Vision AI on Nvidia Jetson Nano: Sensor Fusion

Project Members:

Riya Alias : 1119262
Nikita Anant Patil : 1119255
Sreekuttan Sreedharan Thampi : 1123158
Amrutha Venugopal : 1119243

Overview

1. Introduction
2. Flow Diagram
3. Vision AI and Sensor Fusion
4. Nvidia Jetson Nano
5. Edge Impulse SDK
6. Building a dataset in Edge Impulse
7. Designing an Impulse and Validating the model
8. Testing and drawbacks of Edge Impulse
9. YoloV5 model
10. YoloV5 model testing and drawbacks
11. YoloV7 model
12. YoloV7 model testing and results
13. TensorRT model
14. Results
15. Parallel Processing using two cameras
16. Conclusions
17. Future Improvements

Introduction

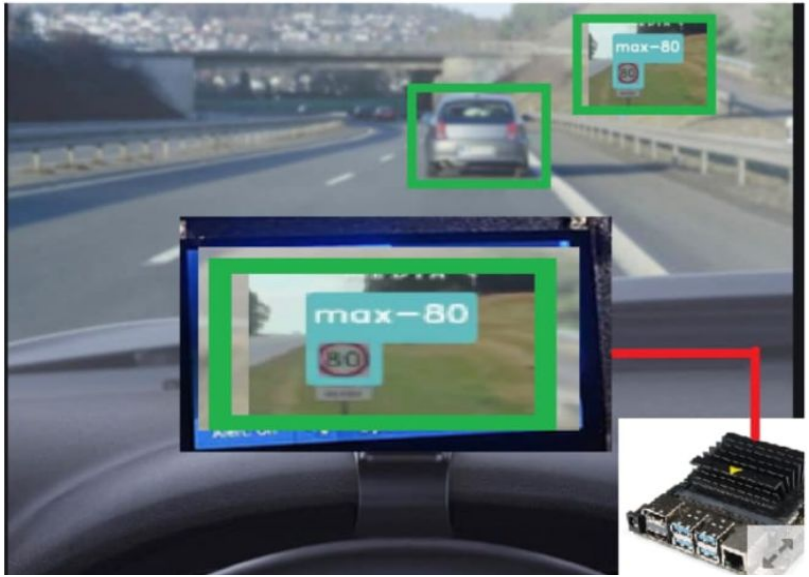
- Develop a comprehensive and robust Vision AI system on the NVIDIA Jetson Nano
- Sensor fusion techniques to enhance drowsiness detection and traffic sign detection.
- Drowsiness Detection :
    - Implement a drowsiness detection algorithm that analyzes facial cues
    - Optimize the algorithm for real-time processing using the Jetson Nano's GPU acceleration.

- Traffic Sign Detection:
    - Help drivers by providing real-time information about traffic signals and providing timely alerts or interventions.
    - Implement a mechanism to recognize and react to different types of traffic signs, such as speed limits, stop signs, and yield signs.

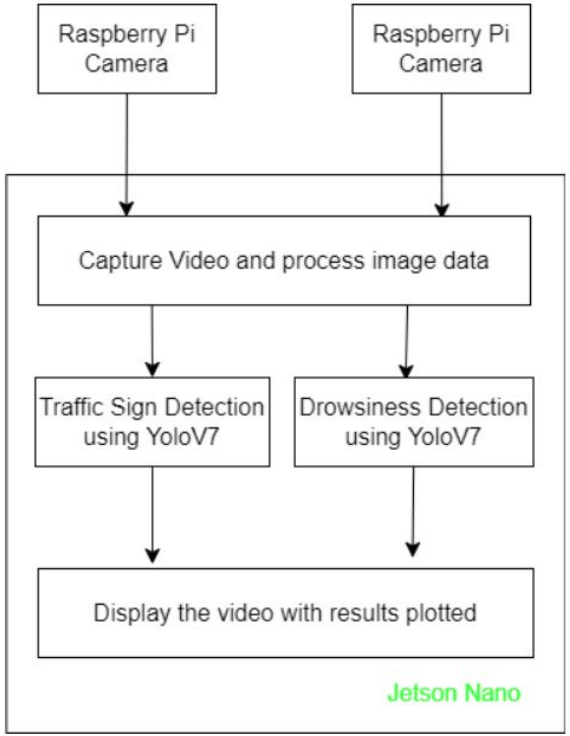
- Aims to create an innovative solution that demonstrates the capabilities of sensor fusion in enhancing safety and awareness in driving scenarios.
- The integration of drowsiness detection and traffic sign detection through sensor fusion showcases the power of AI technologies and their potential to positively impact road safety.

Driver Monitoring system

Driver Traffic Signal Assistance



Flow Diagram



Vision AI and Sensor Fusion

Combining Vision AI and sensor data is of paramount importance in building intelligent systems that can accurately perceive and understand their environment. This integration offers several key advantages that significantly enhance the capabilities and robustness of various applications.

Vision AI :

- Involves computer vision techniques to enable machines to understand visual information from the world.
- This includes tasks such as image recognition, object detection, semantic segmentation, and even more complex tasks like facial recognition.
- Vision AI leverages deep learning algorithms to analyze visual data and extract meaningful insights

Sensor Fusion:

- Sensor Fusion is the process of integrating data from multiple sensors to create a more accurate and complete representation of the environment.
- Sensors can include cameras, lidar, radar, GPS, IMUs (Inertial Measurement Units), and more.
- By combining data from different sensors, a system gains a better understanding of its surroundings, leading to improved decision-making and situational awareness.

Nvidia Jetson Nano

- It is a powerful edge computing platform designed to bring the capabilities of artificial intelligence and deep learning to resource-constrained devices at the edge of networks.
- It provides the computational power necessary for running deep learning models and performing complex AI tasks directly on edge devices.
- Nvidia Jetson Nano is exceptionally well-suited for Vision AI and Sensor Fusion applications due to its powerful hardware, dedicated GPU, and AI-focused software support.

Key Features:

- GPU Acceleration
- Quad-Core CPU
- Memory and Storage
- Connectivity
- AI Software Support

Hardware

- Uses Nvidia Jetson Nano Kit to develop a system that can perform real time image processing and object detection using multiple cameras.
- **Jetson Nano:**
  - small, powerful computer
  - Allows to run multiple neural networks in parallel
  - Applications: image classification, object detection, segmentation, and speech processing.
  - GPU : NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores
  - CPU : Quad-core ARM® Cortex®-A57 MPCore processor
  - Memory : 4 GB 64-bit LPDDR4

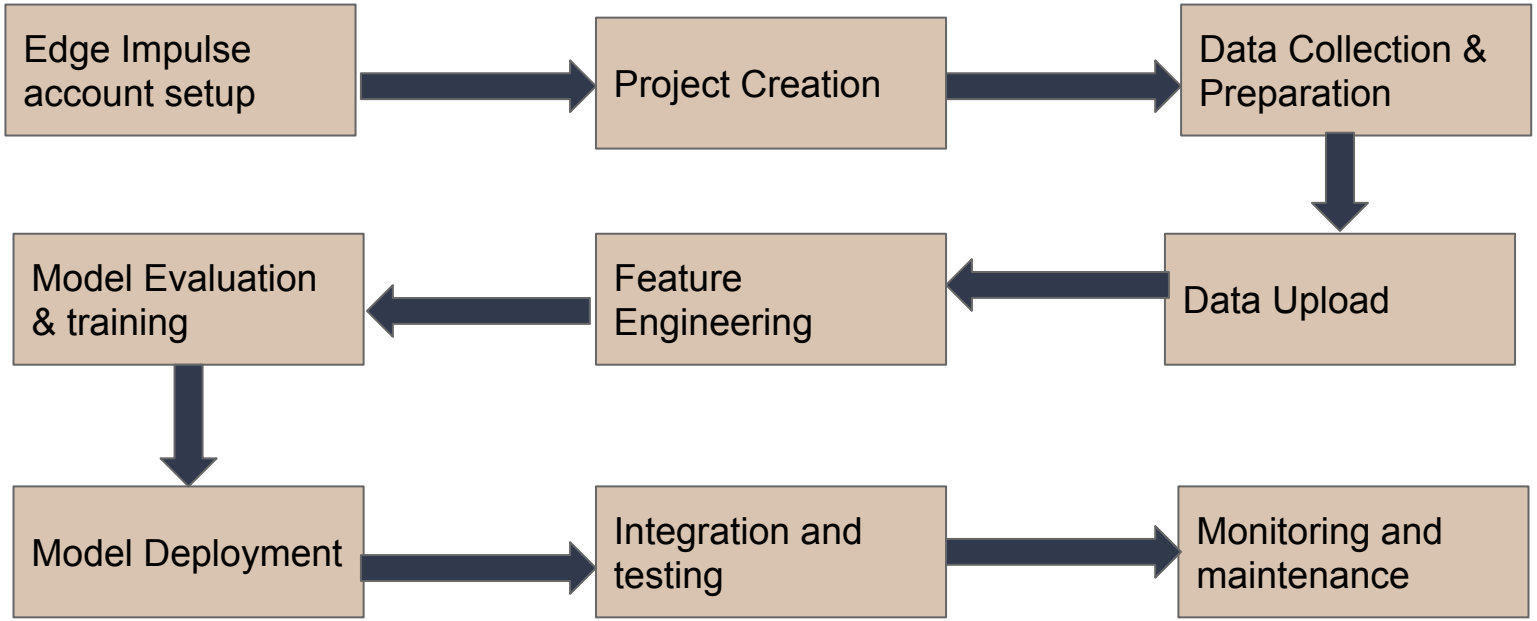
- **Raspberry Pi Camera**
  - Has a Sony IMX219 8-megapixel sensor.
  - Can be used to take high-definition video, as well as stills photographs.
  - supports 1080p30, 720p60 and VGA90 video modes

Implementation

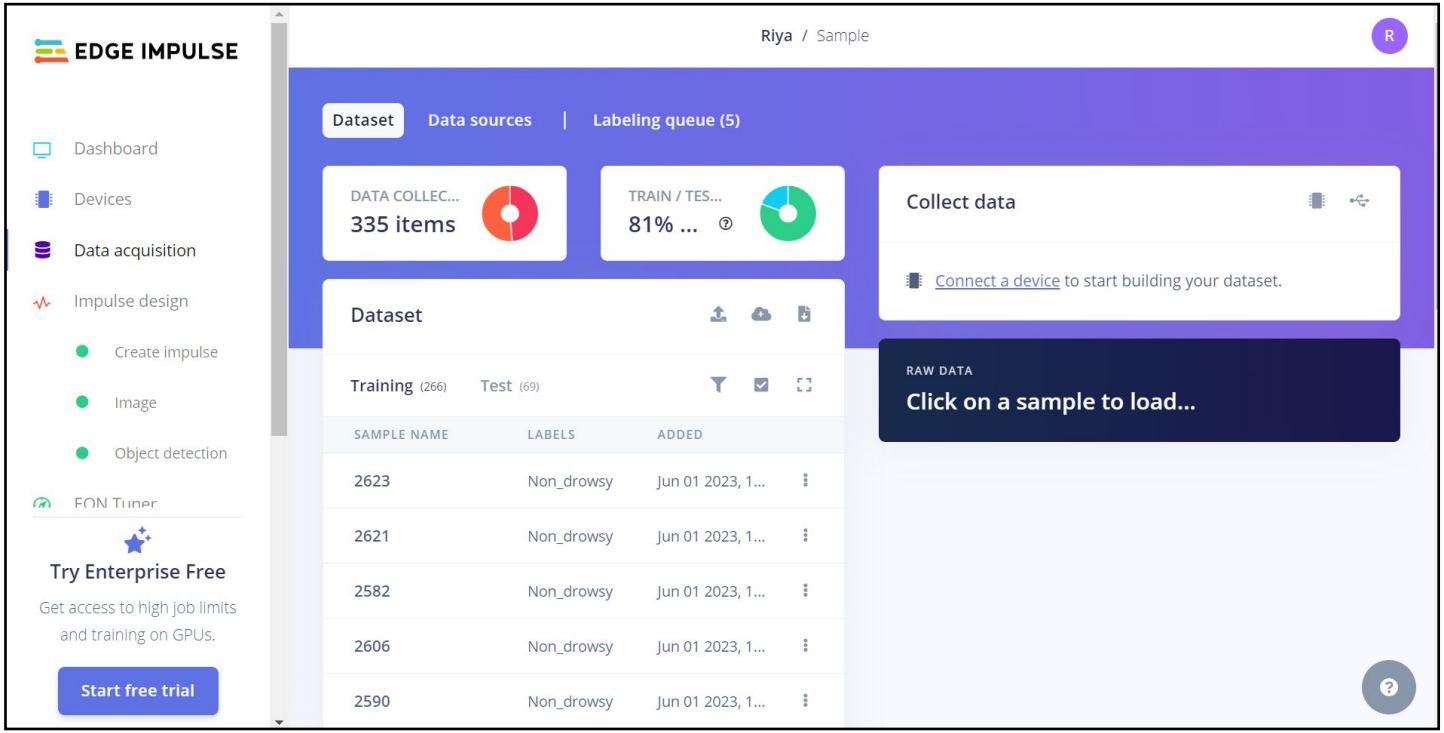
Edge Impulse

- A suite of engineering tools for building, training, and deploying machine learning models on edge devices.
- Cloud based platform for machine learning on edge devices.
- No Coding Required: provides a user-friendly interface
- Edge Computing: ensuring low latency and real-time processing.
- Support for Various Sensors
- Jetson Nano is fully supported by Edge Impulse
- Supports various ML tools like PyTorch and TensorFlow

Flow Diagram

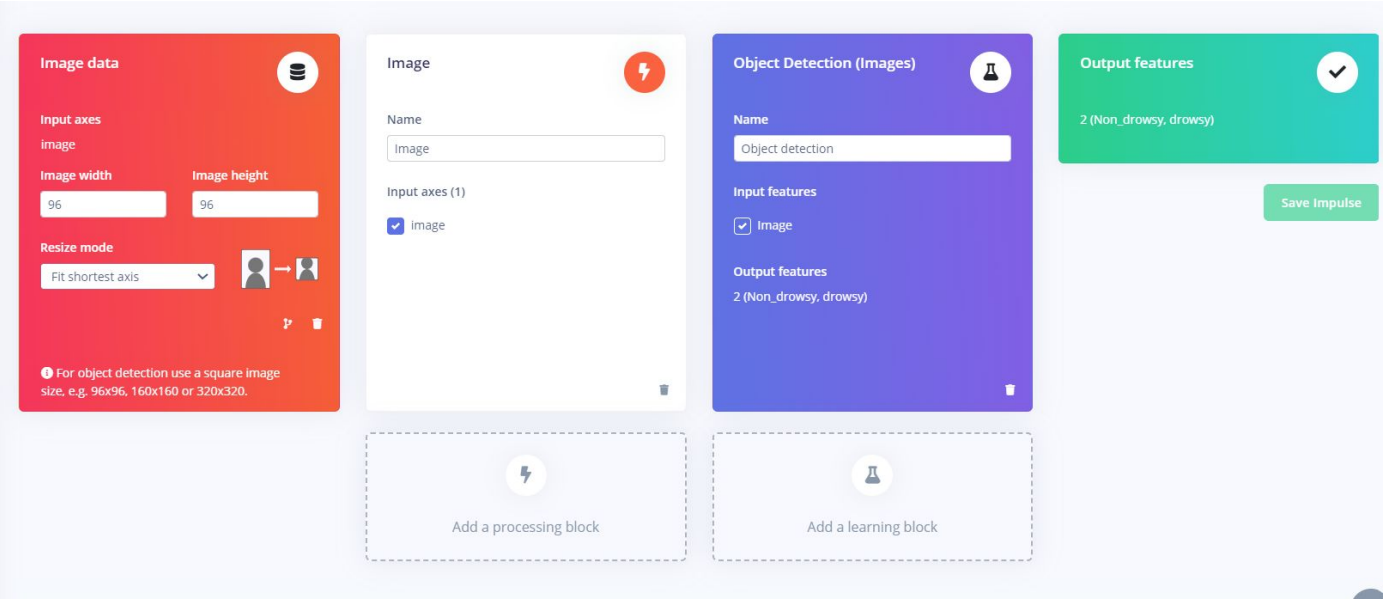


Building a Dataset : data acquisition



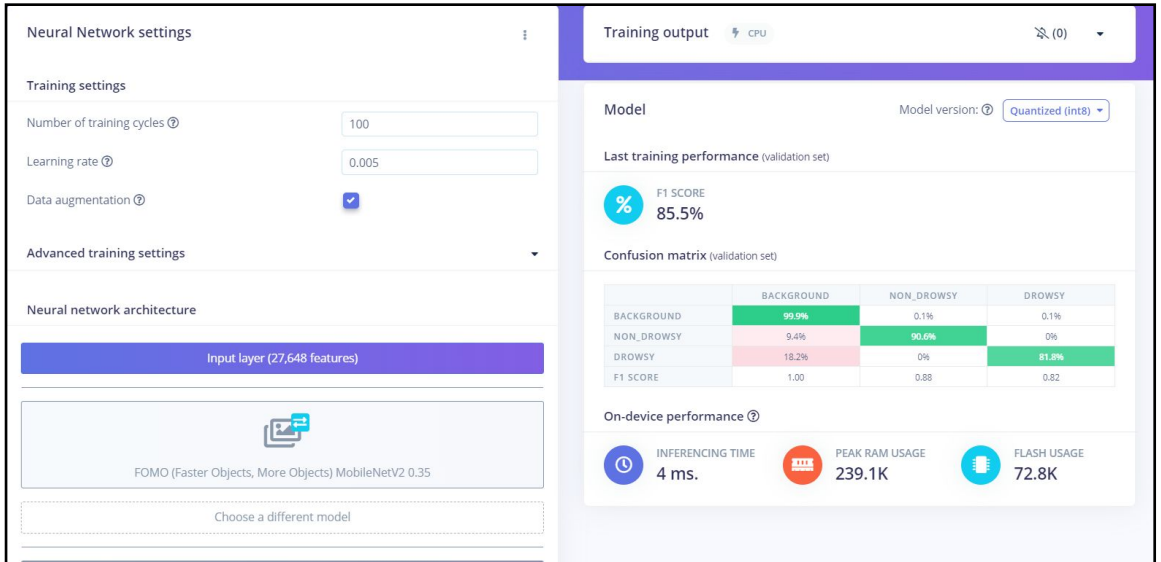
Designing an Impulse

An impulse takes the raw data, adjusts the image size, uses a preprocessing block to manipulate the image, and then uses a learning block to classify new data.



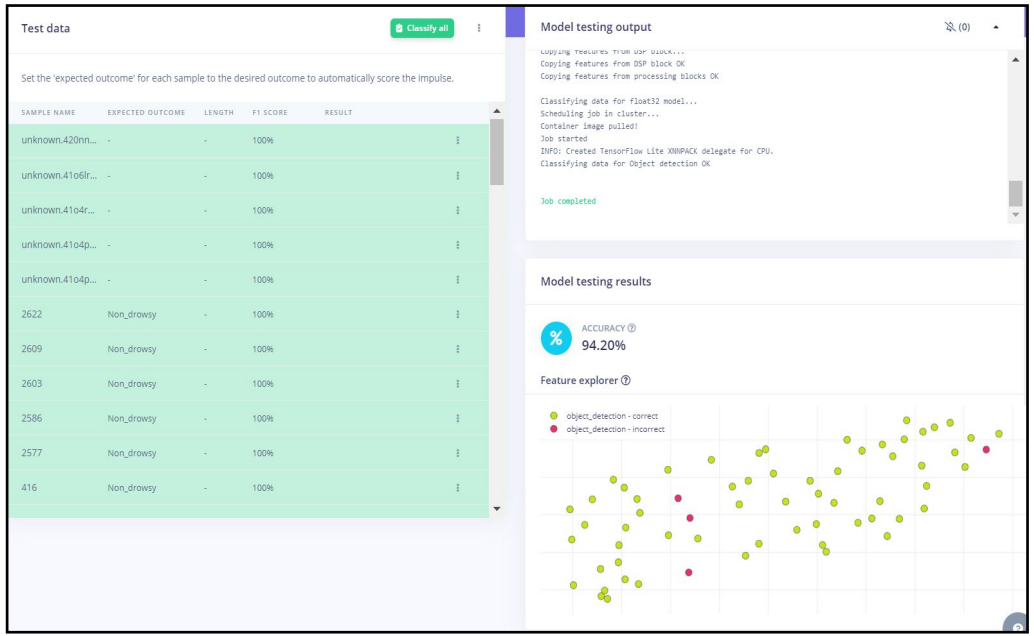
Configuring the transfer learning block

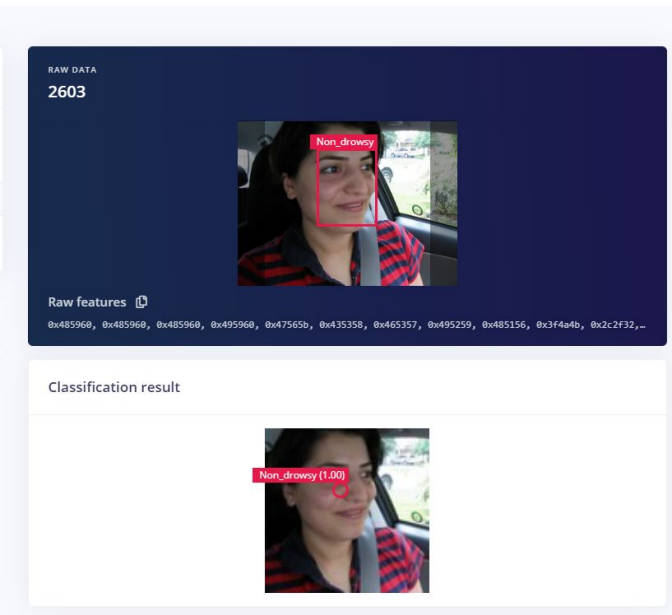
- Start Training the neural network model
- Take the image data as an input, and try to map this to one of the two classes.
- Transfer Learning : only retraining the upper layers of a neural network, leading to much more reliable models that train in a fraction of the time and work with substantially smaller datasets.



Validating the model

Use the data in the testing dataset to validate how well the model will work in the real world.





Testing and Drawbacks of Edge impulse

- Model Performance:
  We were able to achieve an accuracy of 94.20 with the test dataset.
  But with new real time images, the detection results were not that good.

- Model Training Constraints:
  While Edge Impulse provides tools for model training, the training process might be slower and less versatile compared to using more robust cloud-based training platforms.

- Edge Impulse provides two different methods to perform object detection:
  Using MobileNetV2 SSD FPN
  Using FOMO

YoloV5

- YoloV5 is an object detection model, designed to deliver high-speed, high-accuracy results in real-time.
- It is built on PyTorch

Implementation Steps:

Step 1: Dataset Collection and preparation

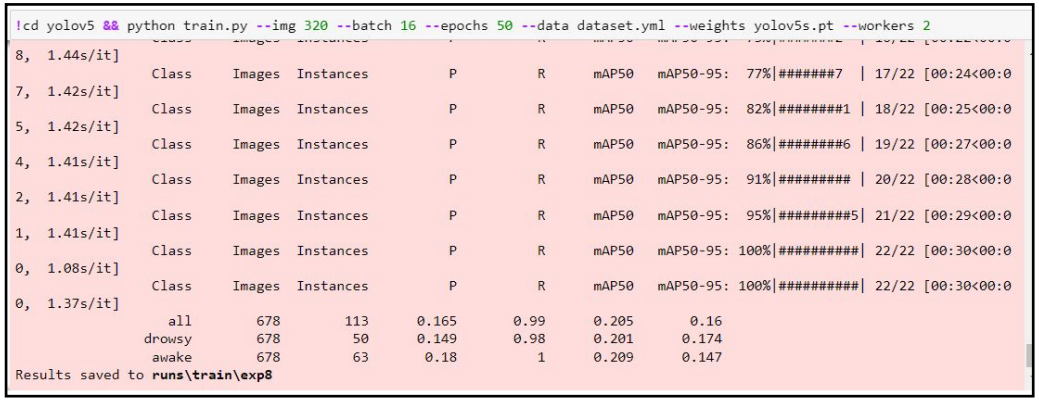
- For Drowsiness Detection : Dataset collected via camera (custom dataset)
  Classes 2 : drowsy and awake
  Number of images : 566
- For Traffic Sign Detection : Dataset took from German Traffic Sign Detection Benchmark (GTSDB)
  Classes 4 : prohibitory, mandatory, danger and other
  Number of images : 900

Divided into two group for training and validating 80-20.
Label text file contains the class, height and width of the image

Step 2 : Setting up the environment in Jupyter Notebook

- Cloning the YoloV5 repository and installing all the necessary packages like opencv, numpy, torch, torchvision etc required to train our model.

Step 3: Train and validate the model



```
!cd yolov5 && python train.py --img 320 --batch 16 --epochs 50 --data dataset.yml --weights yolov5s.pt --workers 2
8,  1.44s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  77%|#######7  | 17/22 [00:24<00:0
7,  1.42s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  82%|########1 | 18/22 [00:25<00:0
5,  1.42s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  86%|########6 | 19/22 [00:27<00:0
4,  1.41s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  91%|######### | 20/22 [00:28<00:0
2,  1.41s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95:  95%|#########5| 21/22 [00:29<00:0
1,  1.41s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 22/22 [00:30<00:0
0,  1.08s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|##########| 22/22 [00:30<00:0
0,  1.37s/it]
                   all        678        113      0.165       0.99      0.205       0.16
                drowsy        678         50      0.149       0.98      0.201      0.174
                 awake        678         63       0.18          1      0.209      0.147
Results saved to runs\train\exp8
```

Training is done using train.py
Image size : 320* 320
Batch : 16
Epochs : 50

YoloV5 model test results and drawbacks

Drawback of using Jupyter Notebook for training:

- The training took approximately 36 hours for traffic sign detection(900 images). Hence could not be used for large number of datasets

Drawback of using YoloV5

- Less prediction rates

YoloV7

- YOLOv7 provides a faster and stronger network architecture
- Provides a more effective feature integration method, more accurate object detection performance, a more robust loss function.

Comparison with YoloV5

- Higher inference speed (more than 20 FPS Faster).
- YoloV7 reduces the number of parameters by 22%, require 8% less computation power and has an increased average precision of 2.2%

Training YoloV7 using Google Collab

Implementation Steps:

Step 1: Dataset Collection and preparation (used the same dataset)

- For Drowsiness Detection : Dataset collected via camera (custom dataset)
  Classes 2 : drowsy and awake
  Number of images : 800
- For Traffic Sign Detection : Dataset took from German Traffic Sign Detection Benchmark (GTSDB)
  Classes 4 : prohibitory, mandatory, danger and other
  Number of images : 900

Step 2: **Clone the repository for YoloV7**

Step 3: Train and validate the model

```
!python train.py --device 0 --batch-size 16 --epochs 30 --img 640 640 --data data/custom_data.yaml --hyp data/hyp.scratch.custom.yaml --cfg cfg/training/yolov7x-custom.yaml --weights yolov7x.pt --name yolov7x-custom

2023-08-08 11:05:53.280887: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-08-08 11:05:54.599694: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
YOLOR 🚀 v0.1-126-g84932d7 torch 2.0.1+cu118 CUDA:0 (Tesla T4, 15101.8125MB)

Namespace(weights='yolov7x.pt', cfg='cfg/training/yolov7x-custom.yaml', data='data/custom_data.yaml', hyp='data/hyp.scratch.custom.yaml', epochs=30, batch_size=16, img_size=[640, 640], rect=False, resume=False, nosave=Fal
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, obj=0.7, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)
```

Image size : 640*640
Batch : 16
Epochs :30
Model : yolov7-x model
Used GPU for training

Testing in Google Colab: Tested images and observed that the prediction is now 80%-90%.

Testing and Analysis of YoloV7 model

Step 4: Deployment into Jetson Nano

- Created separate virtual environments for both the Yolo models for testing
- Installed required packages like OpenCV, Numpy, Matplotlib, etc
- Installed Jupyter Notebook
- Cloned the repository for YoloV7

Step 5: Testing the model using images and videos in Jetson Nano

- Tested few images and videos and observed that the inference time is not as expected.
- The processing speed and the FPS was low.

YoloV7 Prediction results

TensorRT

- High performance deep learning inference runtime for image classification, segmentation, and object detection neural networks.
- Built on NVIDIA's parallel programming model : CUDA

Steps for Converting the YoloV7 model to TensorRT model

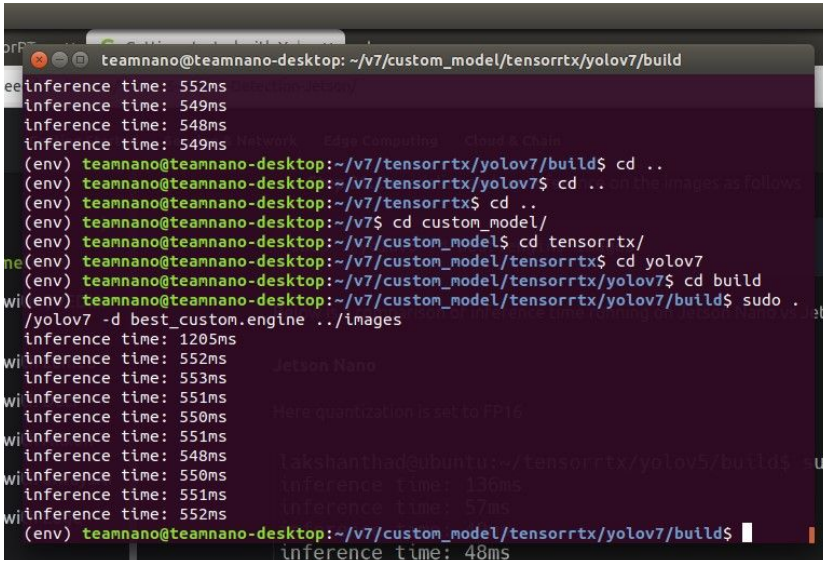
- Cloned the TensorRT repository to the system.

```
git clone https://github.com/wang-xinyu/tensorrtx
```

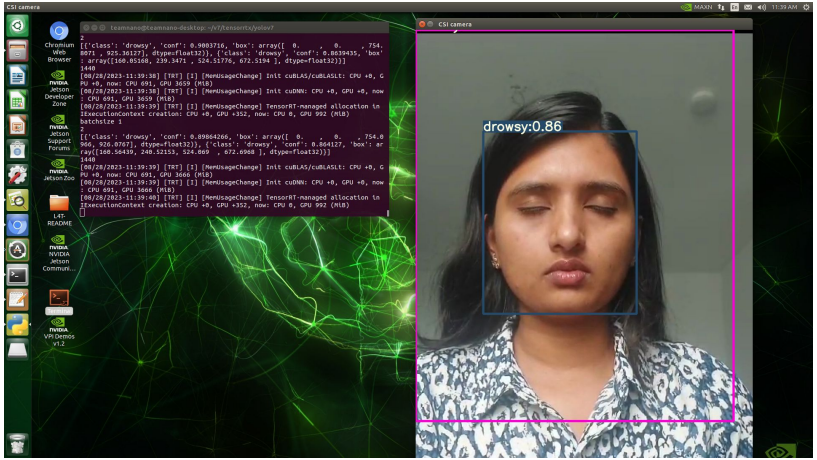
- Converted the .pt weights of the YoloV7 to the .wts file using the gen_wts.py.
- The generated .wts file is then converted to .engine file. This engine file is used for the testing and detections.

Testing and Analysis

- Tested the images and videos using the generated engine file.
- Observed that the inference time is now increased to 2FPS.



Results



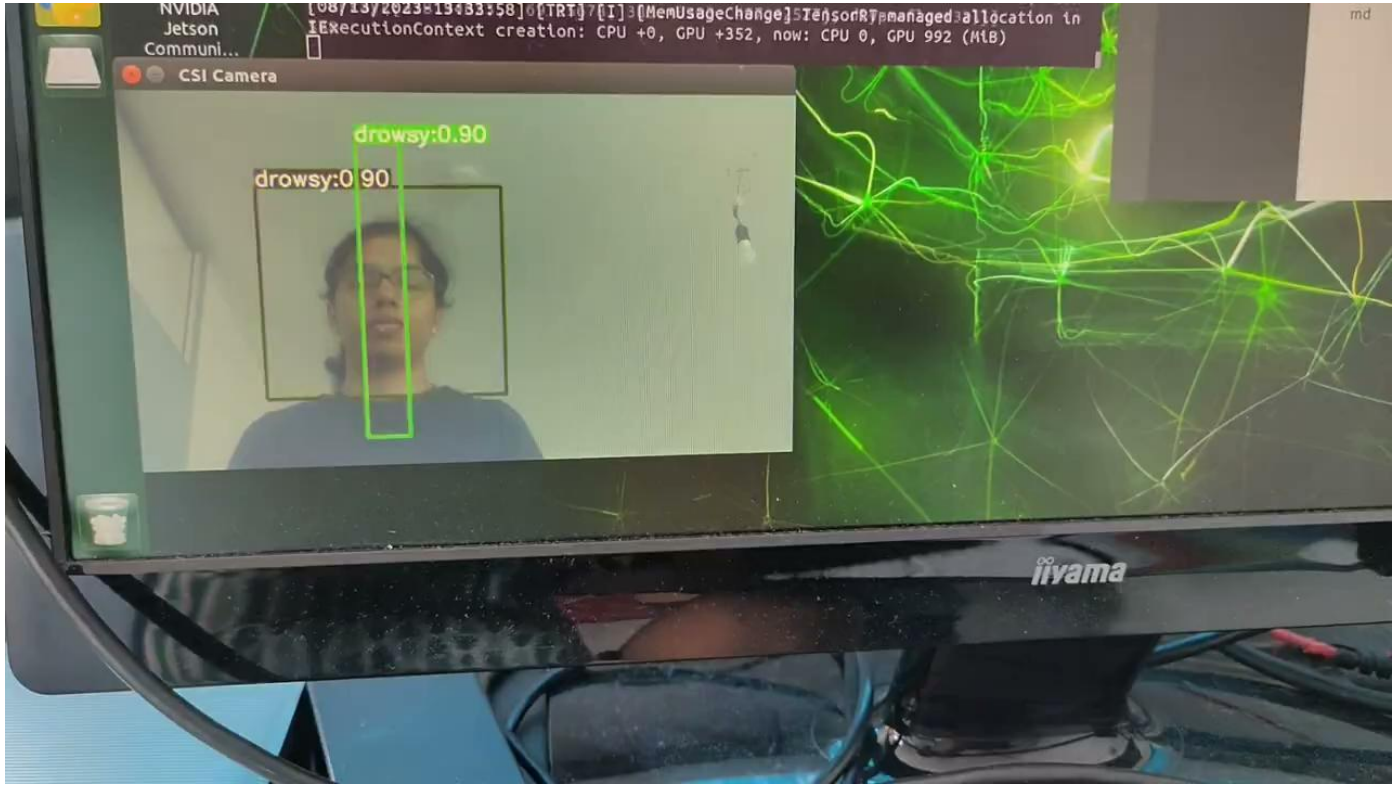
Video testing

Image testing

Results

Video Testing

Real Time Testing



Traffic Sign detection – sample predicted output

Parallel Processing of two cameras

- For testing both the models (driver assistance system and driver monitoring system), live feed from both the cameras are taken and processed.
- Multithreading is used to run the inference on each camera feed independently.
- We observed that running resource-intensive tasks concurrently on the Jetson Nano led to power consumption levels that exceeded the device's capacity.
- As a result, the system experienced performance throttling to prevent overheating and maintain power efficiency.
- Given the limited power available, the execution of parallel tasks placed excessive demand on the device's resources, caused unexpected slowdowns, degraded inference performance, and an automatic rebooting.

Conclusion

- By integrating cutting-edge AI technologies, YOLOv7 and TensorRT, we've created a robust solution that addresses critical aspects of road safety.
- We faced many challenges during the implementation and were able to resolve most of them, including the jetson nano rebooting issues,
- Learned on how to train the model on custom datasets and testing them.
- Learned about different deep learning algorithms.
- Learned how to deploy and test the model in Jetson Nano.
- In conclusion, our project demonstrates the impact of AI-driven driver assistance systems. By leveraging YOLOv7 and TensorRT, we've crafted a technology-driven solution that has the potential to significantly reduce accidents and create safer driving environments.

Future Improvements

Drowsiness Detection

1. Real-time Feedback and Alerts:
    - Alerting the driver, such as haptic feedback on the steering wheel or seat.
2. Integration with Autonomous Systems:
    - Integrate drowsiness detection into autonomous vehicles to ensure that the system can take control when the driver becomes drowsy.

Traffic Sign Detection

1. Multi-Sign Detection:
    - Extend the system to detect multiple signs in a single frame, allowing for a more comprehensive understanding of the traffic environment.
2. Real-time Traffic Analysis:
    - Integrate the detected traffic sign data with mapping and navigation systems to provide real-time information to the driver about upcoming road conditions and regulations.

Future Improvements

Parallel Processing (Sensor fusion) improvement using **DeepStream SDKs**:

- For better performance and efficiency, consider using NVIDIA's DeepStream SDK, which is designed for real-time AI inference on Jetson devices and includes features for working with multiple cameras and models.
- Additionally, optimizing the models and algorithms for efficiency and resource utilization may help mitigate the effects of power throttling which hindered the parallel processing ability.

Reference

- https://github.com/nicknochnack/YOLO-Drowsiness-Detection
- https://github.com/AarohiSingla/yolov5
- https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format
- https://wiki.seeedstudio.com/YOLOv5-Object-Detection-Jetson/
- https://medium.com/@jurespeh/yolov7-with-tensorrt-on-jetson-nano-with-python-script-example-63099fa7c8a5
- https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit
- https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6
- https://viso.ai/deep-learning/yolov7-guide/
- https://github.com/WongKinYiu/yolov7
- https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/object-detection/object-detection
- https://github.com/ultralytics/yolov5
- https://github.com/JetsonHacksNano/CSI-Camera
- https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format

Thank You!