On-Road Obstacle Detection using Machine Learning

Project by:
Darsh Kumar (20BCE0301, VIT Vellore)
Trisha Seal Sharma (20BCE0248, VIT Vellore)

1. Introduction

1.1. Overview

In this project, we aim to train a machine learning model to detect obstacles on the road from images provided by the user. The model is to be contained in a web-app run using Flask framework in python.

1.2 Purpose:

Road obstacle detection is used to improve transportation systems' safety and effectiveness. Potential risks can be detected and reduced by identifying impediments on the road, such as pedestrians, cars, debris, or problems with the road infrastructure. Since it enables autonomous vehicles to navigate and react to obstacles in real-time, reducing accidents and guaranteeing passenger safety, this technology is very important for these vehicles. Road obstacle detection can also help human drivers by alerting them to potential dangers and delivering warnings or alerts, which enables them to take necessary action and avoid collisions. Most autonomous cars employ a variety of sensors, including optical, infrared, and LIDAR sensors, to build depth maps and identify objects from the additional data. Our project aims to reduce the number of sensors required to just an optical sensor using machine learning.

2. Literature Survey

2.1 Existing solutions

2.1.1. Road Obstacles Detection using Convolution Neural Network and Report using IoT

This project's goal is to find negative impediments like road cracks and potholes that are below the surface of the ground. Highway road scenes are captured and tested using the USB camera arrangement. The automatic detection of road cracks features a novel evaluation and comparison mechanism. Convolution neural network classifier was used to process the road crack images that were gathered, and MATLAB was used to simulate the model. By deploying IoT modules, the identified impediments and their locations are transmitted to the roads department.

2.1.2. Detection of road boundaries and obstacles using LIDAR

In this technique, the road surface is extracted using a 2-D Light Detection and Ranging (LIDAR) sensor positioned at a specific pitch angle. Filters analyze the data about the road to determine if the area being scanned is a road or an obstruction. A real-world test vehicle called AKAY01 and simulation are used to apply the suggested methodology.

2.1.3. Stereo vision-based road obstacles detection

This study describes a quick stereo vision-based method for detecting road obstacles. Three primary elements make up the algorithm: road detection, obstacle identification, and obstacle tracking. In order to extract the disparities of the road, a little rectangular shape at the bottom center of the disparity image is used. Using morphological analysis and the Hough transform, the roadsides are found. The segmentation step in the obstacle detection method makes it simple to find the items. The discrete Kalman filter is used to track the obstacles.

2.2. Proposed Solution

In this project, we propose a machine learning model based on ResNet V2 model, with initial weights and architecture taken from TensorFlow Model Garden and re-tuned with a different dataset.

The dataset in question is linked to in Appendix B, and is a COCO format dataset, which consists of a directory filled with image files and a json file that contains the following:

- Directory filled with image files
- JSON manifest file containing the following:
    - List of image files
    - List of detection target categories
    - List of detection targets with bounding box locators

The model trains on the coco dataset and saves its weights after a certain amount of epochs into local directory, which can be retrieved from the web app at runtime to skip training time on each program run.

In order to get predictions on unknown data, the web app contains wrapper functions to convert the individual image to a COCO format dataset with an empty annotations file. The model predicts on the new data through the newly generated dataset. The output is a regular JPEG format image that is shown to the user.

3. Theoretical Analysis

3.1 Block Diagram

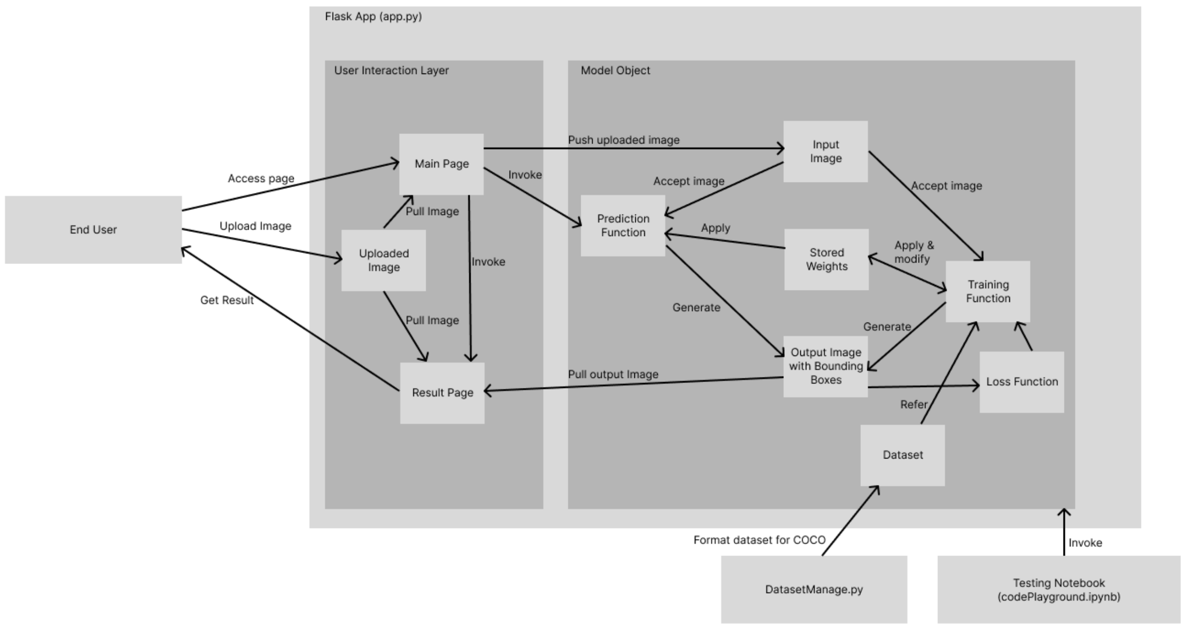


Figure 1: Project Architecture Diagram

3.2. Requirements

3.2.1. Hardware Requirements

- CPU: Intel Core i5 9th Gen (AMD Ryzen 5 3500U equivalent) or higher
- GPU: Nvidia CUDA compatible GPU with 6 GB VRAM minimum
- RAM: 16 GB DDR4 or higher
- Disk Space: 1 GB minimum

3.2.2. Software Requirements

- Windows 7 or higher (without GPU support, refrain from using WSL2 to introduce GPU support as CPU, RAM and storage usage will be steeply increased)
- Ubuntu 18.04 or higher or Latest Arch linux based distribution (with GPU support)
- Python version 3.9
- Nvidia GPU Driver 450 or higher
- CUDA Toolkit 11.8
- CuDNN SDK 8.6.0

3.2.3. Python Dependencies (Install using pip)

- `pprint` : Pretty Printer
- `numpy` : Standard helper library used by TensorFlow
- `matplotlib` : Plotting and visualization library
- `opencv` : Open ComputerVision Library
- `tensorflow` : Main Machine Learning Library used
- `tf-models-official` : Tensorflow Model Garden
- `flask` : Web Application Hosting Library
- `flask-wtf` : Forms support for Flask

3.2.4. Optional Dependencies

- `jupyter` : Optional dependency for testing codePlayground.ipynb

4. Experimental Analysis

4.1. VRAM Utilization Scaling

The amount of GPU Video memory that the model, and subsequently the jupyter kernel, uses increases directly with the input and output image resolutions, size of dataset, model layer complexity, number of trainable parameters as well as whether the model is actively accessing the data or not. With our dataset and model complexity fixed, the main optimization comes from input image resolution, the output image follows the same resolution as the input image. Reducing the input image resolution would reduce information for the model to interpret, while a higher resolution would significantly increase VRAM consumption. The sweet spot for our configuration occurred at 256x256 resolution.

5. Flowchart

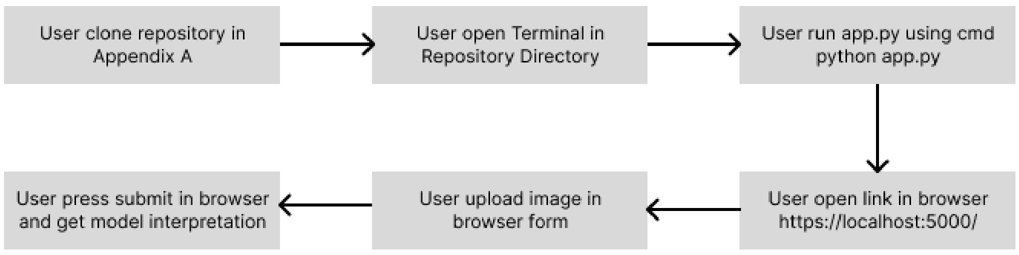
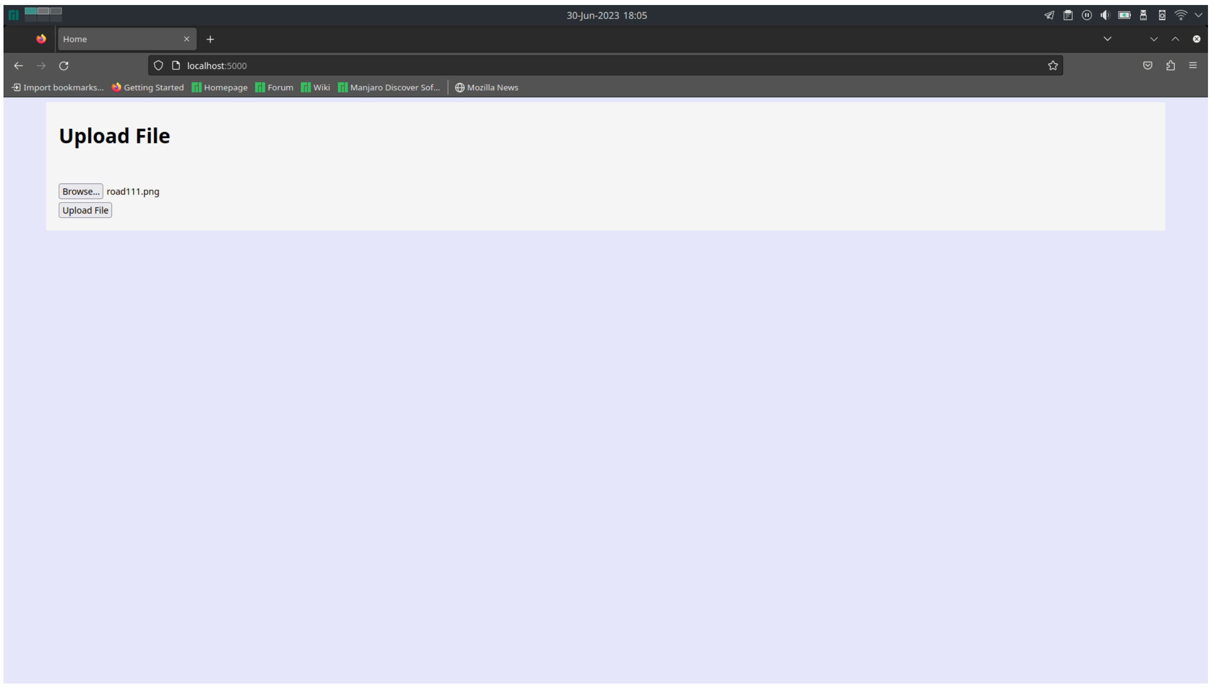
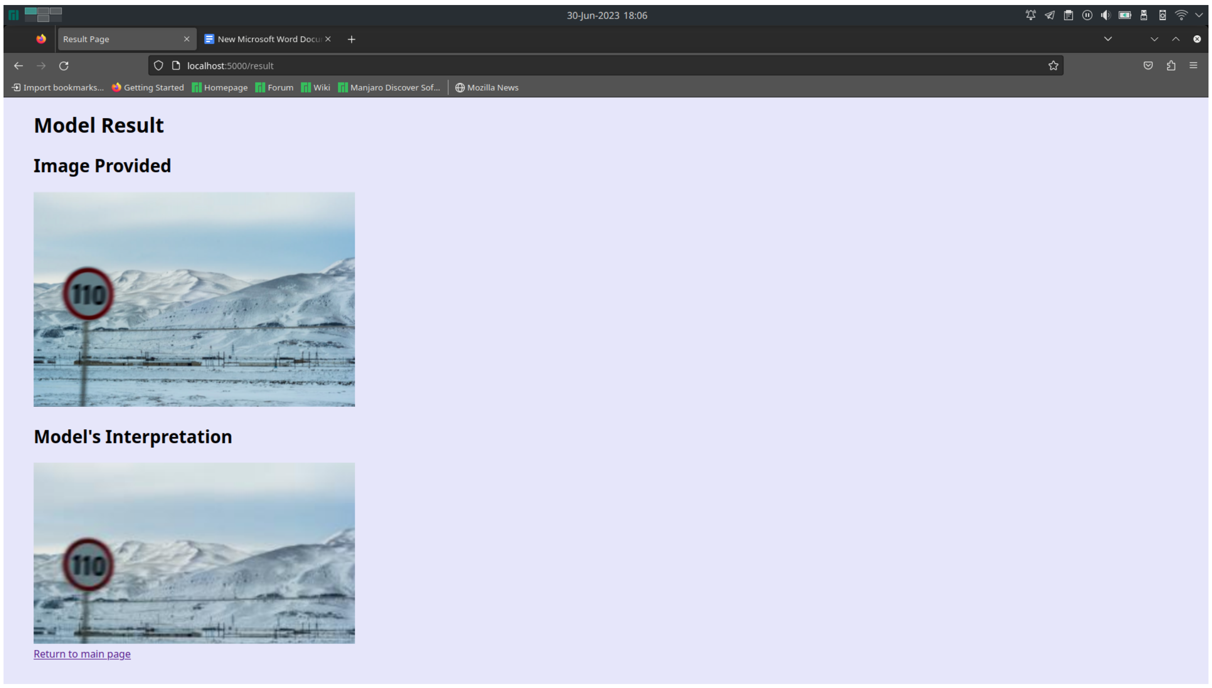

User clone repository in Appendix A
User open Terminal in Repository Directory
User run app.py using cmd python app.py
User open link in browser https://localhost:5000/
User upload image in browser form
User press submit in browser and get model interpretation

Figure 2: Control flow diagram

6. Result

The general design of the web app follows a simplified design philosophy, where the end user just needs to insert the input image and gets the output image after pressing ”Submit.“ The following two figures demonstrate the UI design using a placeholder image fed to an untrained model.





Model Result

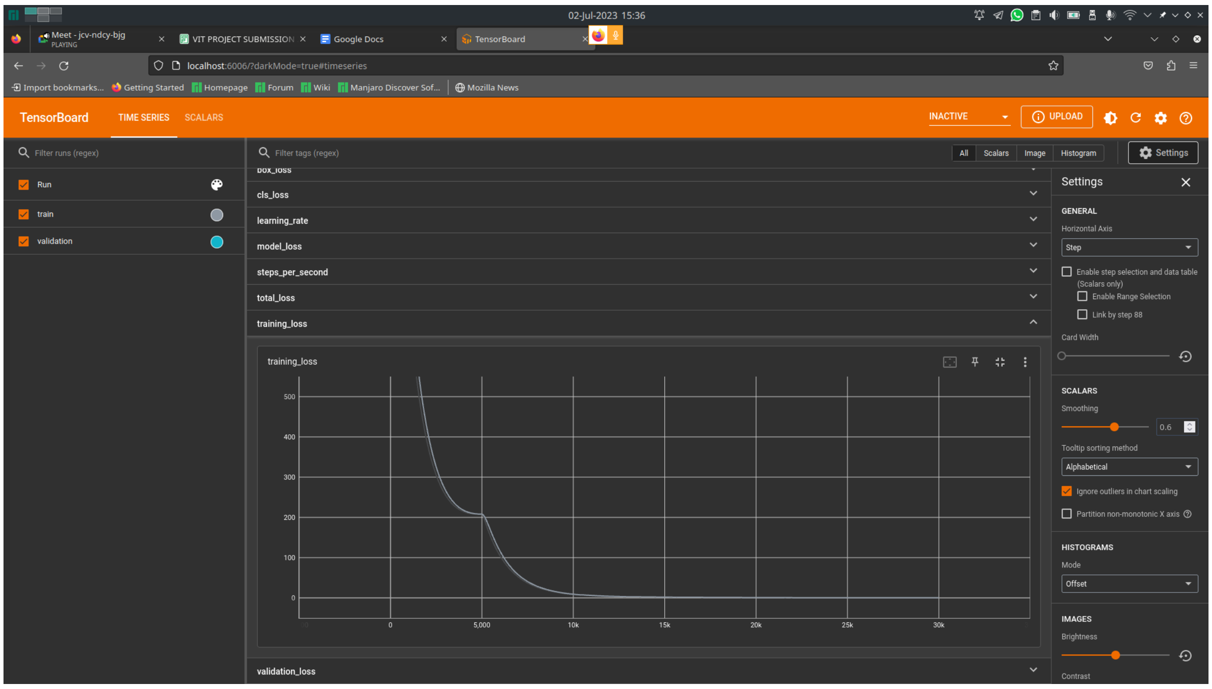
Image Provided

Model's Interpretation

Return to main page

6.2. Training Statistics

The model was trained up to 30,000 training steps, consisting of 110 training loops and 22 validation loops. The training loss trends are displayed in the following graph figure:



Run
train
validation

box_loss
cls_loss
learning_rate
model_loss
steps_per_second
total_loss
training_loss
validation_loss

Settings

GENERAL

Horizontal Axis: Step

☐ Enable step selection and data table (Scalars only)

☐ Enable Range Selection

☐ Link by step 88

Card Width

SCALARS

Smoothing: 0.6

Tooltip sorting method: Alphabetical

☑ Ignore outliers in chart scaling

☐ Partition non-monotonic X axis

HISTOGRAMS

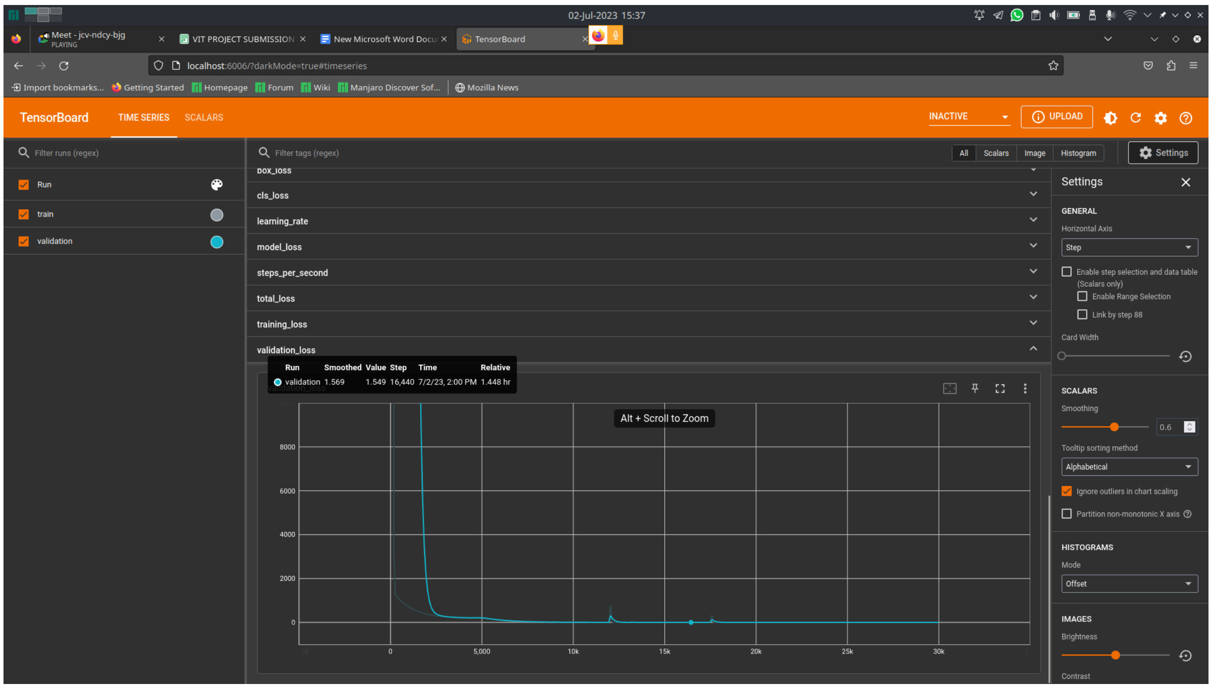
Mode: Offset

IMAGES

Brightness

Contrast

The validation loss is displayed in the figure below:



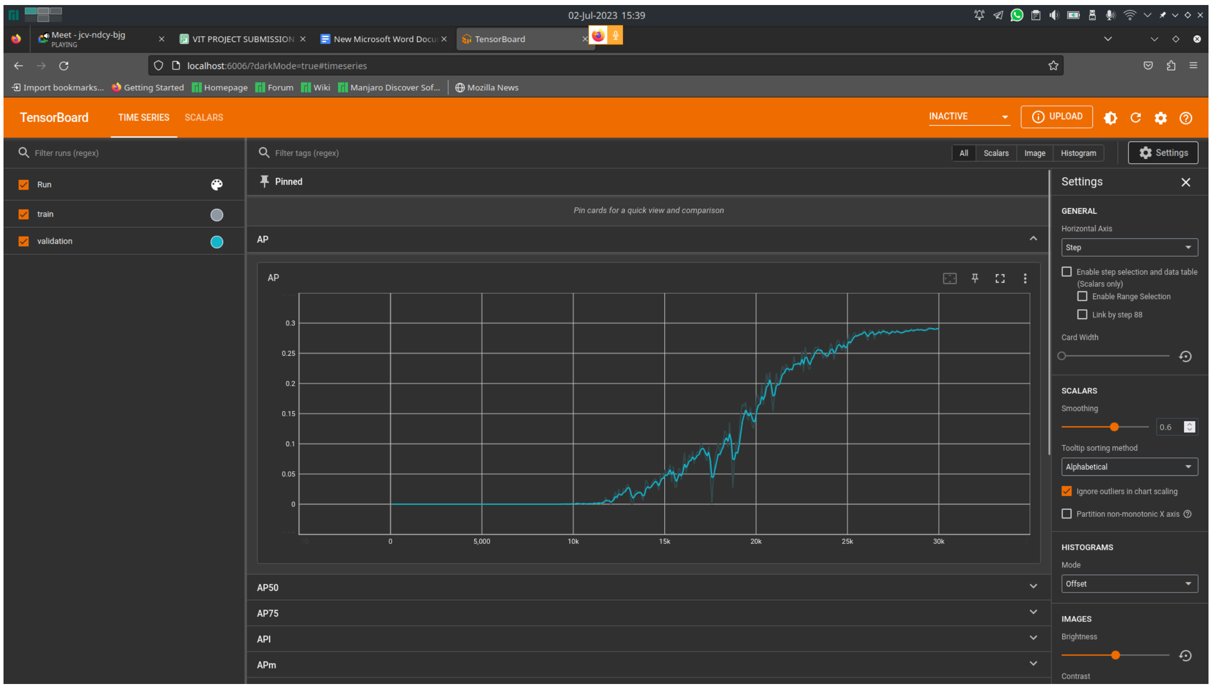
Run	Smoothed	Value	Step	Time	Relative
validation	1.569	1.549	16,440	7/2/23, 2:00 PM	1.448 hr

Runs: ☑ Run, ☑ train, ☑ validation

Settings
GENERAL
Horizontal Axis: Step
☐ Enable step selection and data table (Scalars only)
☐ Enable Range Selection
☐ Link by step 88
Card Width
SCALARS
Smoothing: 0.6
Tooltip sorting method: Alphabetical
☑ Ignore outliers in chart scaling
☐ Partition non-monotonic X axis
HISTOGRAMS
Mode: Offset
IMAGES
Brightness
Contrast

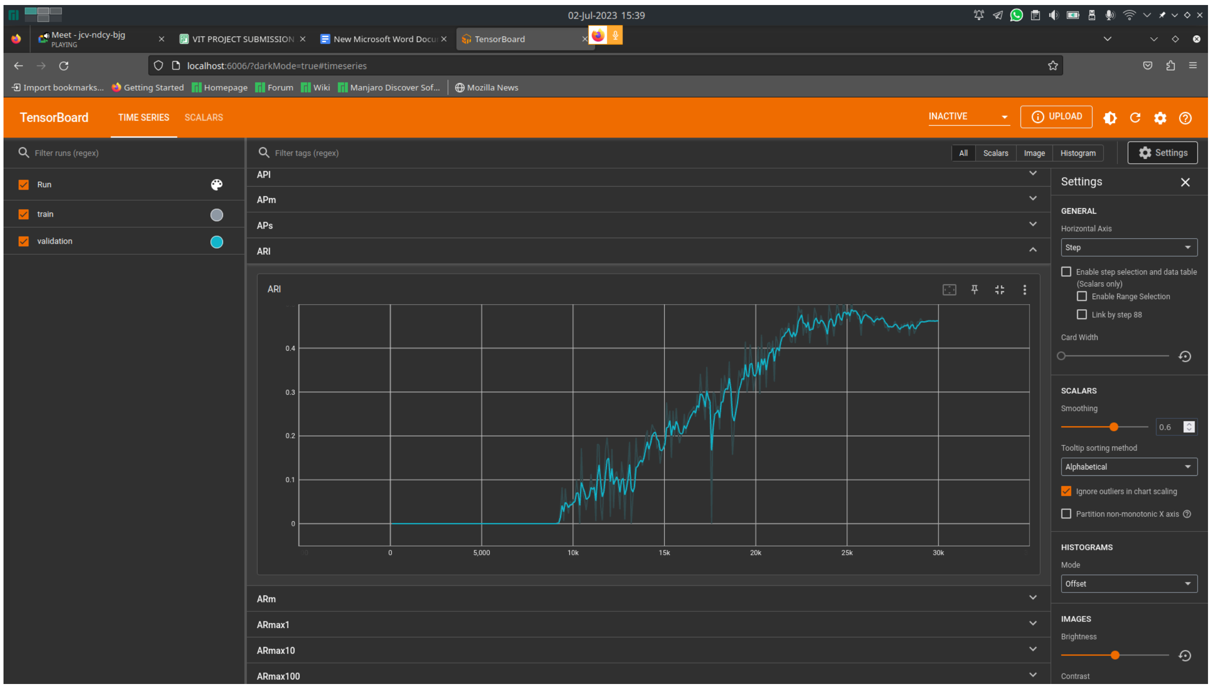
Both training and validation losses follow a similar trend of reduction over epochs.

For object detection tasks, model performance is measured using Average Precision (AP) and Average Recall (AR) measurements. They are shown in the following screenshots:



Runs: ☑ Run, ☑ train, ☑ validation

Settings
GENERAL
Horizontal Axis: Step
☐ Enable step selection and data table (Scalars only)
☐ Enable Range Selection
☐ Link by step 88
Card Width
SCALARS
Smoothing: 0.6
Tooltip sorting method: Alphabetical
☑ Ignore outliers in chart scaling
☐ Partition non-monotonic X axis
HISTOGRAMS
Mode: Offset
IMAGES
Brightness
Contrast



Settings

GENERAL

Horizontal Axis: Step

☐ Enable step selection and data table (Scalars only)

☐ Enable Range Selection

☐ Link by step 88

Card Width

SCALARS

Smoothing: 0.6

Tooltip sorting method: Alphabetical

☑ Ignore outliers in chart scaling

☐ Partition non-monotonic X axis

HISTOGRAMS

Mode: Offset

IMAGES

Brightness

Contrast

It is interesting to note that until around 12,000 training steps, the model is not able to classify anything, resulting in net zero AP and AR measurements

6.2. Testing with New Data

The model's performance with new images is not up to par, many factors can be attributed to this observation, including but not limited to:

- Model complexity
- Dataset balancedness
- Model weights during initialization
- Learning rate decay trends
- Available computing resources
- Input image dimensions

On average, the model gave 40-50% accuracy when validated against new data.

7. Advantages and Disadvantages

The model used for this project is a ResNet derived model. Therefore as a major advantage, it is highly versatile to be expanded in the types and subtypes of obstacles. For example: If a road sign is detected, the type of road sign can be determined by the model.

There are two main disadvantages to our approach to solving the problem. First, the computing resources required to train and run the model in the flask environment is very high. For reference, the model was trained on an Nvidia RTX 3060 mobile GPU with 6 Gigabytes of VRAM and 120 Tensor cores, and the VRAM utilization was peaking at 5.5 Gigabytes with 95% compute resource utilization during training. This disadvantage is one that is easily overcome with time as computing hardware becomes faster and widely available.

Secondly, the number of epochs required to tune the model is very high, and can be overcome by tuning the model training parameters such as learning rate falloff, etc.

8. Applications

For autonomous vehicles to navigate safely and make judgements in real time, road barrier detection is essential. cars can detect and react to impediments including people, bicycles, animals, rubbish, and other cars with the aid of this technology. Autonomous vehicles can change their speed and trajectory and take the required precautions to prevent crashes by detecting impediments in their path.

For systems of traffic management and monitoring, road obstacle detection is useful. Authorities can respond swiftly and send help to clear the road by seeing obstructions like stuck cars, accidents, or fallen objects. By recommending alternate routes and giving drivers timely information, real-time obstacle identification can assist optimize traffic flow and minimize congestion.

9. Conclusion:

In conclusion, obstacle detection plays a crucial role in various fields and applications, ranging from autonomous vehicles and robotics to industrial automation and safety systems. The primary goal of obstacle detection is to enhance the safety, efficiency, and reliability of systems that interact with the physical world. By accurately detecting and recognizing obstacles in real-time, systems can make informed decisions, avoid collisions, and navigate complex environments. Our project shows high promise in processing images and image series to detect obstacles. Clearly, more time and work can be put into it to have better detection using a more balanced dataset and/or more complex model systems. This project can be regarded as an explorative study into emerging and potent technologies.

10. Future Scope:

This project has been made with modularity in mind, and in theory, almost all aspects can be expanded. Let us take a few modules as example:

10.1.1. Input Dataset Packing

Testing of the model is performed using matrices containing one single image. By using TensorFlow and Keras models, we can allow for multiple images to be fed into the model to be interpreted as a batch. This was slightly explored in videoExtract.py in the source code (see Appendix A), but later rejected due to hardware limitations.

10.1.2. Video Prediction and Output

The testing images for the model are individual frames extracted from a single video file. In principle, inverse operation can be performed on a series of output images for the frames, using the previously mentioned Input images packing potential. At the end user, video input can be taken, and the outputs stitched into a video file can be shown as result.

11. Bibliography

- T. Rajendran, M. I. N, J. K and A. K. D, "Road Obstacles Detection using Convolution Neural Network and Report using IoT," 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2022, pp. 22-26, doi: 10.1109/ICSSIT53264.2022.9716337.
- O. Yalcin, A. Sayar, O. F. Arar, S. Apinar and S. Kosunalp, "Detection of road boundaries and obstacles using LIDAR," 2014 6th Computer Science and Electronic Engineering Conference (CEEC), Colchester, UK, 2014, pp. 6-10, doi: 10.1109/CEEC.2014.6958546.
- Z. Khalid, E. A. Mohamed and M. Abdenbi, "Stereo vision-based road obstacles detection," 2013 8th International Conference on Intelligent Systems: Theories and Applications (SITA), Rabat, Morocco, 2013, pp. 1-6, doi: 10.1109/SITA.2013.6560817.

Appendix

A. Source Code

https://github.com/DarshK35/obstacle-detect-ml

B. Dataset

https://www.kaggle.com/datasets/weichiyu/road-coco