Traffic Analysis and Emergency Lane

Utilizing Computer Vision

17 – 9 – 23

Machine Learning and Content Analytics

Prof. Papageorgiou Haris
As. Prof. Perakis Georgios

Team Zulu
MSc Business Analytics Part Time 2023

Table of Contents

Introduction

In the modern context of highroads, a growing issue arises as more and more vehicles deviate from the intended purpose of the emergency lane. This unauthorized behavior not only poses a significant safety concern but also significantly contributes to worsening traffic congestion. When drivers, motivated by personal convenience, improperly use the emergency lane, they not only violate established traffic norms but also undermine the fundamental purpose of this dedicated safety zone.

The emergency lane, a crucial component of contemporary road infrastructure, plays a pivotal role in providing an unobstructed pathway for the smooth movement of emergency vehicles during critical situations. At the same time, it serves as a safe space for stranded drivers, giving them a secure area to address vehicle issues without disrupting the regular flow of traffic. However, the regrettable misuse of this haven by some drivers disrupts its intended function, potentially leading to delays in emergency response times and endangering lives.

Moreover, the repercussions of this audacious behavior ripple through the broader traffic ecosystem. The improper occupation of the emergency lane forces drivers in the regular traffic lanes to react, causing a chain reaction of slowdowns and increased accident risks. This resulting congestion fosters frustration among drivers, encourages road rage incidents, and further compromises safety conditions on highroads.

The primary aim of this paper is to develop an innovative application designed to enhance road safety and traffic management. Our research focuses on building an intelligent system capable of classifying vehicles observed on highways, identifying and categorizing vehicles illegally occupying the emergency lane, and providing real-time traffic analysis.

Our application stands as a testament to the potential of advanced computer vision and machine learning techniques in contemporary transportation challenges. As per the recommendations set forth in the YOLO NAS documentation, we employ cutting-edge object detection and classification algorithms, enabling us to discern various vehicle types. Furthermore, our application extends its capabilities by integrating deep learning models, such as YOLO-NAS-L model, renowned for their efficiency and precision in real-time object detection tasks.

By developing this intelligent traffic management system, we aim to contribute to safer and more efficient highway operations while minimizing the misuse of emergency lanes. This research showcases the potential of emerging technologies in addressing critical issues in modern transportation infrastructure. Additionally, we implemented a real-time traffic analysis algorithm, which counted the different classes of vehicles on the highroads and in the emergency lane, further enhancing the utility of our project.

Vision and Goals

In the ever-evolving landscape of transportation and infrastructure, our project carries a profound vision and a set of ambitious goals aimed at redefining the way we manage traffic on our highways.

Enhancing Road Safety
Our foremost goal is to enhance road safety for all highway users. By accurately classifying vehicles and identifying those encroaching on the emergency lane, our system provides a critical layer of surveillance that can help prevent accidents and ensure the smooth flow of traffic. We aspire to make highway travel not just convenient but, above all, safe.

Improving Traffic Efficiency
Traffic congestion is a persistent challenge on highways worldwide, leading to time wastage, fuel consumption, and frustration. Through real-time traffic analysis and data-driven insights, our system aims to optimize traffic flow. By understanding the patterns of vehicle movement, we can provide recommendations to reduce congestion and minimize delays.

Mitigating Emergency Lane Misuse
The misuse of emergency lanes is not only a traffic violation but also a potentially life-threatening act. Our project takes a strong stance against such misuse by promptly identifying and classifying vehicles in these lanes. By doing so, we discourage unauthorized access and pave the way for emergency services to reach their destinations unhindered.

Showcasing Technological Innovation
Emerging technologies, such as computer vision, deep learning, and real-time data analysis, play a pivotal role in addressing the challenges of modern transportation. We envision our project as a showcase of the potential of these technologies to address critical infrastructure issues. By demonstrating their effectiveness in traffic management, we aim to inspire further innovation in the field.

Methodology

Our project was centered around the development of a robust vehicle recognition model for highroads, with the primary objective of classifying vehicles into distinct categories. To achieve this, we opted for the Super Gradients library from Deci-AI, specifically focusing on the cutting-edge YOLO-NAS model. This choice was influenced by the model's impressive accuracy and its remarkable speed in both training and making real-time video predictions. The YOLO-NAS model offers three different versions, small, medium, and large, prompting us to undertake the training of all three models for a comparative assessment to determine the most suitable one for our project.
To facilitate our training process, we initially sourced a substantial dataset from the web, comprising images of vehicles captured on various North American roads. This dataset also included a CSV file with detailed bounding box information for each annotated vehicle in the images. To enhance our dataset further, we enriched it with self-collected data, encompassing photos and videos of vehicles on Greek roads, with a particular focus on emergency vehicles that featured distinct colors do not present in the North American dataset. We combined the data sources and meticulously curated a consolidated dataset.

Afterward, we initiated the preprocessing stage, where we encountered data format disparities incompatible with the YOLO model. Specifically, we transformed bounding boxes into the YOLO

format, normalized the bounding boxes, and introduced image augmentations as a preparatory step before training commenced. Following this meticulous preprocessing, we initiated the training phase for the three models.
In evaluating their performance, we considered a spectrum of metrics, including the Mean Average Precision (MAP), loss function, loss classification function, and the Intersection over Union (IOU) loss. Post-evaluation, the large version of the YOLO-NAS model emerged as the standout choice, and we conducted rigorous testing on our test dataset. Finally, we assessed its real-world performance by applying the model to random images and videos captured on Greek highways, effectively harnessing its capabilities to recognize and classify vehicles in this dynamic and challenging context.

Data Collection

Publicly available dataset

For our project, we required a dataset consisting of distant photos captured from roads featuring a wide array of vehicle types. This necessity arose from our project's core objective, which is to develop an application capable of identifying vehicles in video footage and accurately classifying them into their respective categories. To meet this requirement, we acquired data from the MIO-TCD dataset. This dataset, among the largest of its kind, offers challenges in vehicle localization and identification. Hosted by the University of Sherbrooke and Miovision Inc. in Canada, it includes over half a million images from traffic cameras all over Canada and the United States by providing two distinct datasets: the Classification Dataset and the Localization Dataset, each tailored to different project requirements and application.

In our specific project, we made the deliberate choice to utilize the Localization Dataset which contains 137,743 images acquired at different times, different periods of the year and different weather phenomena. This dataset is particularly well-suited to our objectives because it contains high-resolution images that include one or more foreground objects. These attributes make it exceptionally suitable for classification and object detection projects, allowing us to extract nuanced information about the objects within the images.
The Localization Dataset not only offers rich and detailed imagery but also comes equipped with annotations for objects within the images. These annotations are categorized into eleven distinct labels, enhancing the dataset's utility and versatility. The labeled categories include:

1. Articulated truck
2. Bicycle
3. Bus
4. Car
5. Motorcycle
6. Motorized vehicle (i.e., vehicles that are too small to be labeled into a specific category)
7. Non-motorized vehicle
8. Pedestrian
9. Pickup truck
10. Single unit truck
11. Work van

This extensive labeling schema greatly simplifies the process of object recognition and classification within the dataset. Furthermore, it allows us to train and fine-tune our machine learning models to perform tasks such as object detection, classification, and even more complex applications like traffic analysis and monitoring.

Figure 1 - MIO-TCD Localization Dataset Examples

Manually created dataset

In our pursuit of developing an application tailored to the intricacies of Greek high roads, we decided to collect photos and videos directly from local roadways. The reason for this approach was the unique nature of Greek traffic. Greece has its own distinctive blend of vehicles, road conditions, and traffic patterns that differ significantly from those found in North America. Therefore, having data that accurately represents the Greek traffic landscape was essential for developing an application that could effectively cater to the intricacies of local traffic scenarios.

Moreover, our focus on including emergency vehicles and taxis in our dataset was a strategic choice born out of necessity. These vehicles possess distinct colors, markings, and vehicle types that are specific to the Greek context. Acquiring data that is both geographically and contextually relevant, not only enhances the application's performance but also ensures its safety and efficiency, especially when dealing with critical entities such as emergency vehicles.

Figure 2 - Manually Created Dataset Examples

To enhance the comprehensiveness and diversity of our dataset, we embarked on a multifaceted data collection journey. This involved both fieldwork and online research. We spent considerable time on a highway bridge, utilizing our mobile phone to capture video footage that featured a wide range of emergency vehicles, including ambulances, police cars, and police motorcycles, as well as taxis in various Greek real-world scenarios.

Additionally, we took advantage of our surroundings, capturing photos of these vehicles from the streets, further diversifying our image database. To bolster our dataset even further, we explored online resources, particularly YouTube, to access videos of Greek roads. From these videos, we extracted relevant frames that showcased the vehicles in different traffic scenarios.

Finally, to meet the specific requirements of our project, which encompassed both object detection and classification tasks, we annotated the vehicles within the self-collected data. This task was efficiently accomplished through the utilization of roboflow.com, a versatile annotation tool that significantly streamlined our annotation workflow. One of the notable advantages of roboflow.com was its ability to generate annotation files in the YOLO format, precisely tailored to our project's needs.

Dataset Overview

The Localization Dataset was initially divided into two distinct sets: the training dataset, comprising 110,000 images, and the test dataset, consisting of 27,743 images.
To explore and gain a deeper understanding of the dataset, we printed a few images with their corresponding bounding boxes. Notably, all images exhibited properly designed bounding boxes, accurately aligned with their respective classes. Furthermore, we sought to ascertain the dimensions of the images, which measured 720 pixels in width and 480 pixels in height. Additionally, we examined the CSV file containing the coordinates of the bounding boxes for each object, finding that the format adhered to the convention of (x1, y1) denoting the top-left corner and (x2, y2) representing the bottom-right corner of the bounding boxes.

In order to have a more detailed look at the localization dataset's composition and gain insights into object class distribution, we constructed a script that counts class occurrences within annotation files. We also calculated the total number of annotations and mapped class names for interpretability. The distribution for each class is shown in the following table:

Classes	Balance
Car	66.42%
Pickup truck	12.60%
Motorized vehicle	7.35%
Bus	3.01%
Articulated truck	2.65%
Work van	2.48%
Pedestrian	2.03%
Single unit truck	1.63%
Non-motorized vehicle	0.67%
Bicycle	0.64%
Motorcycle	0.52%

Table 1 - Localization Dataset Class Balance

Data processing

1. **Modification of bounding boxes format to YOLO format and Normalization**
   As a fundamental component of our data preprocessing, the first action was to employ a dedicated script to process a CSV file containing information about bounding boxes within our image dataset. The initial data format presented image IDs, class names, and bounding box coordinates (x1, y1, x2, y2) which represents the coordinates of the top left corner (x1, y1) and bottom right corner (x2, y2). However, this format needed to undergo a transformation to align with the YOLO (You Only Look Once) object detection framework (x_center, y_center, width_norm, height_norm)

   The newly formatted data featured class IDs, the central coordinates of bounding boxes (x_center, y_center), and the normalized width and height of these bounding boxes (width_norm, height_norm) in generated text files for each image. These text files contained the bounding box information in the YOLO-acceptable format. This reformatting process not only ensured compliance with the YOLO framework but also maintained consistency across bounding box coordinates, even when dealing with images of diverse dimensions.

2. **Class Mapping**
   To enhance our data preparation efforts, we created a class mapping dictionary. This dictionary formed a bridge between class names and unique class IDs. Each class name was assigned a specific integer ID.

3. **Creation of a Sample dataset**
   As we had already made the decision to introduce new classes specific to Greek emergency vehicles, we incorporated the self-collected data pertaining to these emergency vehicles. However, it's important to note that the emergency dataset comprised only 500 images, significantly fewer than the original dataset of 110,000 images.

   To ensure a more balanced and representative final dataset, we made the choice to select an initial sample of 1000 objects (vehicles) for each class from the initial dataset. This approach aimed to bring the final dataset closer to a proportional distribution, enhancing its overall quality and relevance for our project.

   Following the dataset creation, we proceeded to partition it into three subsets: 70% for training, 20% for evaluation, and 10% for testing. We also conducted an analysis of the updated class distribution percentages, which were found to be similar with the initial distribution, prior to the introduction of new categories and the random sampling per class. This observation underscores the effectiveness of our approach in maintaining the proportional representation of classes within the dataset, ensuring the integrity of the data for subsequent modeling and evaluation.

Object Class	Original Balance	Sample Balance
Car	66.42%	56.28%
Pickup truck	12.60%	10.44%
Motorized vehicle	7.35%	8.24%
Pedestrian	2.03%	4.45%
Bus	3.01%	4.30%
Articulated truck	2.65%	3.98%
Work van	2.48%	3.87%
Single unit truck	1.63%	3.27%
Non-motorized vehicle		1.63%
Bicycle		1.57%
Motorcycle		1.43%
Taxi		0.20%
Ambulance		0.14%
Police car		0.09%
Police motorcycle		0.06%
Firetruck		0.06%

Table 2 - Original and Sample Balance Comparison

Model Overview

YOLO-NAS is a neural network architecture designed for object detection tasks. It is an extension of the YOLO (You Only Look Once) family of real-time object detection algorithms. YOLO-NAS incorporates neural architecture search (NAS) techniques to automatically discover an optimal neural network architecture for object detection. The primary goal of YOLO-NAS is to find neural architectures that can achieve high accuracy in object detection while being computationally efficient and suitable for real-time applications.

The architecture of YOLO-NAS employs quantization-aware blocks and selective quantization for optimized performance. It introduces a new basic block that is friendly to quantization, addressing one of the significant limitations of previous YOLO models. YOLO-NAS leverages advanced training schemes and post-training quantization to enhance performance. This innovative iteration of YOLO-NAS not only attains state-of-the-art (SOTA) performance but also excels in the critical domain of accuracy-speed performance, surpassing the capabilities of its predecessors, including YOLOv5, YOLOv6, YOLOv7, and YOLOv8.

The performance of all existing models in terms of accuracy (measured as mAP - Mean Average Precision) and latency (measured in milliseconds) is visually presented in the graph below. This graphical representation vividly illustrates the supremacy of YOLO-NAS when compared to the other models, highlighting its exceptional balance of high accuracy and low latency, a key indicator of its superior real-time object detection capabilities.

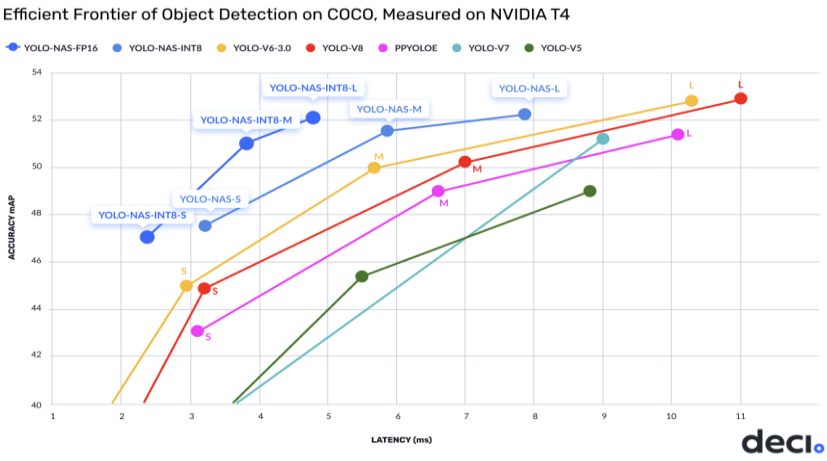

Figure 3 - YOLO Models Performance Comparison

YOLO-NAS has been released in three versions: S, M, and L. The models differ in terms of their mAP latency values, as shown in the table below:

Model	mAP	Latency (ms)
S – Small	47.5	3.21
M – Medium	51.55	5.85
L – Large	52.22	7.87

Table 3 - YOLO NAS Model Variations

In the context of Convolutional Neural Networks (CNNs), filters are learnable parameters that are used to extract features from data. Filters are 2-dimensional matrices/arrays, commonly square in size, that are applied to an input image to create a feature map that summarizes the presence of detected features in the input. In YOLO-NAS, the small, medium, and large models are defined based on the number of filters in the final convolutional layer of the network. The small model has 128 filters, the medium model has 256 filters, and the large model has 512 filters.
Filters can be handcrafted, such as line detectors, but the innovation of CNNs is to learn the filters during training in the context of a specific prediction problem. The filters in CNNs detect spatial patterns such as edges in an image by detecting the changes in intensity values of the image. In YOLO-NAS, the small, medium, and large models are defined based on the number of filters in the final convolutional layer of the network. The difference between these models is their capacity to detect objects of different sizes.

Training

Training Methodology

Transfer Learning Approach: Instead of building the object detection and classification model entirely from scratch, we adopted a transfer learning approach. This technique involves taking advantage of pre-trained models with weights that have been fine-tuned on extensive datasets. Pre-trained Model on COCO Dataset: Our training approach relied on a pre-trained model that had been learned from a massive dataset called Common Objects in Context (COCO) by Microsoft. This dataset contained over 330,000 pictures with around 880,000 descriptions. The model had already 5 classes that we were interested in: cars, buses, trucks, motorbikes, and bicycles. So, we took this pre-trained model as our starting point and fine-tuned it to work well for our specific tasks of spotting and classifying objects. This process helped us build a powerful model without starting from scratch.

Addition of Custom Classes: To adapt the model for our specific requirements, we expanded its capabilities by incorporating our own classes. These additional classes, as previously discussed, were designed to identify and classify objects relevant to our project, such as Greek emergency vehicles with distinct characteristics.

Training with Our Dataset: With the pre-trained model as the basis, we proceeded to fine-tune and adapt it further to accurately detect and classify the newly introduced classes. Our dataset containing vehicles in various scenarios, including those on Greek roads, allowed the model to learn and adapt to the unique characteristics of the new classes.

Leveraging Early Stopping for Training Efficiency
Another crucial technique we utilized during our model training was early stopping. Early stopping is a valuable technique used in training machine learning models, including deep learning models like ours. Its primary purpose is to improve training efficiency and model generalization by preventing overfitting, a situation where a model becomes too specialized in learning the training data but performs poorly on unseen data. The mechanism behind early stopping is straightforward: it continuously monitors the model's performance on a validation dataset during training. When this performance starts to degrade, such as when validation loss increases or accuracy decreases, early stopping intervenes to halt training, ensuring that the model doesn't overfit.

This approach to training is highly advantageous as it doesn't require a fixed number of training epochs but dynamically determines the optimal stopping point based on validation performance. In our specific case, we employed early stopping by closely tracking the Mean Average Precision (mAP) metric on the validation dataset. If, for three consecutive epochs, the model failed to exhibit significant improvements in mAP, we triggered early stopping. This tailored approach allowed us to optimize training efficiency, avoid overfitting, and strike the right balance between model performance and effectiveness, particularly in object detection and classification tasks for our project.

Metrics

Mean Average Precision (mAP): Mean Average Precision, or mAP, is a vital metric for evaluating the accuracy of object detection models. It quantifies how well a model identifies and localizes objects within images. mAP calculates the average precision across all object classes, considering both precision (how accurate the model's positive predictions are) and recall (how many actual objects are correctly identified). This comprehensive metric offers a single score that reflects the overall performance of an object detection model, making it invaluable for model selection and optimization.

Loss Function: The loss function is a critical component in training deep learning models, including object detectors. It measures the difference between the model's predictions and the ground truth during training. The goal is to minimize this difference, guiding the model towards accurate predictions. In YOLO object detection, the PPYOLOELOSS is a variant of the YOLO loss function used in the Paddle-Paddle deep learning framework. PPYOLOELOSS is a combination of several loss functions, including objectless loss, classification loss, and bounding box regression loss. Objectless loss is used to determine whether an object is present in an image, while classification loss is used to classify the object. The bounding box regression loss is used to predict the location of the object in the image.

Classification Loss: The loss classification function focuses specifically on the accuracy of class predictions within object detection. It evaluates how well the model assigns the correct class labels to detected objects. This metric ensures that the model not only detects objects but also identifies them correctly, which is crucial for tasks like image and object classification. By minimizing the loss classification function, the model becomes more adept at recognizing and categorizing objects accurately.

Intersection over Union (IoU) Loss: IoU loss is essential for precise object localization in object detection tasks. It measures the spatial alignment between predicted bounding boxes and ground truth. The IoU score is the ratio of the overlapping area between these boxes to their union. During training, minimizing IoU loss encourages the model to predict bounding boxes that closely match the actual object locations. This enhances the accuracy of object detection and ensures that objects are correctly localized within images, making IoU loss a key component for robust detection models.

Training Parameters Analysis

Continuing with the training process is a crucial step in creating a strong and accurate model for spotting and categorizing objects, especially when it comes to our YOLO NAS model. To get a better understanding of this important part, let's take a closer look at the settings we've carefully chosen to make sure our model works its best.

1. **Silent Mode (silent_mode)**

We set this to False to allow progress information to be printed to the console during training. It helps us monitor the training process.

2. **Average Best Models (average_best_models)**

Enabling the True option allows us to save the average of the best models during training. It's useful for model selection and achieving better model performance.

3. **Warm-Up Mode (warmup_mode):**

"linear_epoch_step" was chosen as the warm-up mode. This means that the learning rate will increase linearly during the warm-up period, gradually adapting the model to training data.

4. **Warm-Up Initial Learning Rate (warmup_initial_lr):**

A low initial learning rate of 1e-6 during warm-up ensures that the model starts with small weight updates to prevent instability.

5. **Warm-Up Epochs (lr_warmup_epochs):**

Explanation: A small number of warm-up epochs (3) allows the model to gently transition into the main training phase.

6. **Initial Learning Rate (initial_lr):**

Explanation: A higher initial learning rate of 5e-4 provides an effective starting point for training, balancing rapid convergence with stability.

7. **Learning Rate Mode (lr_mode):**

The Cosine learning rate scheduler is a type of learning rate schedule that drops the learning rate in the form of a sinusoid. It starts with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again.
Using a Cosine learning rate scheduler can help improve the performance of the model by effectively adjusting the learning rate throughout the training process.

8. **Cosine Final LR Ratio (cosine_final_lr_ratio):**

This value determines the final learning rate ratio. Setting it to 0.1 ensures a gradual decrease in the learning rate toward the end of training.

9. **Optimizer (optimizer): Adam**

Optimizer Adam, which stands for Adaptive Moment Estimation, is a popular optimization algorithm used in deep learning. It adjusts the parameters of a neural network in real-time to improve its accuracy and speed. Adam combines the features of two other extensions of stochastic gradient descent, AdaGrad and RMSProp. The algorithm computes individual adaptive learning rates for different parameters based on estimates of first and second moments of the gradients. This means that it adapts the learning rate of each parameter based on its historical gradients and momentum.
In the context of YOLO NAS model training, the Adam optimizer helps adjust the weights in an iterative way during training. It's particularly useful because it's computationally efficient, requires little memory, is invariant to diagonal rescale of the gradients, and is well-suited for problems that are large in terms of data and parameters.

10. **Optimizer Parameters (optimizer_params):**

A weight decay of 0.0001 is applied to control overfitting and improve model generalization.

11. **Zero Weight Decay on Bias and Batch Norm (zero_weight_decay_on_bias_and_bn):**

Enabling True option sets weight decay to 0 for bias and batch normalization layers, potentially enhancing model performance.

12. **Exponential Moving Average (ema):**

EMA is utilized during training to stabilize training and improve the model's final performance.

13. **EMA Parameters (ema_params):**

EMA's decay rate is set to 0.9 with a decay type of "threshold" to achieve the desired smoothing effect.

14. **Maximum Epochs (max_epochs):**

The variable EPOCHS defines the maximum number of training epochs. A higher value, such as 300, allows for extended training.

15. **Mixed Precision (mixed_precision):**

Enabling mixed precision training leverages lower-precision data types to accelerate training without compromising model accuracy.

16. **Loss Function (loss):**

The "loss" parameter specifies the loss function used during training. In this case, it's set to PPYoloELoss, which is a custom loss function tailored for YOLO models with the below sub-parameters.

- **use_static_assigner** False: This indicates that a dynamic assigner is used during training rather than a static one. A dynamic assigner adapts to different object scales and aspect ratios, making it suitable for object detection tasks with varying object sizes.
- **num_classes**: The number of classes is determined by the length of classes in the dataset. This ensures that the loss function accounts for the correct number of object classes in the classification task.
- **reg_max** 16: The "reg_max" parameter specifies the maximum regression range for bounding boxes in pixels. Setting it to 16 allows the model to predict bounding boxes accurately even for objects of various sizes, as the predicted coordinates can vary within this range.

17. **Validation Metrics (valid_metrics_list):**

The "valid_metrics_list" parameter defines the set of metrics and criteria used to evaluate the performance of the YOLO NAS model during the validation phase of training. It consists of two distinct detection metrics configurations:

- **DetectionMetrics_050:** This configuration defines a set of evaluation criteria for assessing object detection performance with a primary focus on precision and recall.
    - **score_thres** 0.1: The confidence score threshold specifies the minimum level of confidence required for a detection to be considered valid. Predictions with confidence scores below this threshold are disregarded.
    - **top_k_predictions** 300: During evaluation, the model's top 300 predictions per image are considered, allowing for comprehensive analysis of detection quality.
    - **num_cls**: The number of classes considered in the evaluation is determined by the length of classes present in the dataset, ensuring that class-specific performance metrics are calculated.
    - **normalize_targets** True: This setting indicates that targets, or ground truth annotations, are normalized during evaluation, contributing to the consistency and fairness of the evaluation process.
    - **post_prediction_callback**: This callback function defines post-processing steps to refine predictions further.
- **DetectionMetrics_050_095:** This second configuration builds upon the previous one and extends the evaluation criteria to encompass an additional measure of detection

quality, specifically focused on evaluating detections with higher intersection-over-union (IoU) thresholds.
    - **score_thres**: 0.1: Like the first configuration, this sets the confidence score threshold for valid detections.
- **top_k_predictions**: 300: It considers the top 300 predictions per image for evaluation.
- **num_cls**: The number of classes considered remains consistent with the dataset's class count.
- **normalize_targets**: True: Targets are normalized for fairness and consistency.
- **post_prediction_callback**: The same post-processing callback function is applied to refine predictions.

Model Selection

Yolo NAS Small

In this section, we evaluate the performance of the YOLO-NAS Small model, trained with 50 epochs and 1000 samples per each class in our dataset. We will report the train loss, the valid loss, the learning rate (lr), the valid mAP@0.5:0.95, the train loss_cls, the valid loss_cls, the train loss_iou and the valid loss_iou. The graphs of the evolution of these metrics are depicted in the following figure.

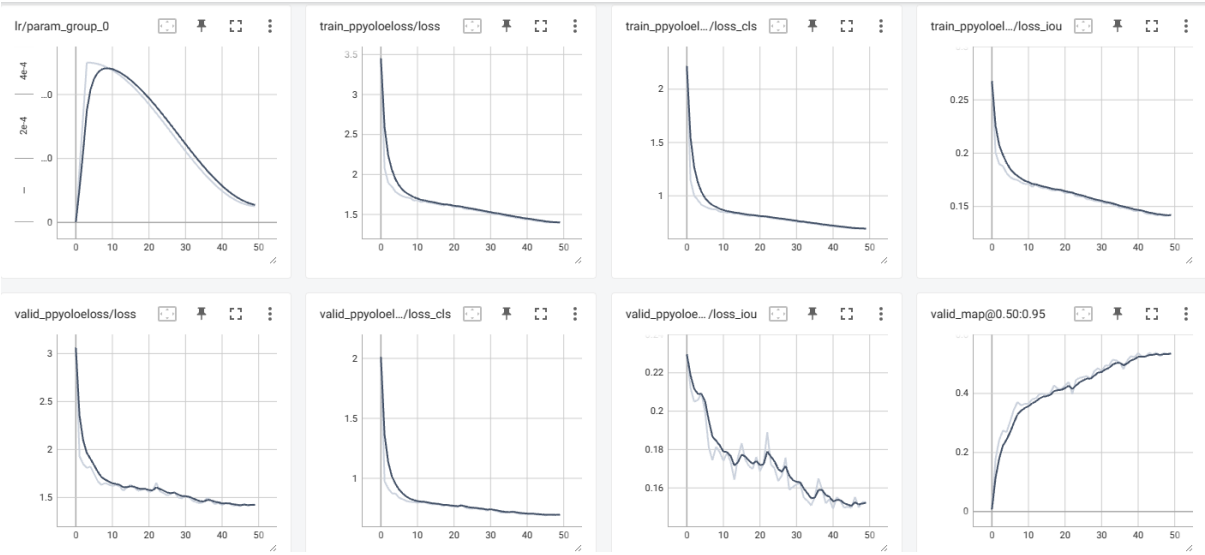

Figure 4 - Yolo-NAS S Metrics. 1000 Items per Class & 50 Epochs

Based on the above, it appears that our model has performed well in terms of training and validation losses. The training and validation losses have decreased steadily over time, indicating that our model is learning from the data. Although, the learning rate (lr) seems to start decreasing after the 10th epoch, which seems that our model is learning at a smaller rate after this point.

Regarding mAP@0.5:0.95, we have set a threshold of 0.475 for this metric, based on the mAP score that YOLO-NAS Small already had. Comparing this threshold with the value obtained by our model (0,537) we identify that we have achieved satisfactory performance.

To determine whether our model has overfit or not, we compare the training and validation losses and we identify that there is no significant difference between these metrics, so we can assume that our model has not overfit.

Finally, regarding the overall performance of this model, we can identify that the model seems to improve very slowly as the epochs go by. Furthermore, in the evolution of mAP@0.5:0.95, we can identify that the line starts being straight, which means that it starts not improving. This is an indicator that more epochs may not be helpful in this case. However, by increasing our dataset (and especially the emergency dataset) we may have a better performance in our training.

Yolo NAS Medium

In this section, we evaluate the performance of the YOLO-NAS Medium model trained with 50 epochs and 1000 samples per class in our dataset. We report the train loss, the valid loss, the lr, the valid mAP:0.5:0.95, the train loss_cls, the valid loss_cls, the train loss_iou and the valid loss_iou. The graphs of the evolution of these metrics are depicted in the following Figure.

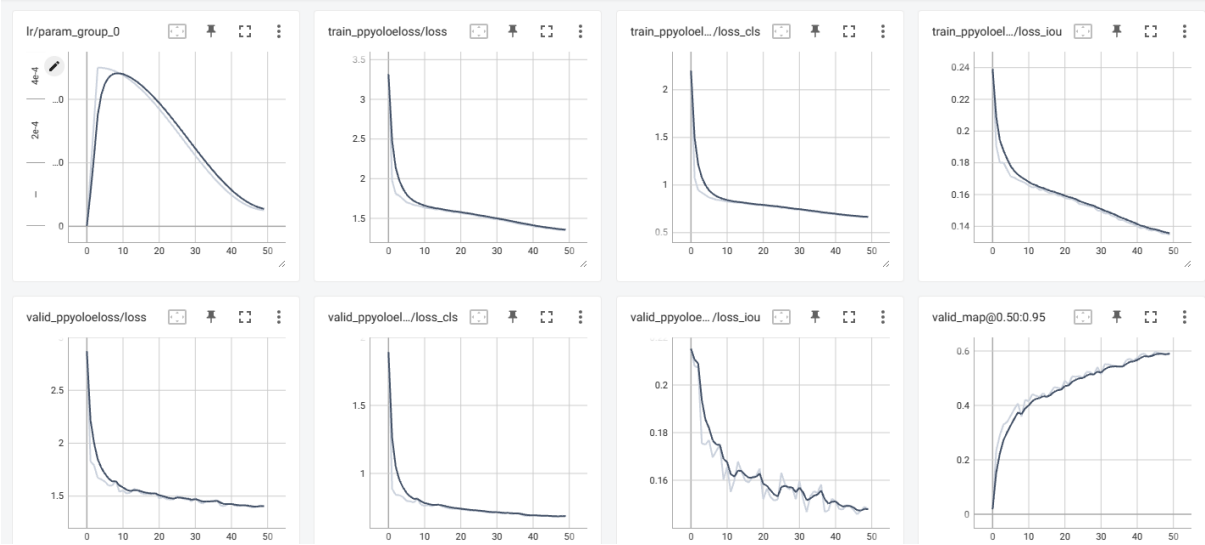

Figure 5 - Yolo-NAS M Metrics. 1000 Items per Class & 50 Epochs

Based on the above, it seems that our model has performed well in terms of training and validation losses. The training and validation losses have decreased steadily over time, which indicates that our model is learning from the data. Again, the learning rate (lr) seems to start decreasing after the 10th epoch, which seems that our model is learning at a smaller rate after this point.
Regarding mAP@0.5:0.95, we can determine that our model has achieved a good performance since the corresponding score is 0.59. This means that our model has managed to overcome the initial score of mAP@0.5:0.95 which is 0.515.

In terms of overfit, it seems that this does not take place in this model, since again there is no significant difference between training and validation losses, so we can assume that our model has not overfit.

Again, regarding the overall performance of this model, we can identify that the model seems to improve very slowly as the epochs go by. The evolution of mAP@0.5:0.95 is increasing, which means that more epochs may be helpful for this model. Finally, again, by increasing our dataset (and especially the emergency dataset) we may have a better performance in our training.

Yolo NAS Large

In this section, we evaluate the performance of the YOLO-NAS Large model, trained with 50 epochs and 1000 samples per each class in our dataset. Again, we report again the same metrics in the following graphs.

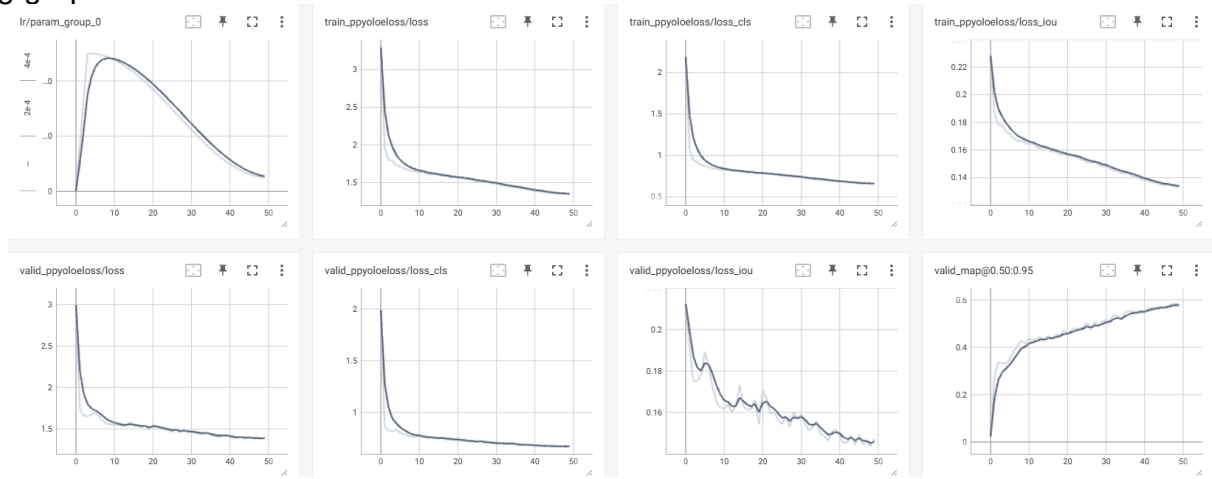

Figure 6 - Yolo-NAS L Metrics. 1000 Items per Class & 50 Epochs

Based on the above graphs, it seems that our model has performed well in terms of training and validation losses. The training and validation losses have decreased steadily over time, which indicates that our model is learning from the data. Again, the learning rate (lr) seems to start decreasing after the 10th epoch, which seems that our model is learning at a smaller rate after this point.
Considering the initial score of mAP@0.5:0.95, which is 0.522, we can say that our model has achieved a great performance with a score of 0.578.

Having no significant difference between training and validation losses, again we can assume that our model does not overfit.

Again, regarding the overall performance of this model, we can identify that the model seems to improve very slowly as the epochs go by, since the learning rate has decreased significantly. The evolution of mAP@0.5:0.95 is increasing, which means that more epochs may be helpful for this model. Finally, the enhancement of the dataset (and especially the emergency dataset) may be important for a better training performance.

Final Model and Fine Tuning

Since we observed that the YOLO NAS L model performed the best compared to the others, we decided to select it and fine tune it to pick the best version of it. Given that the sample size was the same for all three models (1000 per class) and that with increasing the filters of the network by choosing the Medium over the Small and the Large over the Medium models we did not see any significant changes in the performance we decided to:

1- increase the size of the training data.
2- implement early stopping.

YOLO NAS L v1: 1000 Samples and Early Stopping

We started version 1 with the base large model with 1000 samples and added early stopping. As seen from the following plots, the training stopped at the 15th epoch as no improvement in the mAP was observed. Although the learning rate started to drop slightly, the rest of the metrics had indications of potential improvement.

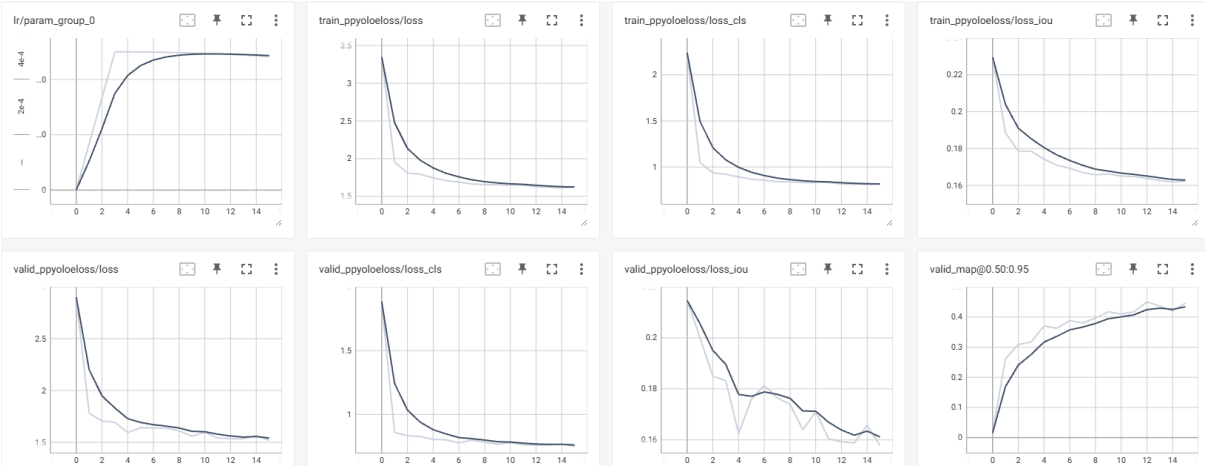

Figure 7 - YOLO NAS L Metrics. 1000 Samples per Class & 50 Epochs with Early Stopping

The mAP@0.5:0.95 was equal to 0.44 which was lower than the threshold of 0.475 and 0.522 which was the mAP without early stopping. This led to the conclusion that early stopping stopped the training as intended but not in the optimal epoch.

By default, early stopping helps to preserve overfitting which can also be validated by the plots seen above.

Given all the above, we concluded that by adjusting the early-stopping threshold to more than 3 epochs, it would not significantly improve the performance as we had already trained a model for 50 epochs.

YOLO NAS L v2: 4000 Samples and Early Stopping

The fine-tuning process continued with version 2 where we picked a sample of 4000 objects per class. This was a significant increase in the dataset size, and we expected better overall performance. Since the training time was long enough, we initiated early stopping to find the optimal epoch without overfitting. Surprisingly, the training stopped in the 5th epoch. As seen from the following plots, the early stopping worked as intended and stopped training at the 5th epoch as no improvement in the mAP was observed. Although the learning rate started to drop slightly, the rest of the metrics had clear indications of potential improvement.

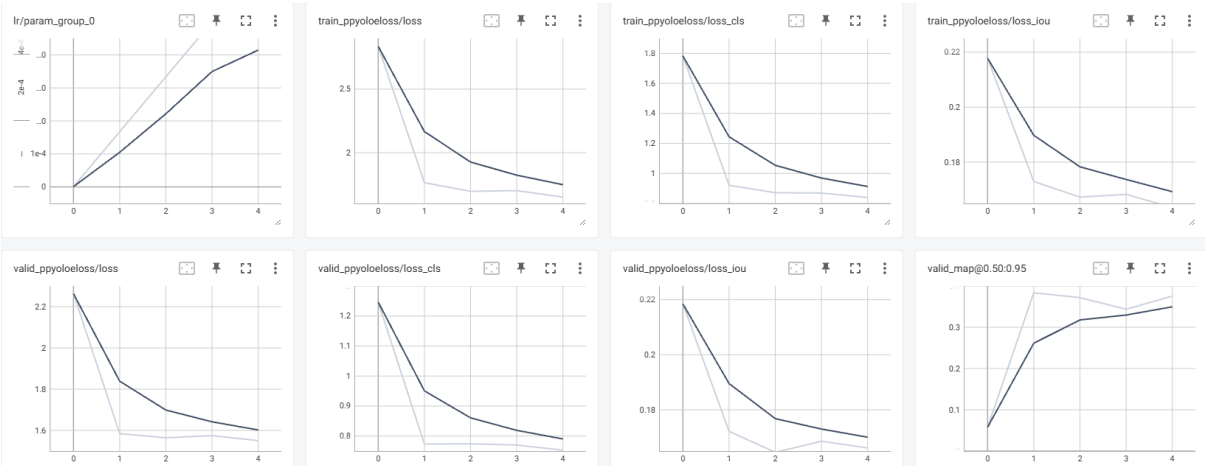

Figure 8 - YOLO NAS L Metrics. 4000 Samples per Class & 50 Epochs with Early Stopping

Given all the above, we concluded that by adjusting the early-stopping threshold to more than 3 epochs, it would improve the performance as we had already proof of that in version 1. On the other hand, we found early stopping to be a bit misleading as there may be some cases where the performance would not increase for several consecutive epochs but increases afterwards. This assumption can be validated in version 3 where the performance increased significantly after epoch 10 regardless of the drop before it.

YOLO NAS L v3: 4000 Samples without Early Stopping

To further improve the performance of our model, we plan to fine-tune the final model with version 2. For this purpose, we will evaluate the metrics for the YOLO-NAS Large model with 4000 items per class and train it for 50 epochs without early stopping. By doing so, we aim to explore the potential of this larger data set and assess its suitability for our specific task. The evaluation of the metrics will provide us with valuable insights into the model's performance and guide us in making informed decisions regarding its usage in our project. The corresponding graphs are shown in the following figure.

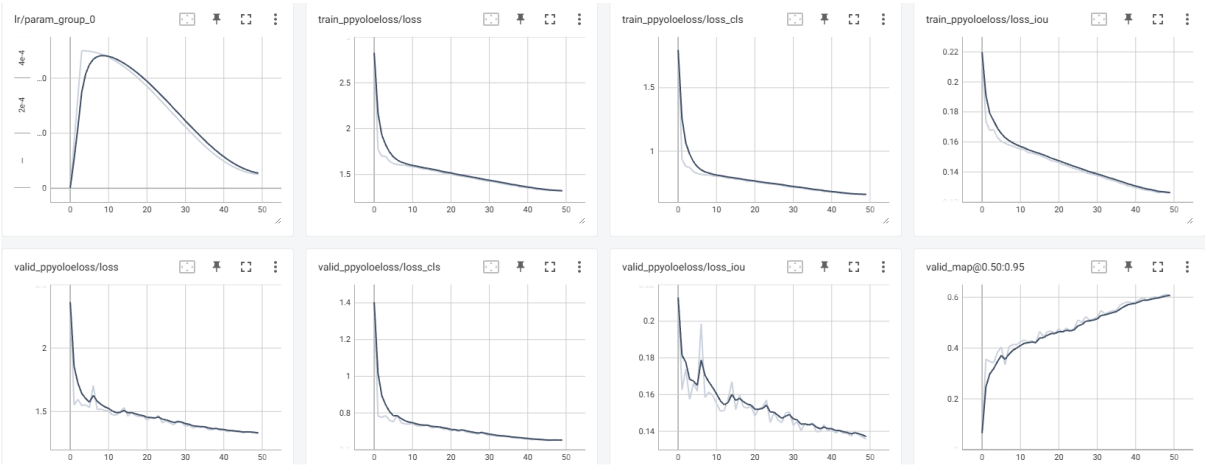

Figure 9 - Yolo-NAS L Metrics. 4000 Items per Class & 50 Epochs without Early Stopping

Based on the metrics of this model, we see that the learning rate has again a decreasing ratio. However, we identify that the lr is not decreasing as fast as in the previous models. Once again, the training and validation losses have decreased steadily over time, which indicates that our model is

learning from the data. Given that these metrics have no significant difference between training and validation losses, again we can assume that our model has not overfit.
Considering the initial score of mAP@0.5:0.95, which is 0.522, we can say that our model has achieved a great performance with a score of 0.61.
Here, the enhancement of the dataset has taken place (as we had mentioned in the previous chapter) and the performance of this model seems to improve as the epochs go by. Given the mAP@0.50:0.95 curve, we may say that the model needed more epochs in order to achieve the best possible scores. Although, we examine that the classification loss has not been improved, as its score is 0.65 (considering that the medium model has achieved a score of 0.66)

Model Evaluation and Inference

Evaluation

To evaluate the model's performance, we first used the test dataset to see how the model performs in unseen, out of sample data.

The results from the test output were as follows:

- **Precision@0.50:0.95:** 0.12
- **Recall@0.50:0.95:** 0.73
- **mAP@0.50:0.95:** 0.63
- **F1@0.50:0.95:** 0.20

The model demonstrated a high level of performance, with a mAP of approximately 63.30%, which is significantly higher than the threshold. This indicates that the model was able to accurately detect objects in the majority of instances within the test dataset.

The precision of the model at a range of IoU thresholds from 0.50 to 0.95 was approximately 12.30%, indicating that out of all predicted positives, around 12.30% were true positives. Low precision indicates a high number of false positives.

The recall of the model at the same range of IoU thresholds was approximately 73.67%, suggesting that the model was able to correctly identify around 73.67% of all actual positives in the dataset, indicating that the model is able to correctly identify a high percentage of vehicles present.

The F1 score, which is the harmonic mean of precision and recall, was approximately 20.23% at IoU thresholds ranging from 0.50 to 0.95. This metric is low due to the low precision of the model.

In conclusion, the object detection model exhibited strong performance in detecting objects within the test dataset, surpassing the predetermined threshold for satisfactory performance. However, there seems to be room for improvement in Precision and F1 Score to reduce the number of false positives and improve the overall performance.

Inference

To further evaluate the performance of the model, we have performed several tests on images from the test dataset, the web and on actual video from the Greek motorway. Through this process, we would like to investigate how our model performs on new/unseen data, in order to be able to understand the predictive capabilities of the model in actual scenarios.

Regarding the prediction from the test dataset, we identify that the model performed well on predicting the class of each vehicle. Below, we can find several examples with images from the test dataset and the corresponding prediction in the bounding boxes, along with the confidence level of each prediction.

Figure 10 - Inference on Test Images

From the above images, we can identify that our model is able to recognize the vehicles' classes, both for classes that have been derived from our initial dataset (Localization) and for the extra classes that we have added to our dataset. In the corresponding bounding boxes, we can see the predicted class for each vehicle, along with the confidence level for this prediction. We could also say that the model is able to recognize classes with high confidence, which means that it can accurately identify and locate the vehicles in the images.

Furthermore, we have performed the same exercise for predictions on a custom video with different confidence levels ranging from 0.5 to 0.9. In order to avoid cases of misclassification, we have performed several tests with the confidence level, and we identified that setting the confidence level to 0.7 reduces the cases of misclassification.

Figure 11 - Inference on Custom Video

From inference on the custom video, again we can see that our model performs great on identifying the classes of the vehicles. This time, we can see a little smaller confidence level on each vehicle prediction, compared to the ones of the images on Figure 10. In this snapshot from Greek Motorway, we could see several classes like Car, Work van and Motorcycle from vehicles that crossed the street. However, there are also several cases of misclassification, like the vehicle on the right side of the image, which has been tracked as a Police car, while it is a simple car. This is something that is expected in several cases, given that our model has a classification loss of 0.65, which is not as small enough in order to eliminate such cases.

Traffic Analysis

Implementation

In addition to object detection and classification, we integrated a specialized feature into our project for traffic analysis. This functionality allows us to accurately count vehicles passing through different lanes, including the emergency lane, and categorize them into their respective classes such as cars, buses, and taxis. The process involves breaking down a video into individual frames, and at the frame level, we perform the analysis and object detection, assigning vehicles to their appropriate classes. To facilitate this, we implemented two lines within the video—one that intersects with the emergency lane and another that spans all other lanes of the road. Each time a vehicle crosses one of these lines, the corresponding class counter increments to keep track of the vehicle count accurately. An essential component of our approach is the use of "trackers." These trackers identify and follow the same object across consecutive frames, ensuring that we don't count the same vehicle multiple times. It's worth noting that our system is customized to work seamlessly with our test videos, and we've designed the line parameters to align with the size and coordinates of the road in those specific videos. However, these parameters are highly adaptable and can be easily adjusted to accommodate different types of roads and the positions of respective security cameras, making our traffic analysis solution versatile and applicable to various scenarios.

Evaluation

To evaluate the performance of the traffic analysis implementation, we used the predictions made from the final model on the custom video with different confidence levels ranging from 0.5-0.9. Based on the results, we noticed that by lowering the confidence level, the model was able to identify more vehicles but there were cases of misclassification. By increasing the confidence, the model was able to identify more accurately the vehicle classes, but there were some vehicles not being classified at all. We also found cases where a vehicle was misclassified as an emergency vehicle while not being one.
Given the above observations, we decided that a confidence level of 0.7 was satisfactory as it did not misclassify non-emergency vehicles as such. Although this option sometimes makes the model unable to identify some classes, we preferred to have untracked some violations in Emergency Lane, rather than track violations of misclassified vehicles. This tradeoff means that we accept that the traffic analysis will not be 100% accurate on counting vehicles, but it ensures that a car violating the emergency lane will not be classified as an emergency vehicle and thus be excluded from the violators.

Results

The machine learning project was centered around the detection of vehicles on roads, utilizing the pretrained model YOLO NAS. This model was further trained on custom data through the process of transfer learning.

The results from the training and validation stages indicate that the model is highly sensitive, with a strong ability to correctly identify vehicles. This is evidenced by the high recall values obtained in both stages. However, the precision of the model is relatively low, suggesting that it often misclassified non-vehicles as vehicles.

The Mean Average Precision (mAP) is a key metric in evaluating the performance of object detection models. It combines both precision and recall into a single metric, providing a holistic view of the model's performance.

In this project, the mAP values obtained during validation and testing were quite promising, particularly at an Intersection over Union (IoU) threshold of 0.50. This suggests that the model was able to accurately detect a substantial proportion of vehicles in the images. However, when the IoU threshold was increased to 0.95, the mAP value decreased, indicating that the model's predictions were not as precise at this higher threshold.

In the testing phase, a similar trend was observed. The model continued to demonstrate high sensitivity but struggled with precision. This pattern suggests that while the model is adept at identifying vehicles in images, it also tends to generate a significant number of false positives.

These outcomes could be attributed to various factors such as the quality and diversity of the training data, the complexity of the model, or the choice of hyperparameters. Future work could explore methods for improving precision without significantly reducing recall. This might involve collecting more diverse and representative training data, fine-tuning the model architecture or hyperparameters, or applying post-processing techniques to filter out false positives.

Challenges and Improvements

The project encountered several challenges that significantly impacted our progress and required careful consideration and strategic planning for resolution. These challenges can be summarized as follows:

- **Dataset Unavailability:** A pivotal hurdle we confronted was the unavailability of the dataset website, which posed a substantial challenge. In response, we had to invest substantial time in identifying an alternative data source and subsequently uploading it to our Google Drive repository. This process proved to be time-consuming and introduced substantial delays to our project timeline.

- **Resource Costs**: Another critical factor that demanded our attention was the cost associated with essential resources, specifically GPU resources, and computational units within Google Colab. These resources were indispensable for effective model training, requiring more than 1,000 computational units on Google Colab Pro. Consequently, we needed to ensure consistent access to these resources while managing the associated expenses judiciously.

    - **Improvement**: To manage resource costs, we consider exploring alternative platforms that offer better or lower-cost GPU resources on a budget. Additionally, optimizing the model and code can help us reduce the computational resources required.

- **Training Time:** The time required for model training emerged as a critical consideration. Given the necessity for training our model over numerous epochs, meticulous planning and resource management became imperative to keep the project on track.

    - **Improvement**: One potential solution to reduce training time is to use more efficient models or training techniques.

- **Data Discrepancies:** Adapting our model to the differences between the TCD-Localization dataset and our custom dataset presented substantial challenges. This involved a thorough analysis of the data and iterative adjustments to our model to achieve satisfactory results.

    - **Improvement**: To handle discrepancies between different datasets, we consider using data augmentation techniques to make the model more robust to variations in the data.

- **GPU RAM Limitations:** The limitations on GPU RAM within the Google Colab environment created additional complexities. Specifically, it constrained our ability to increase the batch size parameter for training, thereby extending the overall training time. As a result, we had to optimize our code and employ resource-efficient strategies.

    - **Improvement:** To overcome GPU RAM limitations, we consider exploring new techniques, which allow for larger effective batch sizes without increasing memory requirements while finding a better GPU on a budget.

- **Lack of Data**

    - **Emergency Vehicle Data:** Capturing data related to emergency vehicles posed a unique and significant obstacle. These vehicles are relatively rare, demanding specialized handling during data collection and preprocessing to ensure adequate representation in our dataset.
    -
    - **Greek Vehicle Data:** Initially, we grappled with the absence of Greek vehicle data within our datasets. This scarcity limited our capacity to train and validate our model within the specific domain we aimed to target.

        - **Improvement**: For challenges related to specific types of data, such as emergency vehicles or Greek vehicles, we consider collaborating with local authorities or organizations who may have access to relevant data. Alternatively, synthetic data generation techniques could be used to augment the existing dataset.

- **Monitoring Challenges:** A time-consuming aspect of our project involved the continuous monitoring of vehicle behavior within training videos. The scarcity of emergency vehicles passing through the streets added complexity to our data collection efforts, requiring extensive data gathering to meet the project's requirements.

    - **Improvement:** To alleviate the burden of continuous monitoring, we are considering implementing automated monitoring and alerting systems. These systems can notify you when certain conditions are met, reducing the need for constant manual monitoring.

In summary, our project faced a multitude of challenges ranging from data-related issues and resource constraints to specific domain intricacies. Overcoming these challenges demanded meticulous planning, efficient resource utilization, and adaptability in our approach to ensure the successful development of our model.

Members/Roles

The project team was composed of four members. All team members are students enrolled in the MSc in Business Analytics program at Athens University of Economics and Business.
Our team is composed of individuals with diverse expertise and backgrounds, a strategic choice aimed at enhancing our project's efficiency.

1. Athanasios Alexandris (p2822202): Responsible for sourcing the initial dataset, collecting self-generated data, annotating, and meticulously preprocessing the data to meet the model's requirements, ensuring a robust foundation for the project. Also took charge of compiling the project report and presentation.

2. Ioannis Demertzis (p2822210): End to End Development and Project Management. Participation and coordination on every project's phase, specializing in designing, training and evaluating the model's performance.

3. Stamatia Pantazopoulou (p2822218): Data preprocessing, capture additional images to effectively enrich our dataset, annotate the images captured and involvement in Traffic Analysis and report.

4. Lampros Stroumpakis (p2822023): Models Training and Traffic Analysis; he has performed training in possible models, object detection in Video and development in the Traffic Analysis. Also, he took part in the model evaluation, and he has performed tests on the prediction capabilities of the outcome models.

15.TimePlan

- Search for a dataset: 22/5
- Finalize Project Decisions and agree on the plan: 22/5
- Find dataset to train the model: 23/5 - 30/5
- Align on the processes and datasets/models to be used: 31/5 - 6/6
- Exploration and Preprocessing: 7/6 - 20/6
- Create Extra Data: 21/6 - 27/6
- Development and Modelling: 28/6 - 18/7
- Import and Train the Model: 19/7 - 29/8
- Traffic Analysis: 30/8 - 5/9
- Evaluation and Fine Tuning: 6/9 - 14/9

References

1. Z. Luo et al., "MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization," in IEEE Transactions on Image Processing, vol. 27, no. 10, pp. 5129-5141, Oct. 2018, https://tcd.miovision.com/challenge/dataset.html doi: 10.1109/TIP.2018.2848705.

2. Aharon, S., et al. (2021). Super-Gradients [GitHub repository]. GitHub. https://github.com/Deci-AI/super-gradients

3. Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software]. https://roboflow.com