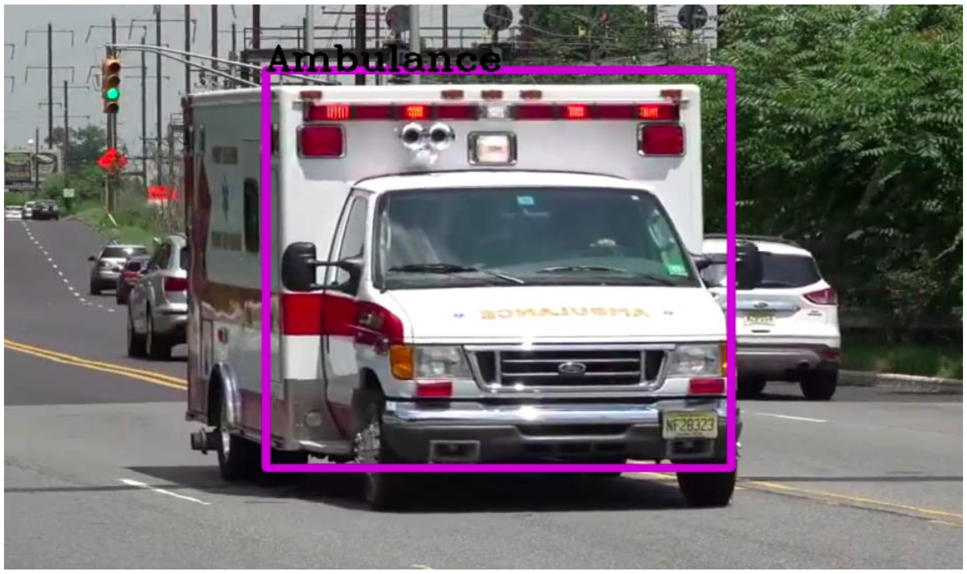
- YOLO can make duplicate detections for the same object. To fix this, YOLO applies non-maximal suppression to remove duplicates with lower confidence.
- Here is the one of the possible non-maximal suppression implementation:
- While there are remaining boxes:
    1. Pick the box with the highest confidence score.
    2. Discard any remaining boxes with IoU>=0.5 with the box output in the previous step.

Results and Discussion

Ambulance Detection

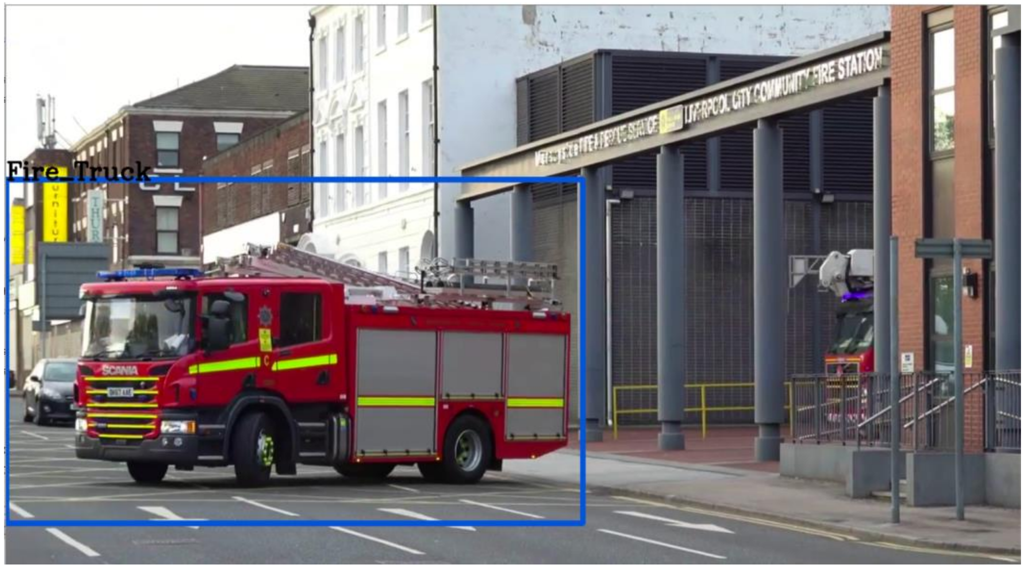
Our model detects the ambulance object in a video frame based on the height and width measurements as well as the distinct red color on the body of the vehicles.Object detection is based on the trained data set of 694 ambulance vehicle images in different angles to obtain a accuracy of 40%. We also successfully demonstrated drawing boxes encompassing the object.

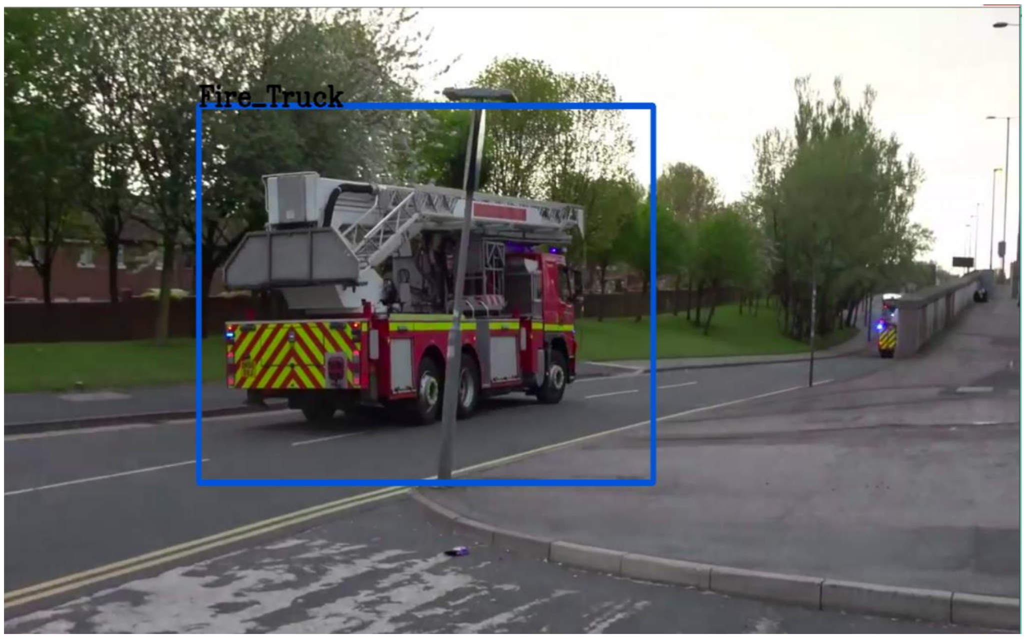




Fire Truck Detection

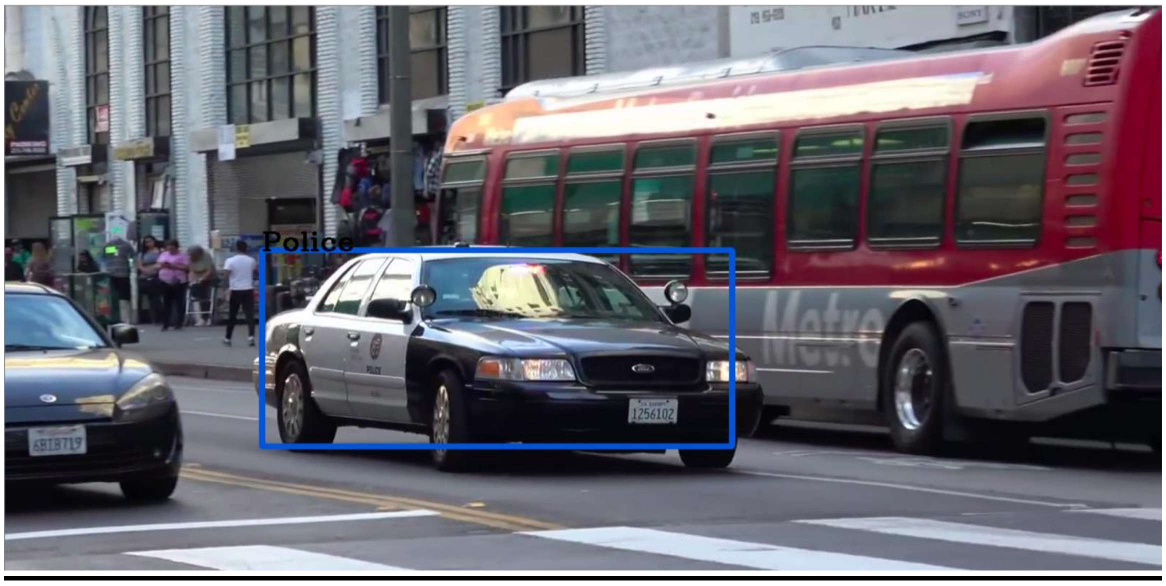
Our model detects the Fire Truck object in a video frame based on the height and width measurements as well as the distinct red color with yellow stripes on the body of the vehicles. Object detection is based on the trained data set of 1001 Fire Truck vehicle images in different angles to obtain a accuracy of 45%. We also successfully demonstrated drawing boxes encompassing the object.

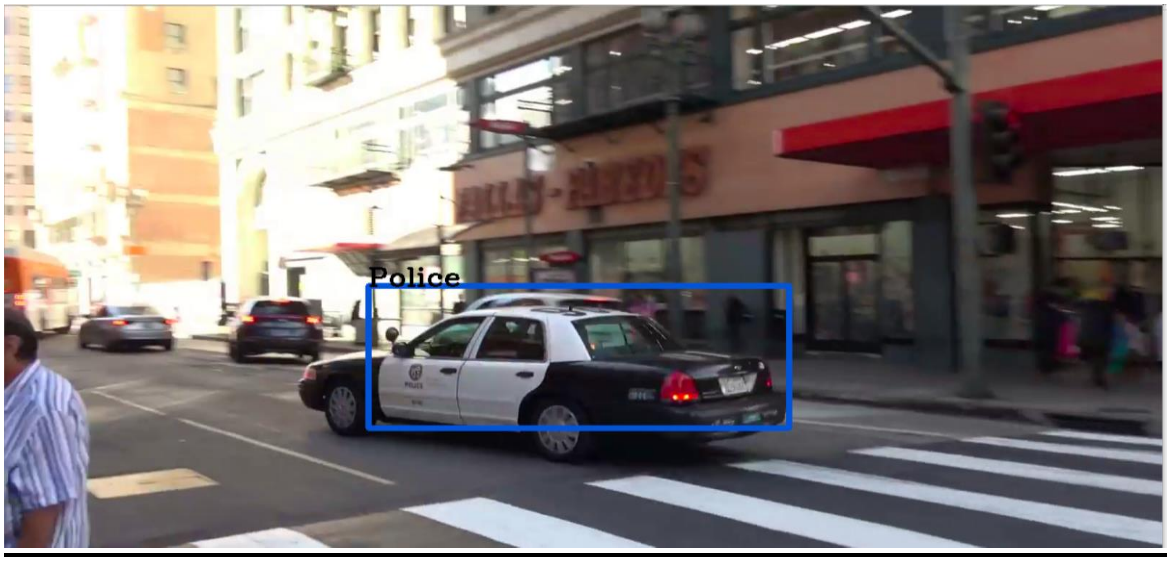




Police Vehicle Detection

Our model detects the Police Vehicle objects in a video frame based on the height and width measurements as well as the distinct black and white color on the body of the vehicles as well as the sides of the vehicle has POLICE written. Object detection is based on the trained data set of 1032 Police vehicle images in different angles to obtain a accuracy of 51%. We also successfully demonstrated drawing boxes encompassing the object.





Conclusion

This model successfully differentiates between emergency and non emergency vehicles, classifying emergency vehicles into 3 classes, Police Vehicles, Ambulances and Fire Trucks.
We have demonstrated our model over images from both day and night times, providing reasonably good results.
A few highlights -

- The model accurately detects all the classes of - Fire Trucks, Police vehicles, Ambulance, with relatively higher confidence values.
- The model works well on detecting emergency vehicles at from various viewpoints - not just the frontal, rear and side views, a critical quality in high traffic zones.
- The model is having difficulties, classifying emergency vehicles during night time, as we are lacking training data for those sort of image distortions.

Also, we believe with little effort - as Tiny YOLO is computationally inexpensive - it can be easily installed on to a Raspberry PI to build a working prototype, could be used to integrate both object detection for self driving capabilities as well for traffic control when there are emergency vehicles around

Team Mates

1. **Siddi Avinash Chenmila:**
   Worked on YOLO model, tuning the parameters , trained the model over various types of YOLO frames works. Wrote all the utility files.
   Things that went well:
   The model came out fine considering that we worked on a small dataset. Of course the concept of transfer learning helped us on that.
   Things that could have been better:
   With a larger dataset we could have achieved a higher confidence score on the bounding boxes and the mislabelling would have been reduced.
   We could have trained on YOLOv2 or YOLOv3 but we did not have enough GPU power to do that. With better models the accuracy would have been better.

2. **Murtuza Shareef:**
   Worked on the VGG16 model - preprocessing to make the data work for VGG16; replaced the last 4 layers to make the model learn the classification between emergency & non-emergency
   Things that went well:
   Got to learn and work on wide variety of concepts using - Transfer learning ; VGG -16 and creating the training data with the bounding boxes as it helped in understanding it better.
   Things that could be better:
   With more time, we could have created more data - to get better accuracy & confidence in the detected images. More importantly, with little effort - as Tiny YOLO is not computationally intensive - it can be easily installed on to a Raspberry PI to build a working prototype, to be used in the traffic monitoring systems

3. **Shubham Pudi:**
   Worked on designing the semantics of metadata to define the annotations which represent contours of bounding boxes for YOLO. Mapping each annotation file to its respective image data file.
   Things that went well:
   Feature Extraction within a bounding box of an image. Deeper understanding on the Convolutional Neural Networks and the various architectures working on it.

   Things that could be better:

Training our model to detect emergency vehicles from other countries including Asia and Europe and also inclusion of other classes of Emergency Vehicles as required.

4. **Sunchit Sehgal:**

   Worked on the VGG16 model including preprocessing of the data. Also aided in designing the Metadata semantics for feature extraction. Tested the trained YOLO model on several videos and looked for anomalies and look for reasons resulting to such anomalies.

   Things that went well:

   Got to learn a variety of concepts regarding CNN and particularly in regards to VGG16, a deeper understanding as to how feature extraction must take place.

   Things that could be better:

   A more efficient way to implement the idea into self-driving cars, with real-time detection of Emergency vehicles and working on ways to classify Emergency vehicles efficiently during nighttime.

References

- https://towardsdatascience.com/transfer-learning-946518f95666
- https://www.learnopencv.com/tag/transfer-learning/
- https://stackoverflow.com/questions/43292549/will-yolo-anyhow-perform-differently-from-vgg-16-will-using-it-for-image-classi
- https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
- Darkflow: https://github.com/thtrieu/darkflow
- Real-time Object Detection with YOLO: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088