CNST 6308-Data Analysis in Construction Management

Traffic Monitoring & Detection

Team Members:

Varshitha Challa- 2156555

Janaki Obillaneni-2149246

CODE: https://github.com/VarshithaChalla9/TrafficMonitoringAndDetection

Table of Contents

Abstract **3**

Introduction **3**

Literature Survey **4**

Experiment **5**

- **Dataset** **5**
- **Data Augmentation &Data preprocessing** **6**
- **Methodology & Implementation** **7**

Results **13**

- **YOLO_V5** **14**
- **YOLO_NAS** **14**

Conclusion **15**

References **15**

ABSTRACT:
Due to their inability to handle the ever-changing and complicated nature of contemporary urban traffic, traditional traffic management solutions frequently result in problems including traffic jams, accidents, and inefficient traffic flow. Traffic flow prediction is very important in urban regions where traffic congestion is a major problem. So, the project primarily focuses on the requirement for advanced systems that can accurately deliver the transportation-related real-time road analysis. The main objective is to develop a system that can detect and track various objects like traffic, pedestrians, and different vehicles such as truck, car, motorbike and bicycle in a different weather and traffic situations using computer vision and object identification techniques. By using this large dataset of traffic camera photos from many nations, this project aims to ensure worldwide viewpoints and applications in various urban environments. Under different environmental situations, accurately documenting these photos for object detection and planning works as well. With this effective city planning, lowers accidents it helps to create smart cities as well. This system needs to be scalable and flexible enough to accommodate changes in traffic and urban infrastructure in the future, in addition to being able to satisfy the needs of traffic management now. The model, we used was the YOLOv8, YOLO-NAS model where it uses a neural network to the full image, and then divides the image into grids and predicts bounding boxes and probabilities which improve efficiency and guarantee greater accuracy in the identification of objects in traffic.

INTRODUCTION:
In the rapidly changing world of urban development, traffic control and road safety are critical issues. A variety of road users, including bicycles and pedestrians, along with an increase in the number of vehicles on the road have highlighted the necessity for advanced and effective traffic monitoring systems. The goal of object detection, a crucial area of computer vision, is to find and categorize things in an image that have a variable number. This technology uses deep learning or image processing to recognize and identify traffic signs, cars, and pedestrians in traffic scenes, providing the foundation for the development of intelligent transportation. Convolutional neural network-based object identification algorithms can be broadly classified into two groups: one includes one-stage algorithms denoted by the YOLO series, and the other comprises two-stage algorithms represented by the RCNN series. By using modern computer vision techniques and object recognition technologies, the Traffic Monitoring & recognition project aims to address these issues.

The project's focus on using a large and varied dataset of traffic camera photos that are sourced from several different nations mainly from turkey. This dataset includes a detailed description of many things encountered in everyday traffic scenarios, such as vehicles, pedestrians are carefully labeled. This kind of information is very helpful in training and developing advanced object detection algorithms that can operate with high reliability in real-world scenarios. The Traffic Monitoring & Detection project focuses on adaptability to various environmental circumstances. The dataset comprises photos shot at different times and in varying weather conditions, enabling the system to function well in a range of external scenarios. This flexibility is essential for guaranteeing the system's accuracy and resilience in every circumstance, whether it's rainy, or early in the morning.

This project has several applications that go beyond just tracking traffic. It seeks to make a major contribution to raising road safety, lowering traffic jams, and facilitating better traffic flow generally. The technology can support proactive traffic management, accident avoidance, and well-informed urban planning by offering real-time data and analysis. Furthermore, it is in line with the growing trend towards more intelligent and linked urban surroundings as a tool for the development of smart cities.

Overall, the Traffic Monitoring & Detection project is an innovative method to control traffic that makes use of state-of-the-art technology to satisfy the complex requirements of modern urban infrastructure. In addition to improving traffic monitoring and safety, the success might serve as a model for further advancements in the field of artificial intelligence-driven urban management systems.

LITERATURE REVIEW:

[1] The study "The Real-Time Detection of Traffic Participants Using YOLO Algorithm" investigates the use of the YOLOv3 algorithm for real-time traffic participant detection, an essential function for ADAS and autonomous vehicles. It deals with identifying distinct traffic elements in varied situations, such as vehicles, trucks, and people. The Berkeley Deep Drive dataset initially using a model pretrained on the COCO dataset is used to train the YOLOv3 algorithm, which is renowned for its accuracy and real-time processing. It displays considerable increases in accuracy and precision. The study shows how well YOLOv3 performs in a variety of weather and traffic scenarios and suggests ways to improve it even further by adding more training and integrating more sensors. This research makes a significant contribution to the field of autonomous driving systems object detecting technology.

[2] The study "Traffic Flow Prediction Using YOLO Algorithm" by creating a system for traffic flow prediction that makes use of the YOLO (You Only Look Once) algorithm. In this paper urban traffic congestion is addressed. It highlights how well the YOLO algorithm processes road pictures for vehicle detection, even in the face of obstacles like interference from the environment and stationary vehicle identification. The COCO dataset was used to train the system, which shows good reliability and accuracy. The report also suggests a mobile application that classifies traffic as light, medium, or heavy and provides real-time traffic situation updates. This work advances traffic management technology by providing a workable method for predicting traffic flow and reducing urban congestion.

[3] The study "Object Detection in Autonomous Vehicles: Status and Open Challenges" by Abhishek Balasubramaniam and Sudeep Pasricha's provides a thorough examination of object detection technologies in autonomous vehicles (AVs), with a particular emphasis on deep learning-based techniques. It illustrates how object detection has developed over time, from simple methods to sophisticated deep learning models like SSD, RetinaNet, and YOLO (You Only Look Once). The study describes the different iterations of the YOLO algorithm, highlighting their advances in real-time processing and detection accuracy. The difficulties in integrating these technologies in AVs are also covered, mostly because of the constrained onboard computer power. To overcome these challenges, the research explores optimization strategies like quantization and pruning. Lastly, it highlights unresolved issues and potential paths forward in AV object detection, including the necessity of effective processing, sensor fusion, and semi-supervised learning

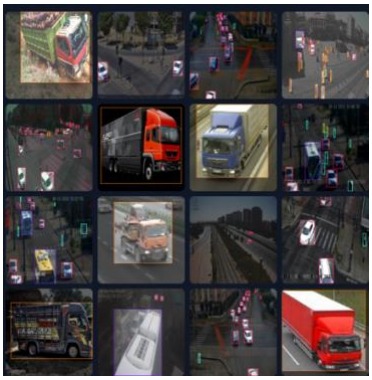
strategies. This paper offers a thorough analysis of object detection in autonomous cars, covering both present and future developments.

[4] The paper "Small Object Detection in Traffic Scenes Based on YOLO-MXANet" A novel method for enhancing object recognition in traffic scenes. The COCO dataset was used to train the system. The authors present YOLO-MXANet, an improved iteration of the YOLO algorithm designed to detect small objects in difficult traffic locations with greater efficiency and accuracy. A Multi-Scale Feature Enhancement Fusion Network for efficient feature integration, an SA-MobileNeXt backbone network for reduced complexity, and the usage of the SiLU activation function for quicker model convergence are some of YOLO-MXANet's key characteristics. The study shows that YOLO-MXANet outperforms current algorithms in terms of accuracy and speed, indicating that it is a viable option for applications involving autonomous driving and intelligent transportation.

[5] The development of a traffic sign identification system utilizing YOLOv8 is the main objective of the paper "A performance comparison of YOLOv8 models for traffic sign detection in the Robotaxi-full scale autonomous vehicle competition". By supporting human and autonomous drivers in comprehending their environment and responding properly to road conditions, the system seeks to increase road safety. It evaluates the efficacy and suitability of several YOLOv8 model configurations for autonomous driving applications. The paper contains information on the development of datasets, the use of software, and the outcomes of experiments. It illustrates how deep learning and AI may improve the effectiveness and safety of transportation.

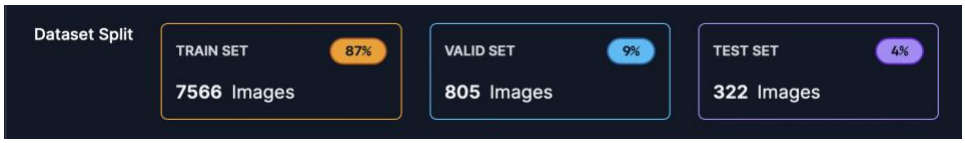
EXPERIMENT:

- **Dataset**:
  The Street view Object detection dataset we used in this project was exported from roboflow.com. Three types of data are available: 322 photos for test, 7566 images for training, and 805 images for valid data. This collection has 8693 photos in total. Majority of the photos in the collection were gathered from Turkish cities like Konya, Istanbul, and Bursa. And not just from cities in Turkey, but also from foreign nations. Many items seen in typical traffic scenarios are described in depth in this collection, including cars, trucks and pedestrians with meticulous labeling. Below are the images for splitting the data and dataset.



Train Data

Test Data



- **Data Augmentation & Data Pre-processing:**
  Data augmentation: It is a technique used in Machine learning model which increases the size of the data for training models without gathering new data by performing different operations, such as flipping, rotating, and zooming. We have listed a few augmentation techniques that we used in our project models below.
    - Horizontal flip: This augmentation creates a mirror-like appearance where the image is flipped horizontally (i.e., from left to right). It improves the model's ability to identify items regardless of how they are oriented.
    - **Saturation (-61% to +61%):** This augmentation technique is performed to teach the model to identify things in a range of color environments that we set. In this project we modified the images color saturation to -61% to +61% where if the saturation raises to 61% then it makes the image color more vivid and lowering to 61% means it makes the image color to grayscale.
    - **Brightness range:** These need to be specified in the form of a list [lower, higher]. The brightness of the image will be within the range. In this project we set the range to -25% to +25%. Varying within the given range modifies the image's brightness to replicate various lighting conditions.
    - **Noise:** The training dataset grows larger with the addition of noise. Random noise is added to the input variables each time a training sample is subjected to the model, causing the variables to change each time. In this project we set the noise to 2% of pixels which means this effect applies random noise to a maximum of 2% of an image's pixels. By simulating flaws in real-world photos, such as graininess or sensor noise, this makes the model more resilient to changes of this kind.
    - In data augmentation, "outputs per training example" means producing several altered versions of each original image in a dataset. In our project it is set to 3 which means three augmented copies of each original image using different adjustments in the collection are produced. By improving dataset variety, this method contributes to the creation of machine learning models that are more resilient.

  Data Pre-processing: Preprocessing is an essential stage of making data ready for deep learning and machine learning models. Preprocessing the data before feeding it to a model is an important step as we can clean the irrelevant or unnecessary data and improve the accuracy and the performance of the model.
    - **Auto-Orient:** Using embedded orientation metadata, this phase makes sure that images are aligned correctly. It is necessary for consistency since otherwise, images that are processed differently could be those that are shot with various devices or in various positions (such as landscape or portrait).
    - **Resize:** It is nothing but resizing all the images to the same size as every image is in different sizes. So, we have resized all the images in the same size .All the images in the dataset are resized or stretched to a consistent 640x640 pixel size throughout this phase. The word "stretch" refers to the process of scaling an image to fit this size, sometimes requiring an adjustment to the aspect ratio. For neural networks to

be trained, which usually require inputs of regular dimensions, uniform image sizes are essential.

The overall goal of these preprocessing and augmentation techniques is to produce a strong and diverse dataset, which is important for a reliable and strong computer vision model.

- **Methodology & implementation:**

YOLO is a real-time object detection technique that predicts bounding boxes and class probabilities straight from an input image by utilizing a single neural network. Compared to conventional object detection techniques, which usually include two stages: first producing region proposals, and then categorizing each proposal, this makes it much faster.

YOLO Architecture:

The four primary parts of the YOLO architecture are as follows:

i. Backbone: Convolutional neural networks (CNNs), which are the backbone of the system, are used to extract information from the input image. A sequence of convolutional layers, pooling layers, and activation functions make up the CNN.
ii. Neck: The neck is made up of several layers that join the prediction head and the backbone. The accuracy of object detection can be increased by fusing features from various backbone levels using the neck layers.
iii. Prediction Head: The prediction head is a collection of layers that estimates class probabilities and bounding boxes for every cell in the input picture. Typically, the prediction head is made up of activation functions and completely connected layers.
iv. The output layers comprise a series of layers that transform the prediction head's output into a comprehensible manner. Class probabilities and bounding box coordinates are commonly found in the output layers.

Below is a more detailed explanation of each component:

Backbone: It extracts the features from the input image. The image's characteristics serve as a representation that holds the crucial data needed to identify objects. Activation functions, pooling layers, and convolutional layers combinedly make up the backbone. To extract local patterns from the image, convolutional layers are employed. The characteristics are made less dimensional by using pooling layers. Non-linearity is added to the model using activation functions. Usually, a CNN that has already been trained on a sizable image dataset serves as the foundation. As a result, the backbone can extract high-quality features for object detection.

Neck: The neck is in connection for joining the prediction head and the backbone. Features from several levels of the backbone can be combined using the layers of the neck. Because the lower-level features have more detail information and the higher-level features contain more semantic information, this can increase the accuracy of object detection. Common neck architectures include the following:

Feature Pyramid Networks (FPNs): FPNs combine features from several backbone tiers by means of a set of lateral connections.

Path Aggregation Networks (PANets): PANets combine features from several backbone levels by using a collection of top-down and bottom-up paths.

Prediction Head: For every cell in the input image, the prediction head is responsible for estimating bounding boxes and class probabilities. Typically, the prediction head is made

up of activation functions and completely connected layers. The neck's features are merged into a single vector using the completely connected layers. The model's non-linearity is added using the activation functions. A tensor with the same dimensions as the input image is the prediction head's output. The tensor holds the following data for every cell in the input image:

a. A coordinate vector for a bounding box
b. A collection of probability for each class

The location and dimensions of the bounding box for the cell are specified by the bounding box coordinate vector. The likelihood that each class's object is present in the cell is indicated by the class probabilities.

Output Layers: The output layers are responsible for transforming the prediction head's output into an understandable manner. Class probabilities and bounding box coordinates are commonly found in the output layers. Most of the time, the bounding box coordinates are normalized to have values between 0 and 1. Usually, the class probabilities are transformed into a one-hot vector, where the predicted class is indicated by the index with the highest likelihood.

Advantages of YOLO:

i. Real-time: Since YOLO is an object recognition method, it can identify objects in pictures at a high frame rate. For applications like autonomous driving and video surveillance that demand real-time object detection, this makes it a great fit.
ii. Precise: Yolo is precise, particularly while dealing with bigger items.
iii. Effective: YOLO's efficiency allows it to identify objects in photos with minimal computing overhead. Because of this, it works well with apps that have constrained processing capabilities, such those on mobile devices.

Disadvantages of YOLO:

i. Accuracy with small things: Compared to larger objects, YOLO is less accurate with small objects.
ii. Localization errors: The anticipated bounding boxes might not always be perfect since YOLO occasionally makes these mistakes.

As an object detection method, YOLO is strong and adaptable, making it suitable for a wide range of applications. To evaluate the performance of the models and examine the observations of them at various stages for detecting objects we used YOLO as mentioned earlier. In working with YOLO, we implemented two models,

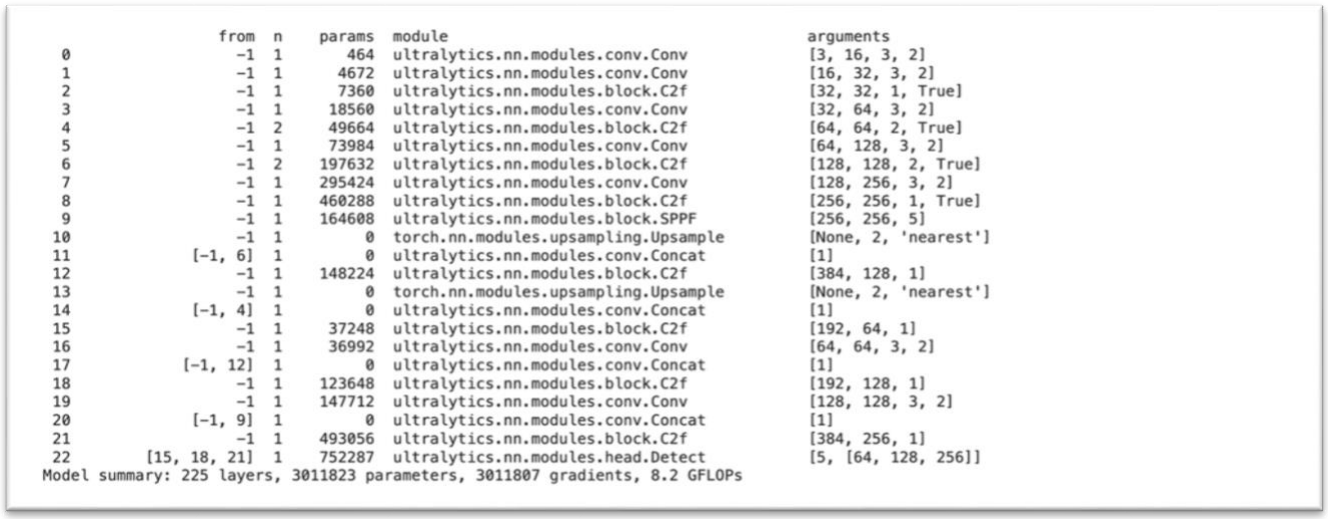
a. YOLOv8 (you only look once)
b. YOLO-NAS

- **Yolo V8:**

In computer vision, YOLO (You Only Look Once) is a well-liked collection of object detection models used for real-time object recognition and classification. It was first created by Joseph Redmon, Ali Farhadi, and Santosh Divvala with the goal of achieving high object detection accuracy at a fast real-time speed. The model family is categorized as one-stage object detection models, which use a convolutional neural network (CNN) to process an entire image in a single forward pass. YOLO's primary feature is its a single-

step detection method, which is intended to identify things quickly and accurately. In contrast to two-stage detection methods, like R-CNN, which identify and then propose regions of interest, YOLO processes the entire image in a single pass, which results in a faster and more efficient process.

We choose to use YOLOv8 in this project is likely to train more quickly than the other two-stage object detection models and YOLOv8's excellent precision over its earlier models. A popular collection of object detection models named YOLO (You Only Look Once) is used for real-time object detection and classification. The architecture of the model is described in this summary, together with information on each layers and its type, connections, and the number of parameters. Below is the summary of the model we have implemented.



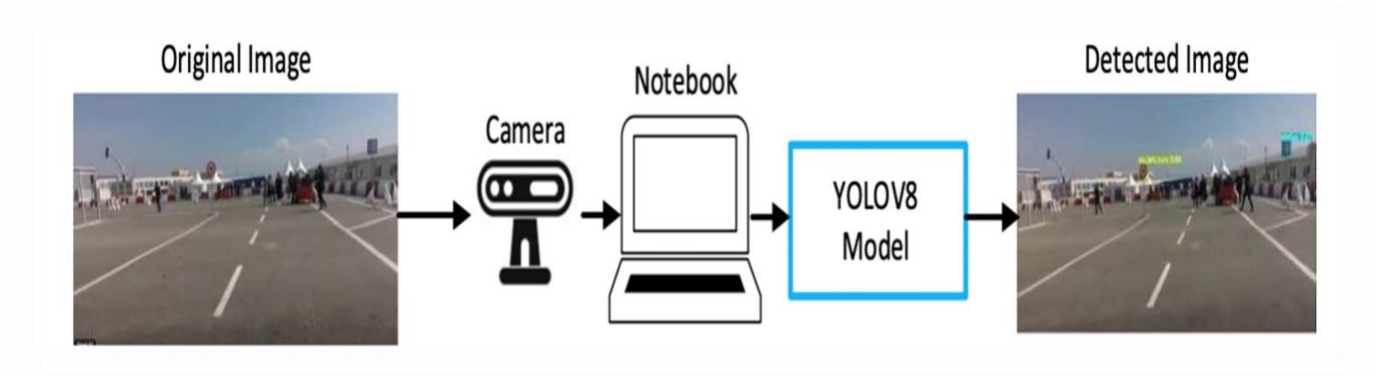
	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
16	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
19	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
22	[15, 18, 21]	1	752287	ultralytics.nn.modules.head.Detect	[5, [64, 128, 256]]

Model summary: 225 layers, 3011823 parameters, 3011807 gradients, 8.2 GFLOPs

In the above summary diagram, From represents the input layers. Layers denoted by negative numbers are those that count backwards from the current layer i.e., '-1' denotes the preceding layer. N represents the number of model iterations for this layer. params represents the quantity of the layer's parameters. The model's parameters are components that are acquired through training data. Module represents the layers which covers common neural network layers such as convolutional layers (Conv), particular blocks such as C2f, and additional layers such as Upsample or SPPF (Spatial Pyramid Pooling in the Feature Space). From the summary, we can say that we used 225 layers on the model. The total number of parameters that can be trained is 3011823 and the gradients that are utilized in backpropagation model training are 3011807. The model's computational complexity is indicated by GFLOPs (Giga Floating Point Operations).The number of GFLOPs used are 8.2. The Greater the GFLOPs may indicate that the model requires more processing power to run.

Later, the model training session on YOLO (You Only Look Once) object detection is performed. It goes over performance measures, the training procedure, and the assessment at the end. It provides the details the pretrained weight transfer learning, automatic optimizer selection, and data augmentation processes used in the training of a YOLO object identification model. It records training metrics over several epochs, displaying gains in precision and decreases in losses. Model

efficiency and performance by class are reported in the final evaluation. The location for storing the training results is also given, along with a summary of the model's complexity.



The above picture shows how to use a YOLOv8 model to visually describe the process of object detection in an image and describes the workflow of an object identification system, starting with the original image capture and ending with an output image that contains objects that have been discovered after computational analysis using an advanced YOLOv8 model. It has a three-part structure that illustrates how input is transformed into output. The three primary elements are:

- Original Image: The leftmost frame presents the unprocessed image, depicting multiple vehicles serving as the raw data input.
- Process Flow: A camera icon represents the process of capturing images and represents the first stage of data acquisition. Then, a picture of a laptop with the annotation "Notebook" represents the computer platform used to process data. Next to it, an analytical engine denoted by a blue box labeled "YOLOv8 Model" is where the image is processed algorithmically to find objects.
- The Detected Image: The last frame on the right shows the result of the analysis; it is now assumed that the original scene included bounding boxes that identified and localized objects.

- **YOLO-NAS:**

  YOLO-NAS is an upgraded basic model for object detection, modeled after YOLOv6 and YOLOv8. Improved training techniques, AutoNac optimization, post-training quantization, a new quantization-friendly basic block, and pre-training on top datasets are all shown. It outperforms existing YOLO models on many datasets and demonstrates remarkable performance in real-time edge device applications.

  YOLO-NAS, created by Deci AI, is the most advanced object detection model available today, utilizing Deci's own Neural Architecture Search technology, AutoNAC. YOLO-NAS surpasses rival models in speed and accuracy by incorporating the most recent developments in deep learning to improve quantization support and the accuracy-latency tradeoff of the current YOLO models. By providing improved accuracy-latency and quantization support tradeoffs in object identification, this novel technique guarantees superior performance. With the addition of YOLO-NAS, the model becomes more suitable for real-time edge device applications by increasing its performance-per-compute ratio, localization accuracy, and capacity to detect small objects. We choose to use YOLO-NAS in this project because it has several benefits when used for traffic observation.

Below are the benefits that I mentioned.

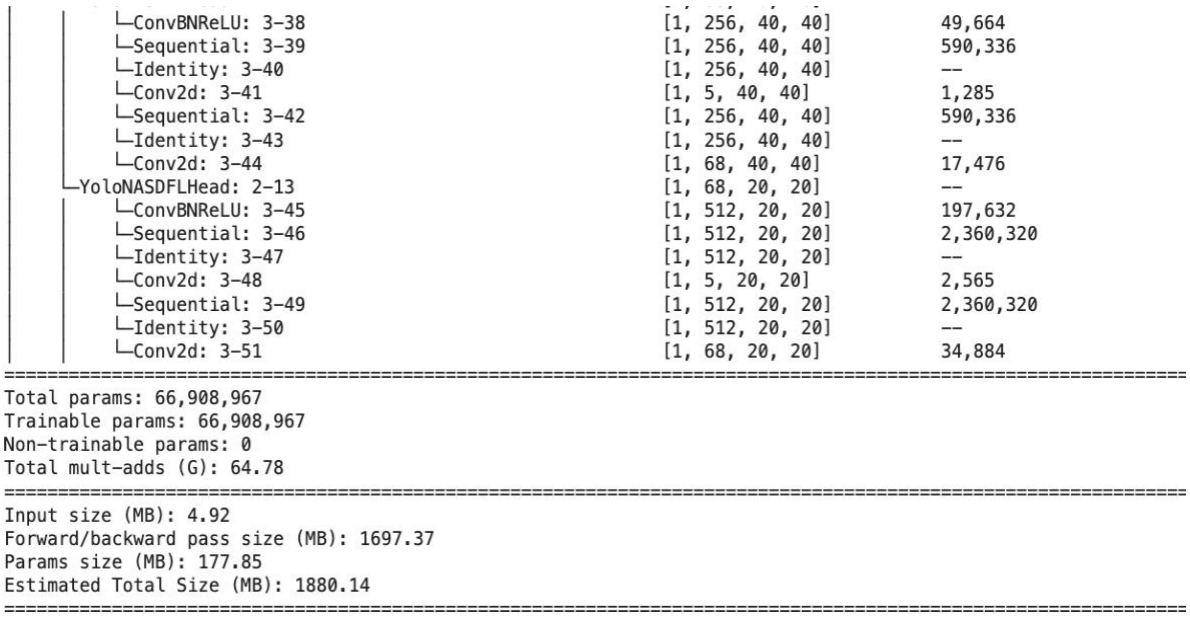
a. Precision: YOLO-NAS recognizes traffic with a high degree of precision using visual data.
b. Efficiency: YOLO-NAS is an object detection model that functions remarkably quickly, which makes it perfect for tasks that require a quick turnaround.
c. Adaptability: Yolo-NAS has a versatile design that makes it easy to modify for a range of datasets and use scenarios.

Because of its versatility, YOLO-NAS can be used for a variety of traffic observation applications, such as:

a. Traffic Monitoring: By using YOLO-NAS, traffic flow management could be improved, and regions that are prone to congestion might be identified.
b. Incident Awareness: YOLO-NAS can recognize events connected to traffic, like crashes.
c. Data acquisition:YOLO-NAS can help collect vehicle counts and classifications, among other traffic-related data.

A large amount of traffic photo data is used to train YOLO-NAS. This enables it to identify many different types of moving vehicles, such as motorbikes, lorries, buses, and cars. Furthermore, YOLO-NAS can identify traffic even in challenging circumstances like dim lighting and shadows. Below is the summary of the model we have implemented.

YoloNAS_L	[1, 8400, 4]	--
├─NStageBackbone: 1-1	[1, 96, 160, 160]	--
│ └─YoloNASStem: 2-1	[1, 48, 320, 320]	--
│ │ └─QARepVGGBlock: 3-1	[1, 48, 320, 320]	3,024
│ └─YoloNASStage: 2-2	[1, 96, 160, 160]	--
│ │ └─QARepVGGBlock: 3-2	[1, 96, 160, 160]	88,128
│ │ └─YoloNASCSPLayer: 3-3	[1, 96, 160, 160]	758,594
│ └─YoloNASStage: 2-3	[1, 192, 80, 80]	--
│ │ └─QARepVGGBlock: 3-4	[1, 192, 80, 80]	351,360
│ │ └─YoloNASCSPLayer: 3-5	[1, 192, 80, 80]	2,045,315
│ └─YoloNASStage: 2-4	[1, 384, 40, 40]	--
│ │ └─QARepVGGBlock: 3-6	[1, 384, 40, 40]	1,403,136
│ │ └─YoloNASCSPLayer: 3-7	[1, 384, 40, 40]	13,353,733
│ └─YoloNASStage: 2-5	[1, 768, 20, 20]	--
│ │ └─QARepVGGBlock: 3-8	[1, 768, 20, 20]	5,607,936
│ │ └─YoloNASCSPLayer: 3-9	[1, 768, 20, 20]	22,298,114
│ └─SPP: 2-6	[1, 768, 20, 20]	--
│ │ └─Conv: 3-10	[1, 384, 20, 20]	295,680
│ │ └─ModuleList: 3-11	--	--
│ │ └─Conv: 3-12	[1, 768, 20, 20]	1,181,184
├─YoloNASPANNeckWithC2: 1-2	[1, 96, 80, 80]	--
│ └─YoloNASUpStage: 2-7	[1, 192, 20, 20]	--
│ │ └─Conv: 3-13	[1, 192, 40, 40]	74,112
│ │ └─Conv: 3-14	[1, 192, 80, 80]	37,248
│ │ └─Conv: 3-15	[1, 192, 40, 40]	332,160
│ │ └─Conv: 3-16	[1, 192, 20, 20]	147,840
│ │ └─ConvTranspose2d: 3-17	[1, 192, 40, 40]	147,648
│ │ └─Conv: 3-18	[1, 192, 40, 40]	110,976
│ │ └─YoloNASCSPLayer: 3-19	[1, 192, 40, 40]	2,595,716
│ └─YoloNASUpStage: 2-8	[1, 96, 40, 40]	--
│ │ └─Conv: 3-20	[1, 96, 80, 80]	18,624
│ │ └─Conv: 3-21	[1, 96, 160, 160]	9,408
│ │ └─Conv: 3-22	[1, 96, 80, 80]	83,136
│ │ └─Conv: 3-23	[1, 96, 40, 40]	18,624
│ │ └─ConvTranspose2d: 3-24	[1, 96, 80, 80]	36,960
│ │ └─Conv: 3-25	[1, 96, 80, 80]	27,840
│ │ └─YoloNASCSPLayer: 3-26	[1, 96, 80, 80]	2,546,372
│ └─YoloNASDownStage: 2-9	[1, 192, 40, 40]	--
│ │ └─Conv: 3-27	[1, 96, 40, 40]	83,136
│ │ └─YoloNASCSPLayer: 3-28	[1, 192, 40, 40]	1,280,900
│ └─YoloNASDownStage: 2-10	[1, 384, 20, 20]	--
│ │ └─Conv: 3-29	[1, 192, 20, 20]	332,160
│ │ └─YoloNASCSPLayer: 3-30	[1, 384, 20, 20]	5,117,700
├─NDFLHeads: 1-3	[1, 8400, 4]	--
│ └─YoloNASDFLHead: 2-11	[1, 68, 80, 80]	--
│ │ └─ConvBNReLU: 3-31	[1, 128, 80, 80]	12,544
│ │ └─Sequential: 3-32	[1, 128, 80, 80]	147,712
│ │ └─Identity: 3-33	[1, 128, 80, 80]	--
│ │ └─Conv2d: 3-34	[1, 5, 80, 80]	645
│ │ └─Sequential: 3-35	[1, 128, 80, 80]	147,712
│ │ └─Identity: 3-36	[1, 128, 80, 80]	--
│ │ └─Conv2d: 3-37	[1, 68, 80, 80]	8,772
│ └─YoloNASDFLHead: 2-12	[1, 68, 40, 40]	--



```
│    │    └─ConvBNReLU: 3-38                  [1, 256, 40, 40]          49,664
│    │    └─Sequential: 3-39                  [1, 256, 40, 40]          590,336
│    │    └─Identity: 3-40                    [1, 256, 40, 40]          --
│    │    └─Conv2d: 3-41                      [1, 5, 40, 40]            1,285
│    │    └─Sequential: 3-42                  [1, 256, 40, 40]          590,336
│    │    └─Identity: 3-43                    [1, 256, 40, 40]          --
│    │    └─Conv2d: 3-44                      [1, 68, 40, 40]           17,476
│    └─YoloNASDFLHead: 2-13                   [1, 68, 20, 20]           --
│    │    └─ConvBNReLU: 3-45                  [1, 512, 20, 20]          197,632
│    │    └─Sequential: 3-46                  [1, 512, 20, 20]          2,360,320
│    │    └─Identity: 3-47                    [1, 512, 20, 20]          --
│    │    └─Conv2d: 3-48                      [1, 5, 20, 20]            2,565
│    │    └─Sequential: 3-49                  [1, 512, 20, 20]          2,360,320
│    │    └─Identity: 3-50                    [1, 512, 20, 20]          --
│    │    └─Conv2d: 3-51                      [1, 68, 20, 20]           34,884
==========================================================================================================================
Total params: 66,908,967
Trainable params: 66,908,967
Non-trainable params: 0
Total mult-adds (G): 64.78
==========================================================================================================================
Input size (MB): 4.92
Forward/backward pass size (MB): 1697.37
Params size (MB): 177.85
Estimated Total Size (MB): 1880.14
==========================================================================================================================
```

From the above summary diagram, YOLO-NAS_L, a version of the YOLO object detection architecture, is a large and intricate network with 66,908,967 trainable parameters. The model is divided into many stages: heads for jobs requiring detection, a neck for processing features, and a backbone for feature extraction. Specialized layers like YoloNASDFLHead and YoloNASCSPLayer are important parts of the model, demonstrating its sophisticated construction for effective object detection. The output forms for every layer are provided in the summary, showing how the data dimensions change as they move through the network. Given the intricacy of the model, the total computing load is substantial at 64.78 Giga multiply-adds operations. Large resource needs are implied by the anticipated memory footprint for input, forward/backward passes, and parameters. Understanding the YOLO-NAS_L model's design and resource needs is made easier with the help of this summary. It aims in determining if the model is appropriate for implementation in diverse settings, considering elements such as memory limitations and processing power.

Later, the model training session on YOLO (You Only Look Once) object detection is performed. Which is concentrated on training regimen on a single GPU, a sizable model (66.91M parameters), standard batch processing, and a customized learning rate and weight decay configuration, indicating a strategy that strikes a compromise between computing resource limitations and model complexity.

YOLO-NAS Vs YOLOv8

In contrast to previous versions, YOLO-NAS has a unique quantization-friendly basic block that improves quantization performance. With the help of this new block, YOLO-NAS can operate more efficiently and with greater accuracy. YOLOv8 is different from earlier YOLO models in that it uses a transformer-based architecture. Improvements in accuracy and performance have resulted from this creative design. But YOLOv8 lacks the quantization-friendly basic block seen in YOLO-NAS.

RESULTS:

For the Evaluation Criteria, the metrics we used for the project are precision, recall and F1 score. The primary assessment metric that we employed was Mean Average Precision, or mAP. The Intersection-Over-Union (IOU) criterion must be satisfied for the predicted bounding box to be classified as true positive.

- IOU is a measurement of the area that the ground truth bounding box and the expected bounding box share. It is calculated by dividing the intersection of the ground truth and the predicted bounding box by the union of those two boxes. An additional feature of the anticipated bounding box is a Confidence score.

$$\mathbf{IoU} = \frac{\mathbf{GBox} \cap \mathbf{PBox}}{\mathbf{GBox} \cup \mathbf{PBox}}$$

Where, GBox is the Ground Truth bounding box and PBox = Predicted bounding box

- Precision is calculated for evaluating machine learning models, which is important especially for tasks involving object detection and categorization. This is defined as the ratio is the proportion of actual positive predictions to all the model's positive predictions. To simplify, it measures how accurate the positive predictions are. Precision is calculated using the formula below.

$$\boldsymbol{precision} = \frac{\boldsymbol{True\ positive(TP)}}{\boldsymbol{True\ Positive(TP) + False\ Positive\ (FP)}}$$

Where TP is the instances in which the model accurately predicts the positive class and FP is When an actual negative instance is mistakenly predicted to be in the positive class by the model

- Recall is calculated to measures the percentage of real positive examples that the model accurately detects. To simplify, Recall evaluates the model's capacity to locate all relevant cases within a dataset, Recall is calculated using the formula below.

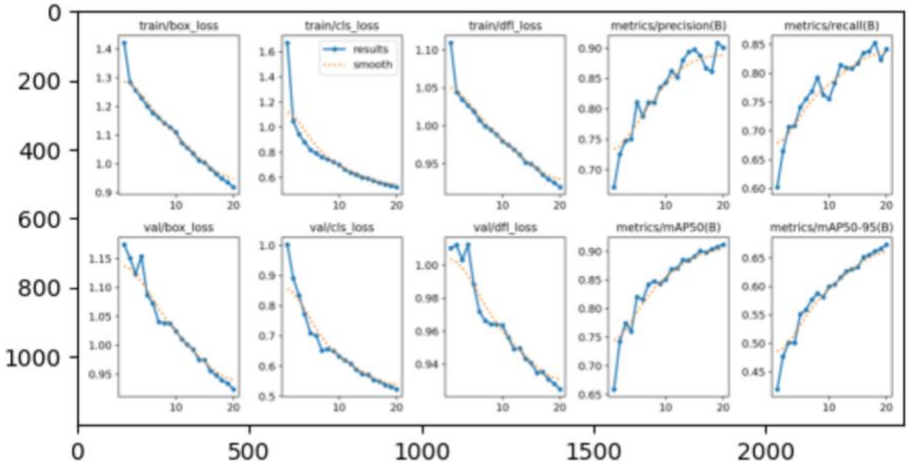
$$\boldsymbol{Recall} = \frac{\boldsymbol{True\ positive(TP)}}{\boldsymbol{True\ Positive(TP) + False\ Negative\ (FN)}}$$

Where TP is the instances in which the model accurately predicts the positive class and FN is When an actual positive instance is mistakenly predicted to be in the negative class by the model.

- F1 score is especially helpful when handling imbalanced datasets or when the cost of false positives and false negatives is significant. The F1 score offers a balanced assessment of a model's performance by combining precision and recall into a single metric.

$$\boldsymbol{F1\ score} = \mathbf{2}\ \boldsymbol{X}\ \frac{\boldsymbol{precision * recall}}{\boldsymbol{precision + recall}}$$

a. YOLOv8:



The above images gives the information about the loss, precision, recall and Map for 20 epochs. Below figure displays the information about the 9 images on a top with a 40x40-inch figure size in a 3x3 grid arrangement from a given directory. Every picture is loaded from the directory, placed into an axis-free subplot, and then rendered.

b. YOLO-NAS:

```
Testing: 100%|██████████| 16/16 [00:17<00:00,  1.03s/it]{'PPYoloELoss/loss_cls': 0.6406531,
 'PPYoloELoss/loss_iou': 0.37436315,
 'PPYoloELoss/loss_dfl': 0.3551168,
 'PPYoloELoss/loss': 1.3701328,
 'Precision@0.50': 0.10107392072677612,
 'Recall@0.50': 0.9644554257392883,
 'mAP@0.50': 0.8780940175056458,
 'F1@0.50': 0.18040421605110168}
```

From the above results, we say that the model's precision at 0.09874741733074188, the IoU threshold of 0.50. This shows the percentage of positively identified cases that were in fact accurate.The model's recall at the 0.50 IoU threshold, or 0.9650881886482239. This measure indicates the percentage of true positives that were appropriately detected.
mAP the important parameter, which represents the total precision-recall balance, is used in object detection. The mean average precision at an IoU threshold of 0.50. is 0.8802666664123535. The F1 score is a statistic that provides a balance between precision and recall, calculated as the harmonic mean of the two. The F1 score at an IoU threshold of 0.50 is 0.17656172811985016.

CONCLUSION:

In this work, in order to identify traffic images from various nations, we have primarily assessed two iterations of YOLO-based object detection models. When comparing the performance of YOLOv8 and YOLO-NAS under various working settings, YOLO-NAS emerged as the top performer, outperforming the others in terms of both overall and class-wise ratings. New YOLO-based model variations may appear in the future and perform even better than the ones we looked at here. We intend to gradually add more models to the collection, including those with varying architectural styles, as well as additional samples and classes, and additional analytical measures for a more thorough assessment.

REFERENCES:

- **https://doi.org/10.48550/arXiv.2201.07706**
- **https://www.researchgate.net/publication/346005917_Object_Detection_and_Tracking_Algorithms_for_Vehicle_Counting_A_Comparative_Analysis**
- **https://ieeexplore.ieee.org/document/8611986**
- R. A. Hadi, G. Sulong, and L. E. George, "Vehicle detection and tracking techniques: a concise review," arXiv preprint arXiv:1410.5894, 2014 **https://arxiv.org/abs/1410.5894**
- **https://medium.com/analytics-vidhya/object-detection-and-deep-learning-identify-google-streetview-image-content-using-yolo-in-r-bca631aa0bf3**
- **https://link.springer.com/article/10.1007/s11042-022-13644-y**
- Dataset Link: **https://universe.roboflow.com/fsmvu/street-view-gdogo/dataset/3**