Object Detection in an Urban Environment

Object detection is an essential component of many self-driving car systems due to its ability to accurately identify and localize objects in an image or video. In this project, we are attempting to develop an AI-based system that is capable of object detection and recognition in urban environment using deep neural networks. We hope to create an efficient model that is able to recognize objects quickly and accurately, allowing self-driving car systems to make well-informed decisions.

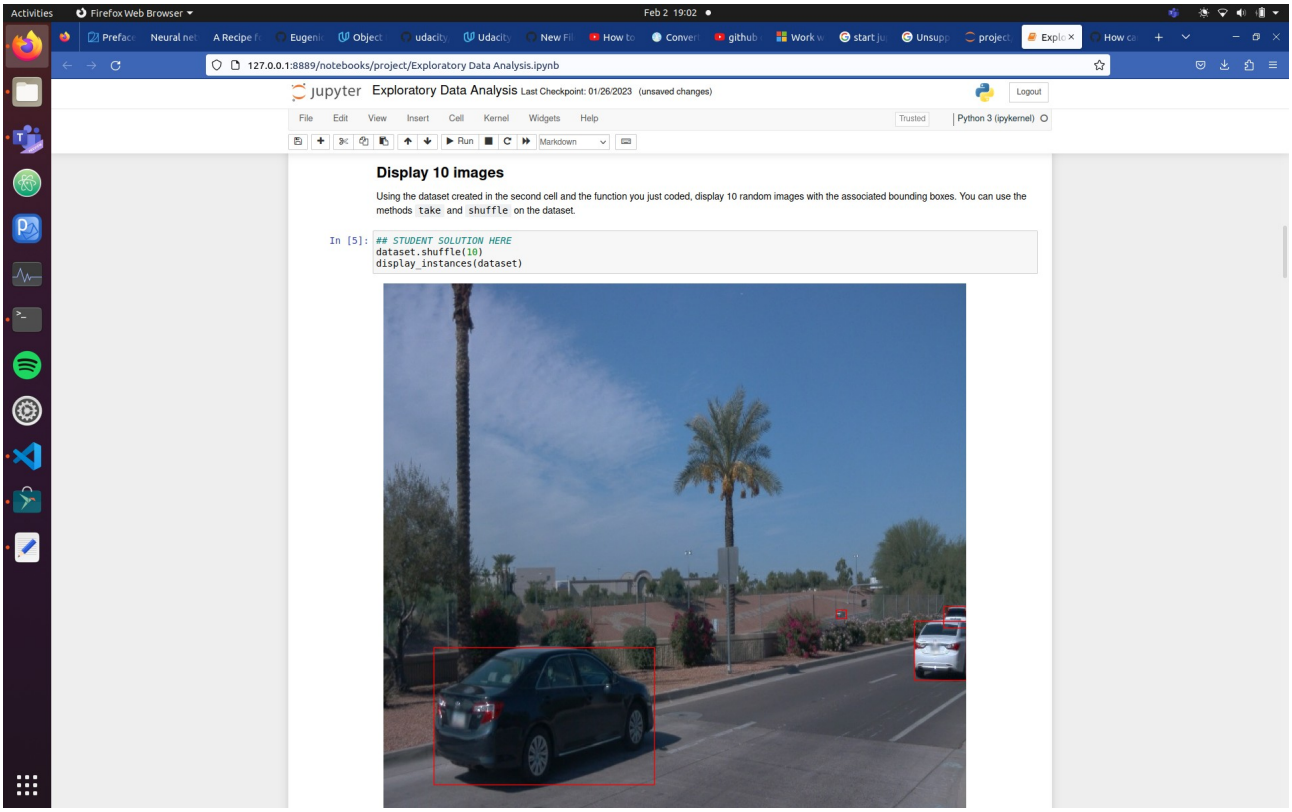
All the code used for this project can be found in this github repository (the dataset used has not been included for obvious file size reasons, but can be downloaded from the waymo dataset, as it is described in the ReadMe file): https://github.com/EugenioCollado/nd013-c1-vision-starter-main

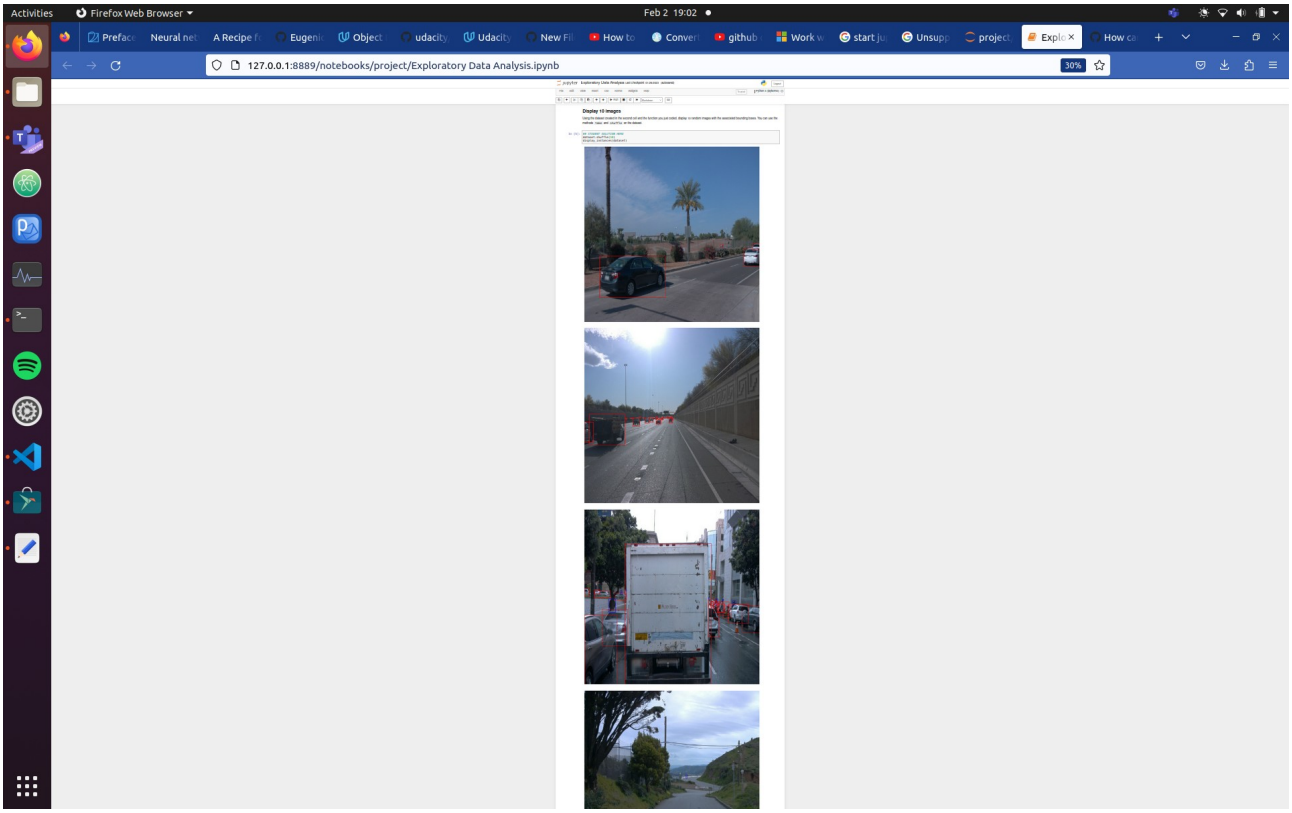
Besides in the ReadMe file are the instructions for setting up the docker container as well as to reproduce the steps followed in this project.

Exploratory Data Analysis (EDA)

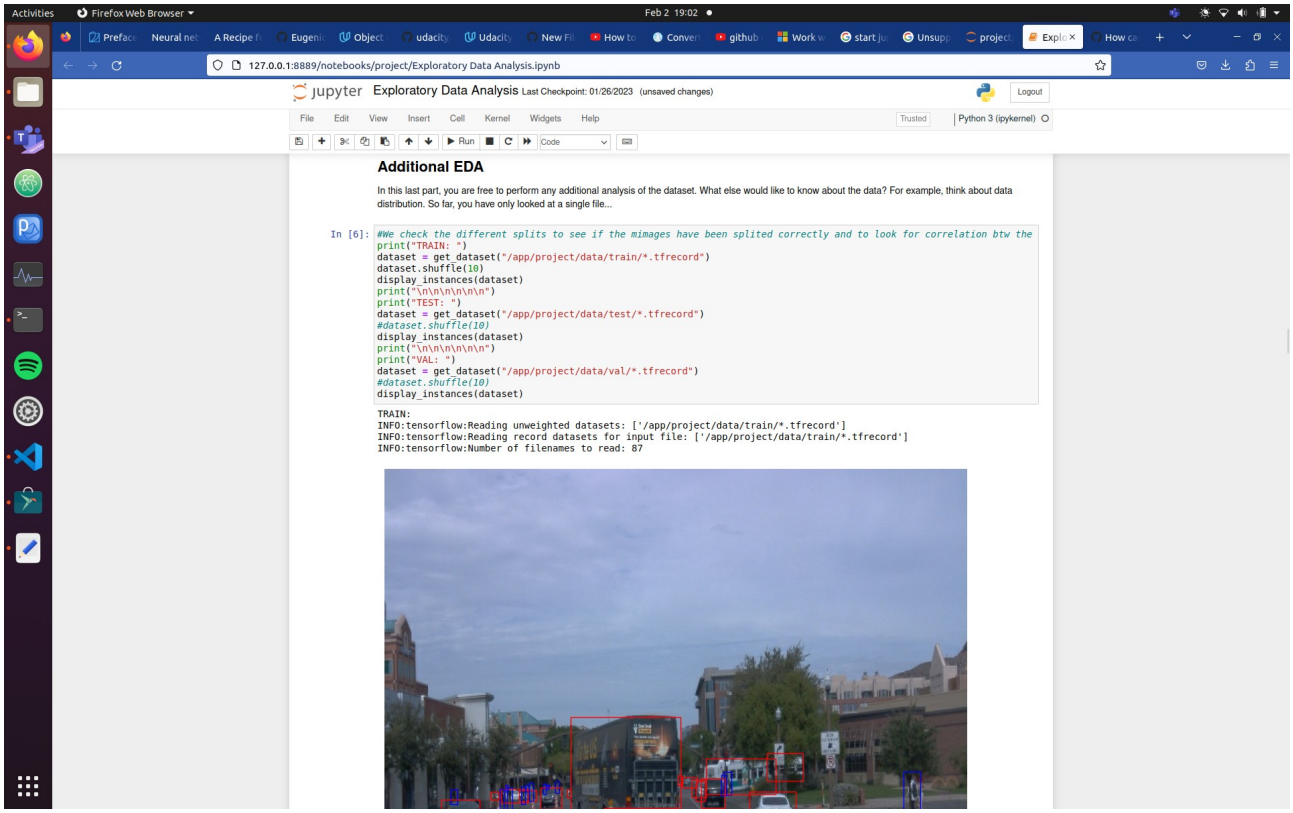
The EDA is necessary because by understanding the data, it is possible to identify any potential issues, such as imbalanced classes, which could affect the accuracy of the model. Additionally, it is also possible to identify any data augmentation techniques that may be necessary to ensure the model is able to generalize well to new data.

The code implemented for the EDA is written in the Jupyter Notebook: “ExploratoryDataAnalysys.ipynb”, and offers the outputs shown in the images below (every class is matched with a different colour):





Once the EDA was completed and the training dataset had been reviewed, as part of the additional EDA the eval and test dataset was also displayed searching for possible correlations in the environmental context of the different datasets due to a bad splitting of the data:



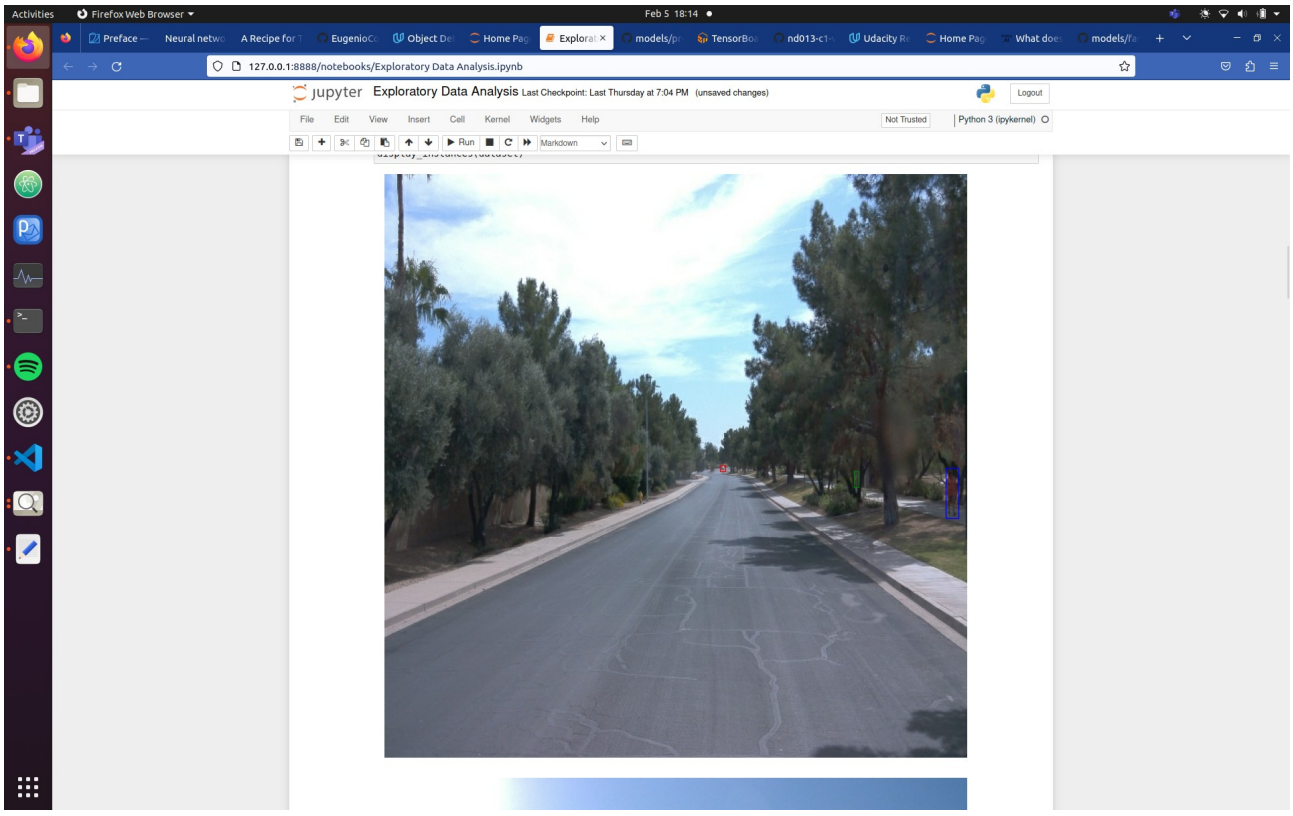
g.

The main conclussions extracted of the EDA are:

- Correct homogeneous distribution of the images between the different datasets. This has been guaranteed by randomizing .the tfrecords spitted in the different datasets:

```
def split(source, destination):
    """
    Create three splits from the processed records. The files should be moved to new folders in the
    same directory. This folder should be named train, val and test.

    args:
        - source [str]: source data directory, contains the processed tf records
        - destination [str]: destination data directory, contains 3 sub folders: train / val / test
    """
    list_files = os.listdir(source)

    random_list = random.sample(list_files, 100)
    for i , file in enumerate(random_list):
        if(i < 87):
            shutil.copy(source + "/" + file, destination + "/train")
        elif(i < 97):
            shutil.copy(source + "/" + file, destination + "/val")
        else:
            shutil.copy(source + "/" + file, destination + "/test")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Split data into training / validation / testing')
    parser.add_argument('--source', required=True,
                        help='source data directory')
    parser.add_argument('--destination', required=True,
                        help='destination data directory')
    args = parser.parse_args()

    logger = get_module_logger(__name__)
    logger.info('Creating splits...')
    split(args.source, args.destination)
```

- The dataset has a good representation of cars, but a poor representation of pedestrians and bicycles, which will lead to a poor classification of those two object classes (Besides, as shown in the image below, usually when a pedestrian or a bycicle appears it is in a small bad placing part of the image).



- Something similar happens with the adverse weather conditions such as rain or night (we will try to overcome this problem through data augmentation).

Reference experiment

In this experiment we are going to train (fine tuning) the (pretrained) model with the pipeline_new.config without any modification. The results of that training are shown below:

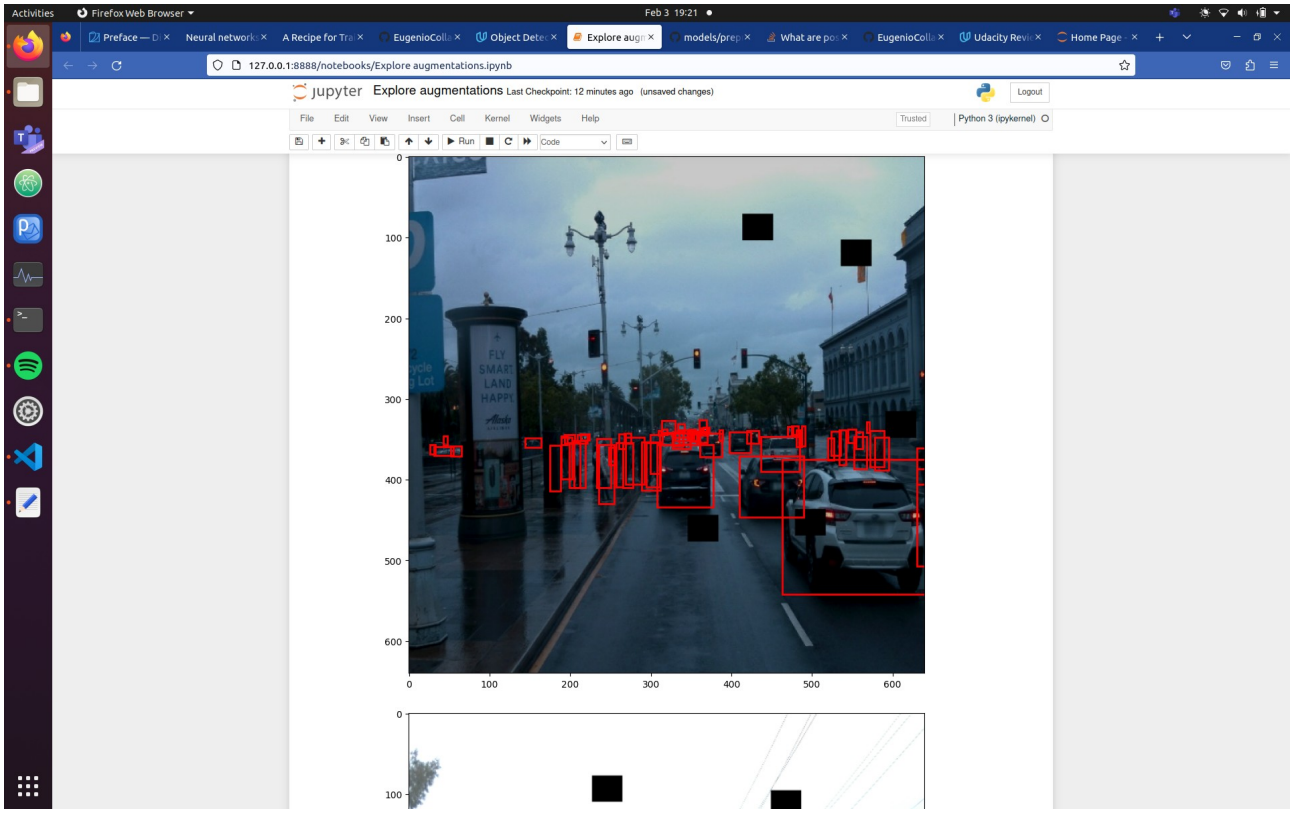
The poor performance and bad metrics shown in the image are understandable given the short dataset (just 100 tfrecords) and the problems annotated before. As this behavior was expected I did not let the training reach the 24K steps as I started to implement augmentations.

Improve on the reference

The augmentations implemented are:

The contrast, saturation, hue and brightness augmentations are related to the adverse weather conditions images. The horizontal flips is related to the small size of the dataset (serves the purpose of “duplicating” the number of different images), as well as the random black patches and random crop (tries to avoid overfitting by not training on the exact same images over epoches).

Here an example of an augmented image:



Results

After the augmentation strategy, this are the metrics of the training, much better than the reference ones:

Finally, to show the good performance of the model, it has been recorded and saved in the repository a video of the model's inference for a tfrecord. In the animation_1.gif can be appreciated the overall good performance of the model detecting cars. In the other hand, in the animation_pedestrian.gif, there are several pedestrians in the video and none of them are detected.