Utilizing Computer Vision Techniques to Address Urban Traffic Congestion

Aaron Moses
Software Engineering
Western University
London, Canada
Email: amoses22@uwo.ca

Rithik Kalra
Software Engineering
Western University
London, Canada
Email: rkalra24@uwo.ca

Scott Murray
Software Engineering
Western University
London, Canada
Email: smurr4@uwo.ca

***Abstract*—With increasing urbanization, traffic congestion poses many challenges to mobility and quality of life. This study investigates the utilization of computer vision techniques to address this issue by assessing street and intersection congestion within urban environments. Leveraging the Cityscapes dataset and advanced computer vision algorithms, our solution accurately gauges traffic busyness levels. Methodologically, we outline data preprocessing, feature extraction, and machine learning model training. Results demonstrate the system's real-time insights into traffic density and flow, aiding urban planners and traffic management authorities. The novelty lies in applying computer vision to tackle urban congestion, offering scalability and cost-effectiveness. Additionally, a project named "Street Sight" explores computer vision as a solution for autonomous driving and urban congestion, developing a convolutional neural network (CNN) model to classify objects on roads and introduce a "busyness" metric for traffic analysis. This project aims to advance autonomous driving technology and urban congestion through improved perception and navigation capabilities. Future work involves refining algorithms, exploring alternative datasets, and integrating models into real-world systems, contributing to urban scene understanding and addressing mobility challenges.**

I. Introduction

Traffic congestion is a growing challenge in urban areas, significantly impacting mobility and the overall quality of life [1]. Given that Canada's population has grown significantly in recent years and the government's immigration targets, this problem will only grow. With most people settling in condensed cities, the demand for efficient traffic management systems becomes increasingly important. Traditional methods of assessing traffic congestion often involve physical sensors and manual surveys, which are costly, time-consuming, and limited in scope. This study aims to explore the application of computer vision techniques to provide a more scalable and cost-effective solution.

Recent advancements in computer vision and machine learning have opened new avenues for analyzing complex urban environments [2],[3]. We aim to develop a computer vision system under the name of "Street Sight" that can accurately assess street and intersection congestion levels using visual data. With varying datasets, collections of urban street scenes with annotations will help enable our analysis. With this application of computer vision techniques, we will gauge traffic busyness levels.

Our steps include data preprocessing, feature extraction, and the training of machine learning models. Through processing the visual data and extracting relevant features, we will build an application capable of providing real-time insights into traffic density and flow patterns. These insights should be helpful for future traffic congestion studies or urban planners to devise strategies to combat congestion and improve mobility. This study may also prove helpful for future semantic segmentation studies in urban environments or autonomous driving projects.

As we move forward, our future work includes improving algorithms and exploring alternative datasets. By addressing the challenges of urban mobility and congestion, we aim to contribute to a deeper understanding of urban scenes and provide an improved solution for traffic management.

II. Background and Related Work

Computer vision excels at real-time tasks. It has been used across various domains and projects. In security, surveillance systems have used computer vision for facial recognition and even intrusion detection [4]. For crowd monitoring, computer vision has been used to aid in public safety by analyzing crowd density to prevent accidents in large gatherings. In traffic management, computer vision has been used to analyze traffic flow to counter congestion and prevent auto accidents. The events industry has also used computer vision for crowd control, enforcing safety regulations during events, detection of unattended bags and suspicious behavior [4]. In workplaces, some companies have employed computer vision for real-time monitoring hazard detection to ensure a safe work setting and compliance with safety protocols. The manufacturing sector has used computer vision for quality control, continuously monitoring production lines to immediately catch defects and oddities as they happen. Computer vision has been used for predictive maintenance to detect early signs of wear and tear to prevent downtime and ensure efficient operation.

While computer vision is a very powerful tool, it is highly reliant on extensive data analysis to recognize images, with deep learning and convolutional neural networks (CNNs) being functional pillars. Machine learning enables computers to learn from data without explicit programming from which CNNs can break down images into pixels with tags or labels and use convolutions to make predictions [4]. Recurrent neural

networks (RNNs) have been employed in video applications to draw connections between frames, expanding the abilities of computer vision systems.

Cityscapes is a vast dataset specifically designed for semantic segmentation in urban settings. It has the largest and most diverse dataset of street scenes that are accompanied by high-quality annotation. This dataset can be used to facilitate the development of an advanced computer vision algorithm designed for use in urban settings [5]. Notably, an off-the-shelf fully-convolutional network trained on Cityscapes outperforms other methods on competing datasets like KITTI and CamVid. There is still room for improvement as challenges remain and the current best performance levels are far from optimal, particularly in the instance-level task.

Urban planning research has turned its attention to the study of transportation congestion as cities get more and more populated. Conventional methods have depended on tangible infrastructure, including CCTV cameras and inductive loop detectors, which have limited coverage and are expensive. Advances in machine learning techniques, namely in computer vision, have resulted in a transition away from physical infrastructure and toward new technologies for traffic analysis and congestion management in recent years. Computer vision techniques have been applied across many aspects of traffic monitoring, including vehicle detection, traffic flow estimation, and congestion analysis [4]. The use of datasets like Cityscapes has helped to facilitate the development of algorithms capable of interpreting urban areas effectively. These algorithms, often based on convolutional neural networks (CNNs), have been effective in identifying and classifying objects, such as cars.

One area of focus within computer vision is semantic segmentation. This technique can be used to enable autonomous vehicles to better understand their surroundings. The overall safety and performance of autonomous transportation systems are enhanced by enhancing the navigational capabilities of autonomous vehicles.

Additionally, there has been interest in combining computer vision methods with more conventional data sources, such GPS, to offer a more comprehensive picture of the traffic conditions in the area. This multi-method approach allows for a better view of traffic patterns and more effective routing strategies.

Computer vision and machine learning have extensive applications in traffic and object recognition in autonomous cars. Tesla's Autopilot uses a range of sensors and cameras, including ultrasonic sensors in the bumpers, cameras mounted on the side, rear, and front of the car, and radar in the front. This is a great illustration of how this technology is used successfully. These sensors provide 360-degree sight, allowing for the detection of objects and vehicles at various speeds and distances. Tesla's onboard computer uses an NVIDIA Drive AGX Pegasus AI processor to process the data that is gathered [6]. Tesla also uses OpenCV for computer vision and image processing. In 2021, they abandoned their use of RADAR and LiDAR technology in favor of a vision-only strategy [7]. In a complex architecture known as Tesla's HydraNet, convolutional and recurrent neural networks (RNNs) are used in the autopilot system.

A less complex version of Tesla's HydraNet, YOLO (You Only Look Once) algorithms are being employed in autonomous driving software. Robotics, autonomous driving, and surveillance are just a few of the real-time applications that can benefit from YOLO's accurate and efficient object detection [8]. The YOLO algorithm is also used by surveillance and law enforcement systems to identify particular people or items. This allows for real-time identification of suspects and subsequent alert triggering or systematic monitoring of their activities. The versatility of the YOLO algorithm extends to various research domains as well. For instance, it aids in wildlife movement detection and target tracking, as well as in spotting patterns in geographic data and swiftly identifying terrain features in real-time [9],[10].

In conclusion, autonomous driving and traffic assessment have both exhibited significant potential for computer vision. By utilizing sophisticated algorithms and extensive datasets such as Cityscapes, creative and expandable solutions that outperform conventional approaches can be created. The promise for increased safety and efficiency in transportation is demonstrated by the application of computer vision in autonomous vehicles and traffic monitoring, as demonstrated by Tesla's Autopilot and YOLO algorithms. The use of computer vision in this industry is predicted to grow and improve as technology progresses, improving our ability to navigate and control traffic in urban areas.

III. Methods

A. Research Objectives

The project aims to determine a busyness level from a given street view using YOLOv5and YOLOv8 pre-trained models. To better evaluate the performance of the method, we will create a U-Net model with a ResNet-34 backbone encoder as the comparison.

The significance of creating the busyness level from the two models will greatly enhance the accuracy and effectiveness of real-time traffic indications. By comparing the outputs of these models, we can obtain a deeper understanding of traffic dynamics, enabling more precise and reliable predictions of congestion levels, and traffic flow patterns.

B. Research Methodologies

1) How does YOLO work?

YOLO processes an input image in a single forward pass, dividing it into an $S \times S$ grid of cells. Each grid cell predicts B bounding boxes and their corresponding C class probabilities. The bounding box prediction has 5 components: $(x, y, w, h, confidence)$ with the (x, y) coordinates representing the center of the box, relative to the grid cell location. These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size. The confidence reflects the presence or absence of an object of any class within the bounded box. There are in total $S \times S \times B * 5$ outputs related to bounding box predictions.

Class probabilities are predicted using the following condition $P(Class(i)|Object)$, ensuring bounded boxes with no object in them do not affect the prediction. The prediction vector for the output tensor is structured in the following manner tensor$(S \times S \times (B * 5 + C))$ [10],[11].

The YOLO models use a CNN like network structure with the following layer breakdown.

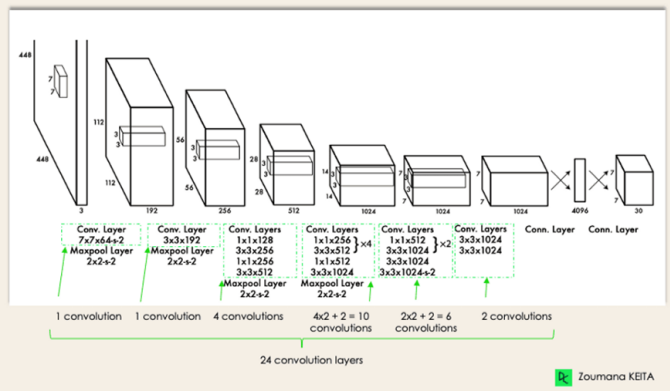

Fig. 1. YOLO model CNN Layer Structure [12]

This CNN architecture consists of convolutional layers, responsible for extracting features such as edges and textures, followed by downsample layers like max-pooling or strides to reduce the spatial dimensions of the feature maps while retaining important information. Intermediate layers then further refine feature representations, leading to higher-level abstractions. After several convolutional layers, YOLO includes fully connected layers to process the extracted features and generate predictions. These layers might be followed by activation functions like ReLU to introduce non-linearity. Finally, the output layer generates predictions, using the previously mentioned bounding boxes and class probabilities.

Non-maximum suppression is used in post-processing to refine predictions by removing redundant bounding boxes [12]. The following is pseudocode describing an implementation of NMS.

Algorithm 1 Non-Max Suppression

```
procedure NMS(B,c)
    B_nms ← ∅                                   Initialize empty set
    for b_i ∈ B do                              Iterate over all the boxes
        discard ← False                         Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for b_j ∈ B do                          Start another loop to compare with b(i)
            if same(b_i, b_j) > λ_nms then      If both boxes having same IOU
                if score(c, b_j) > score(c, b_i) then
                    discard ← True              Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not discard then                     Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
            B_nms ← B_nms ∪ b_i
    return B_nms                                Do the same procedure for remaining boxes and return the final list
```

Fig. 2. Non-Max Suppression Pseudo Code [12]

Various loss functions are used to determine the output accuracy of the model, with the primary goal of minimizing these loss functions during the training of the YOLO models.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

Fig. 3. Loss Function One [11]

This loss function computes the loss related to the predicted bounding box position (x, y). The equation computes a sum over each bounding box predictor j of each grid cell i. The obj returns a 1, if an object is present in grid cell i and the jth bounding box predictor is responsible for that prediction, else it returns a 0 [11].

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

Fig. 4. Loss Function Two [11]

This loss function computes the loss related to the predicted bounding box dimensions (w, h) in a similar manner to the first loss function. The error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this the square root of the bounding box's width and height is used [11].

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

Fig. 5. Loss Function Three [11]

This loss function computes the loss related to the confidence score C for each bounding box predictor. $\hat{C}$ represents the Intersection over Union (IoU) between the predicted bounding box and the ground truth bounding box. It returns a 1, if an object is present in the grid cell, else it returns a 0. Noobj returns the opposite so it returns a 0, if an object is present in the grid cell, else it returns a 1 [11].

$$\sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Fig. 6. Loss Function Four [11]

The final loss function used is essentially a sum-squared error function commonly used in classification with the addition of the conditional 1-obj value discussed earlier [11].

2) How does YOLOv5 work?

Our implementation uses an advanced version of the YOLO model known as the YOLOv5 model. It essentially operates

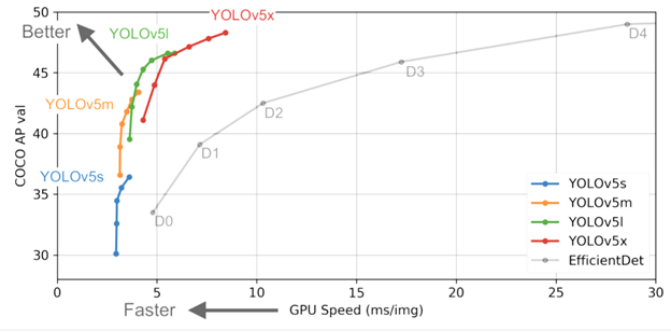

Fig. 7. YOLOv5 Version Performance [14]

like the previous versions of YOLO with the addition of a more complex architecture called EfficientDet. This allows YOLOv5 to achieve a higher output accuracy with better generalization towards a wider range of object categories [15].

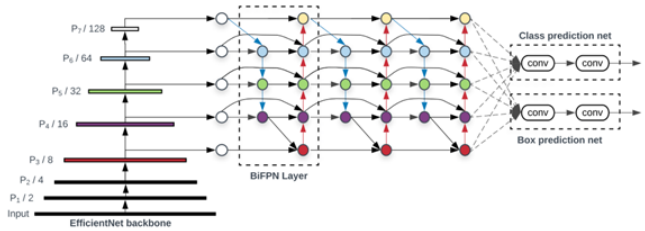

Fig. 8. YOLOv5 Network Structure [15]

In addition to the Box prediction network EfficentDet uses 2 additional layers: the EfficentNet backbone and a BiFPN Layer. The backbone is comprised of ImageNet-pretrained EfficientNets. The BiFPN also known as a Weighted Bi-directional Feature Pyramid Network, is a type of feature pyramid network that allows easy and fast multi-scale feature fusion. The BiFPN layer serves as the feature network taking level 3-7 features P3, P4, P5, P6, P7 from the backbone network and repeatedly applying top-down and bottom-up bidirectional feature fusion. The fused features are then fed to the bounded box network to produce the bounded object predictions [15].

3) How does YOLOv8 work?

The YOLOv8 object identification model, which Ultralytics launched in January 2023, is a notable advancement in the YOLO series. Building upon the fundamental ideas set forth by its forebears, YOLOv8 offers a number of architectural improvements targeted at raising accuracy and efficiency [10],[16]. To accommodate different computing resource requirements, the model is offered in five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra big) [10],[16]. The YOLOv8n version, which provided a notable performance gain of more than 30% over YOLOv5 on comparable devices, was used in our study. Although YOLOv8 is newer and more rapid, its adoption is first hindered by YOLOv5's greater community support and resources. However, it is a promising option for applications in traffic analysis and automated driving systems because of its improved real-time detecting capabilities. The model has the potential to improve the efficiency and safety of urban transportation, as demonstrated by its capacity to reliably identify and track cars in real-time.

YOLOv8 differs significantly from YOLOv5 in that it is an anchor-free model [10],[17]. This indicates that it detects objects without the usage of pre-defined anchor boxes. Rather than relying on box predictions, it makes the bounding box predictions directly, which expedites the Non-maximum Suppression (NMS) process. Especially for smaller models, this method can result in faster inference times, which makes YOLOv8 more appropriate for real-time applications where latency is critical.

Moreover, mosaic augmentation—a data augmentation method that integrates several images into a single training sample—is incorporated by YOLOv8 during training. As a result, the model has access to a wider range of features for learning. To avoid any negative impact on the model's performance, YOLOv8 deactivates this augmentation in the final 10 training epochs.

YOLOv8 introduces modifications to its design in contrast to earlier iterations. For example, the primary building block has changed, and the initial convolution in the stem is now a 3×3 rather than a 6×6 [16]. Furthermore, YOLOv8 employs a new module, C2f, in place of C3, which modifies the concatenation and utilization of the outputs from the bottleneck layers. The enhanced effectiveness and performance of the model are a result of these changes.

The annotation format used by YOLOv8 is still YOLOv5 PyTorch TXT, which is a modified Darknet annotation format [18]. Ultralytics recommends Roboflow as an annotation and export solution for YOLOv8 projects and has worked with it for labeling. Through this partnership, consumers will always have access to effective tools for getting ready their data for YOLOv8.

4) How does the U-Net architecture work?

The U-Net architecture is designed for image segmentation tasks, where the goal is to predict a segmentation mask for each pixel in an input image. It's particularly effective in tasks where high-resolution segmentation with precise localization is required, such as medical image segmentation. It has 2 main layers: encoding, and decoding [19].

When encoding, the input image is passed through a series of convolutional layers with pooling operations to gradually reduce spatial dimensions while increasing the number of channels [19]. This part of the network captures features at different levels of abstraction.

The decoder part of the network consists of a series of upsampling operations followed by convolutional layers. This part of the network gradually recovers spatial information lost during the encoding process. Skip connections are employed between corresponding encoder and decoder layers. These skip connections concatenate feature maps from the encoder with

feature maps at the same spatial resolution in the decoder. This helps the network preserve fine-grained details and spatial information, improving segmentation accuracy [19]. The decoder gradually increases spatial dimensions while reducing the number of channels, eventually producing a segmentation mask with the same dimensions as the input image. The final layer of the network typically consists of a convolutional layer with a softmax or sigmoid activation function to produce the final segmentation mask [19]. For each pixel in the input image, the network predicts the probability of belonging to each class (in the case of multi-class segmentation) or the probability of being the object of interest (in the case of binary segmentation).

U-Net is typically trained using a loss function suited for segmentation tasks, such as cross-entropy loss or dice loss [19]. The model is trained end-to-end using labeled images and optimization techniques like stochastic gradient descent (SGD) or Adam. During training, the model learns to minimize the discrepancy between the predicted segmentation mask and the ground truth mask, adjusting its parameters to improve segmentation accuracy [19].

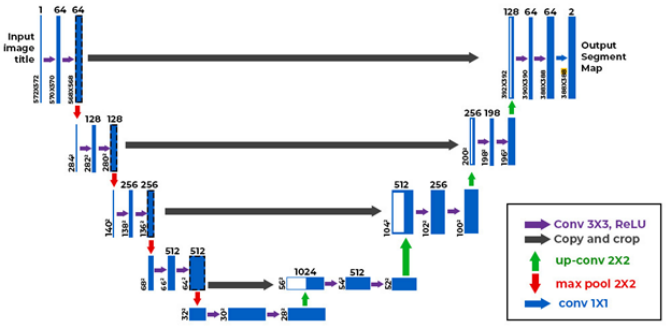

Fig. 9. U-Net Architecture [20]

5) How does ResNet-34 work?

ResNet-34 is a convolutional neural network architecture that belongs to the family of Residual Networks (ResNets). ResNets addressed the problem of vanishing gradients in very deep neural networks by introducing skip connections, or shortcuts, that bypass one or more layers [21].

ResNet-34 consists of basic building blocks called residual blocks. Each residual block typically contains several convolutional layers with batch normalization and ReLU activation functions. The core idea behind ResNet is the skip connection [21]. Instead of directly connecting the output of one layer to the next layer, ResNet introduces a shortcut connection that skips one or more layers. This shortcut allows the gradient to flow more directly through the network during backpropagation, which helps alleviate the vanishing gradient problem. This skip connection adds the input of a layer to its output. Mathematically, if x represents the input to a residual block, and $F(x)$ represents the output of the block (after passing through the convolutional layers), the output of the residual block is $F(x) + x$ [21]. After passing through the convolutional layers and residual blocks, the feature maps are typically processed by global average pooling to reduce the spatial dimensions to a vector. Finally, a fully connected layer followed by a softmax activation function is used for classification.

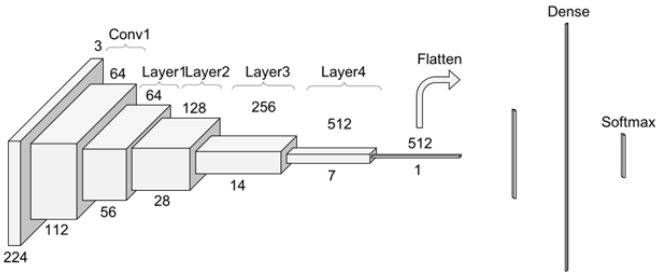

Fig. 10. ResNet Architecture [21]

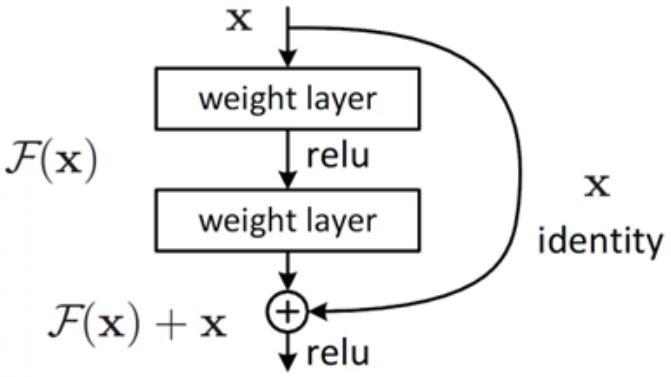

Fig. 11. Residual Block [21]

IV. Experimental Results

A. U-Net Model with a ResNet-34 Backbone Encoder

The CityScapes dataset was used to train and test the model due to its substantial amount of data and is a widely used benchmark dataset for semantic understanding of urban street scenes [22]. This allowed the model to accurately predict segmentation masks as shown in figure 12.

From the semantic segmentation masks generated by the trained model, a measure of road busyness is derived by summing relevant pixels associated with classes such as 'person', 'rider', 'car', 'truck', 'bus', and 'bicycle'. This pixel count is then normalized by dividing it by the total number of pixels in the image, ensuring the measure is independent of image size. Subsequently, the normalized ratio is scaled by a factor of 10 to express road busyness on a comprehensible scale ranging from 1 to 10, where higher values signify increased road activity or congestion. This process enables the objective quantification and comparison of traffic intensity in urban street scenes, offering insights valuable for traffic management, urban planning, and intelligent transportation systems.

B. YOLOv5 Object Detection

The pretrained YOLOv5 model we used in our study is found on GitHub [23]. This model is trained on the COCO

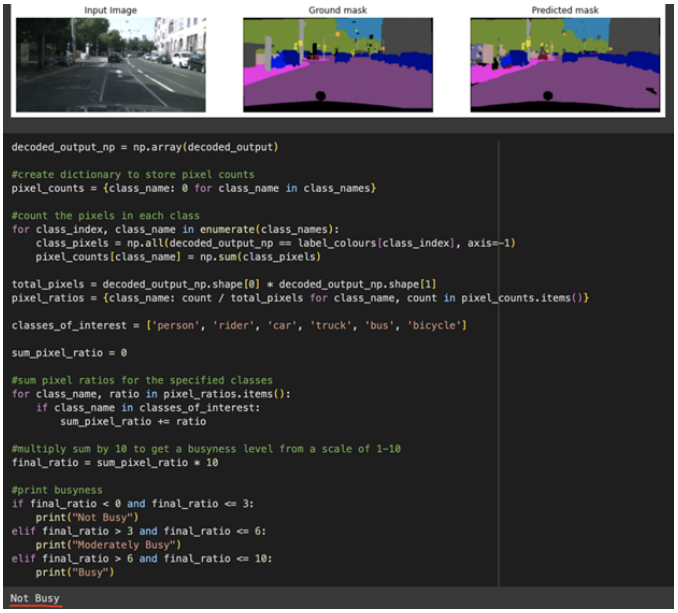

Fig. 12. Predicted Segmentation Mask vs Actual Mask

2017 dataset [24]. To calculate the busyness metric using YOLOv5 we decided to first classify and count the number of vehicles. This would indicate how much traffic is observed in the image, the results are shown in figure 13.

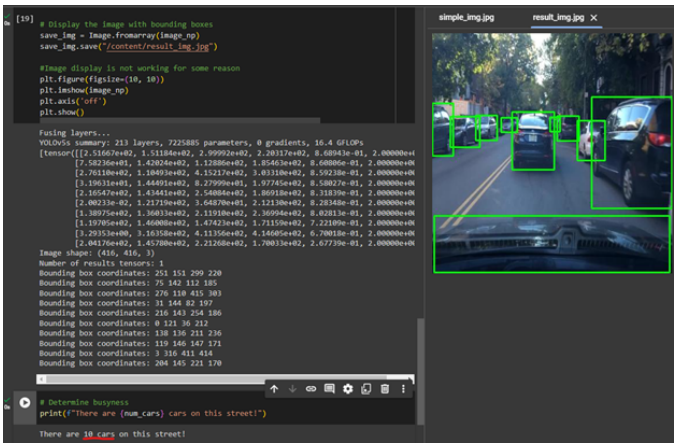

Fig. 13. YOLOv5 Car Identification Test One

Initial results look promising however YOLOv5 has limitations and our metric isn't optimal. The metric would be biased towards smaller streets with highways, usually experiencing high traffic volumes, inaccurately represented in comparison. After testing the algorithm on an observed image of a highway we noticed that not only was the metric inefficient but the YOLOv5 model itself was inadequate for the scale on which we were trying to deploy the metric as seen in figure 14. Only 17 cars are detected with smaller scale and obscure vehicles going unclassified.

We decided to follow through with the following two design changes. Firstly, we needed to make the metric more versatile,

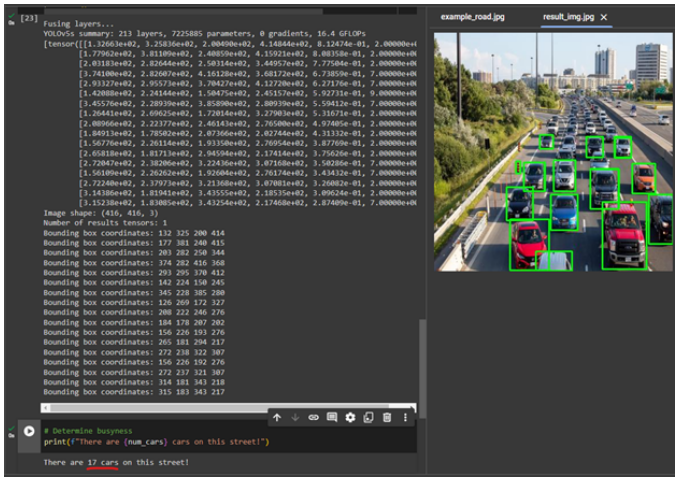

Fig. 14. YOLOv5 Car Identification Test Two

to do this we decided to also classify and count the number of lanes in the observed image as well. The metric would be calculated in the following manner busyness $B_s = (V/L)$ where V is the number of classified vehicles and L is the number of classified lanes in the . Secondly, we decided to migrate to a pre-trained YOLOv8 model to improve the accuracy of classification to resolve the issues in figure 14.

C. YOLOv8 Object and Lane Detection

In our study, we improved the accuracy of object and lane detection in urban traffic scenarios by switching from the YOLOv5 model to the YOLOv8 model. With its enhanced design and performance, the YOLOv8 model has shown to be more successful in lane configuration estimation and vehicle identification.

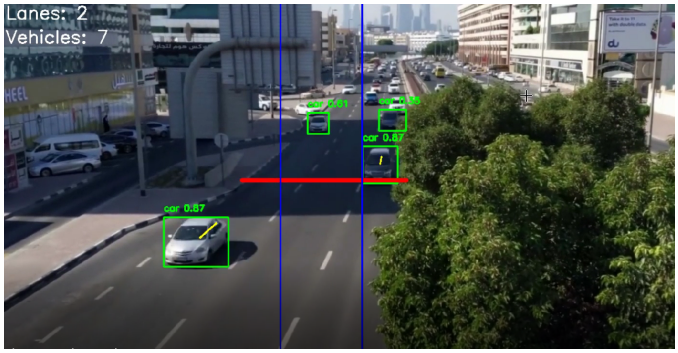

Fig. 15. StreetSight Start of Demonstration

Our Python software, which is accessible on GitHub [25], tracks objects using the SORT method and detects vehicles using the Ultralytics YOLOv8 model. The script's features include processing video streams, precise car detection in every frame, and the ability to estimate the number of lanes on the road by using KMeans clustering on vehicle positions. The script also offers visualization functions, including class labels with confidence scores, vehicle trajectories and estimated lanes, and bounding boxes around observed cars.

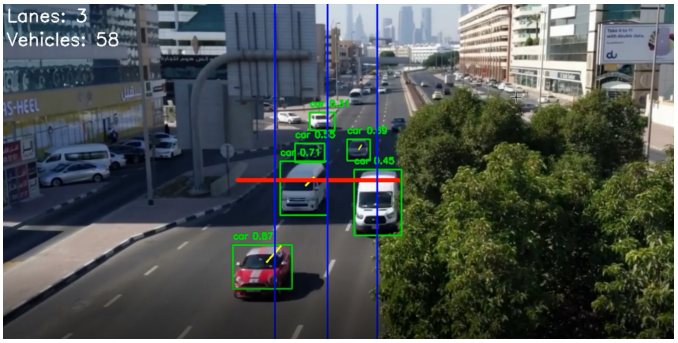

Fig. 16. StreetSight End of Demonstation

Important components of our execution consist of:

- Vehicle Detection: Using the YOLOv8 model to accurately identify cars.
- Object tracking involves following each identified vehicle's path across frames using the SORT method.
- Lane estimation uses the clustering of vehicle positions to estimate the number of lanes on the road.
- Visualization: By highlighting identified vehicles, displaying their confidence and classification levels, and denoting predicted lanes, this technique provides real-time visual feedback.

The video demonstration in the github README is only using the nano version of YOLOv8. Much better performance can be expected when using the top level models of YOLOV8. Python 3, OpenCV, Ultralytics YOLO, the SORT algorithm, and the Scikit-learn libraries are needed for the script. The traffic video, the pre-trained YOLOv8 weights from Ultralytics, and an optional mask picture to target detection at a particular area inside the video frames can all be downloaded and added to the project directory.

The business metric we chose for the YOLOv8 model is very similar to the YOLOv5 model with minor adjustments to match the change from photo to video content. In practice the aforementioned formula can be reapplied with minor adjustments.

By putting this YOLOv8-based strategy into practice, we hope to progress autonomous driving and intelligent transportation systems by offering a more reliable and precise solution for vehicle identification and lane estimates in urban traffic conditions.

V. Future Improvements

For future improvements and iterations on our analysis, we recommend focusing on several key areas to enhance the capabilities and applicability of computer vision systems for urban traffic management. Firstly, refining our machine learning algorithms to improve the accuracy and efficiency of traffic is a straightforward enhancement. Additionally, exploring the use of alternative datasets beyond Cityscapes to enhance the robustness and generalizability of our system to span many cities and potentially countries. Another enhancement would be training the data on real world data by setting up an experiment at several busy intersections. The practical integration of our models into existing traffic management systems and urban planning processes will be important to reveal the full feasibility of our project across cities. Another enhancement would be to expand the scope of our project to include other aspects of mobility, such as walkable cities and public transport optimization. Collaborating with urban planners and traffic management authorities would give us input from real world actors to tailor our solutions to meet specific applicable needs. Moreover, we will ensure to develop privacy-preserving techniques to prioritize the ethical use of visual data in traffic analysis. Advancements in semantic segmentation techniques for autonomous vehicles will also be an area to expand into, focusing on enhancing safety and efficiency in urban settings. For the YOLO models, moving to Google Colab, using the virtual GPUs and selecting the most powerful versions of the v8 and v5 model will be a good way to increase performance. Finally, cross-disciplinary research with experts in transportation engineering, urban planning, and environmental science to create a complete approach to tackle the problem that is traffic congestion management. Overall, this study and "Street Sight" are just a starting point. There are endless enhancements and areas to improve in, and anticipate many improvements that we have not yet considered.

VI. Conclusion

In conclusion, this study successfully demonstrates the potential application of computer vision techniques in analyzing traffic congestion in urban environments. By investigating two research methodologies: leveraging advanced machine learning algorithms and pre-trained models, we were able to assess and evaluate the strengths and weaknesses of both technologies in a traffic setting. Our analysis demonstrates computer vision can output accurate and real-time insights into traffic density and flow. These two metrics are useful for effective traffic management and autonomous transportation software. The expenses and logistical difficulties attached to conventional traffic monitoring techniques, such as tedious surveys or the costly installation of many physical sensors, can be lowered by computer vision. We hope that our study's results will provide urban planners and traffic management authorities useful insights and better plans to improve traffic flow and monitoring. Furthermore, through ameliorating the research methodologies discussed in this study, it is possible to analyze travel patterns within cities and improve their traversal efficiency as well as improve future cities. As urbanization continues and cities become dense concrete jungles, embracing computer vision will be an effective method to address the complex challenges of traffic in urban settings.

Acknowledgment

The professors and teaching assistants of the CS4442 Artificial Intelligence II course have our deepest gratitude and appreciation on behalf of the authors. We would like to express our gratitude to the Western University Faculty of Engineering as well as the broader Western University community for

creating an atmosphere that encourages study and development and helps us become successful students. Your dedication and commitment have contributed heavily to our academic journey, for which we are truly grateful.

REFERENCES

[1] Z. Bao, Y. Ou, S. Chen, and T. Wang, "Land Use Impacts on Traffic Congestion Patterns: A Tale of a Northwestern Chinese City," *Land*, vol. 11, no. 12, p. 2295, Dec. 2022. [Online]. Available: https://www.mdpi.com/2073-445X/11/12/2295.

[2] S. Koutra and C. S. Ioakimidis, "Unveiling the Potential of Machine Learning Applications in Urban Planning Challenges," *Land*, vol. 12, no. 1, p. 83, Dec. 2022. [Online]. Available: https://www.mdpi.com/2073-445X/12/1/83.

[3] V. Chaturvedi and W. T. de Vries, "Machine Learning Algorithms for Urban Land Use Planning: A Review," *Urban Science*, vol. 5, no. 3, p. 68, Sep. 2021. [Online]. Available: https://www.mdpi.com/2413-8851/5/3/68.

[4] IBM, "What is Computer Vision?" [Online]. Available: https://www.ibm.com/topics/computer-vision.

[5] Cityscapes Dataset, "Dataset Overview." [Online]. Available: https://www.cityscapes-dataset.com/dataset-overview/.

[6] A. Mishra, "Decoding the technology behind Tesla Autopilot: How it works," *Medium*. [Online]. Available: https://ai.plainenglish.io/decoding-the-technology-behind-tesla-autopilot-how-it-works-af92cdd5605f.

[7] J. Cohen, "Tesla's HydraNet - how Tesla's Autopilot Works," *Welcome to The Library!*. [Online]. Available: https://www.thinkautonomous.ai/blog/how-tesla-autopilot-works.

[8] Kili, "Yolo Algorithm: Real-time object detection from A to Z." [Online]. Available: https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z.

[9] T. Eduonix, "Real-world implementations of Yolo algorithm," *Eduonix Blog*. [Online]. Available: https://blog.eduonix.com/2022/01/real-world-implementations-of-yolo-algorithm/.

[10] J. R. Terven and D. M. Cordova-Esparaza, "A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond," Under review in *ACM Computing Surveys*. [Online]. Available: https://arxiv.org/pdf/2304.00501v1.pdf.

[11] HackerNoon, "Understanding yolo." [Online]. Available: https://hackernoon.com/understanding-yolo-f5a74bbc7967.

[12] Z. Keita, "Yolo Object Detection explained: A beginner's guide," *DataCamp*. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained.

[13] S. K. Sambasivarao, "Non-maximum suppression (NMS)," *Medium*. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[14] R. Coetzee, "YOLOv8 vs YOLOv5: Choosing the Best Object Detection Model," *Augmented Startups*. [Online]. Available: https://www.augmentedstartups.com/blog/yolov8-vs-yolov5-choosing-the-best-object-detection-model.

[15] Arxiv, [Online]. Available: https://arxiv.org/pdf/1911.09070.pdf.

[16] J. Solawetz, "What is Yolov8? the ultimate guide. [2024]," *Roboflow Blog*. [Online]. Available: https://blog.roboflow.com/whats-new-in-yolov8/.

[17] Roboflow, "Yolov8 vs. Yolov5: Compared and contrasted," [Online]. Available: https://roboflow.com/compare/yolov8-vs-yolov5.

[18] Ultralytics, "YOLOv8," *GitHub*. [Online]. Available: https://github.com/ultralytics/yolov8.

[19] P. S, "A comprehensive guide to UNET architecture: Mastering image segmentation," *Analytics Vidhya*. [Online]. Available: https://www.analyticsvidhya.com/blog/2023/08/unet-architecture-mastering-image-segmentation/.

[20] GeeksforGeeks, "U-Net Architecture explained," [Online]. Available: https://www.geeksforgeeks.org/u-net-architecture-explained/.

[21] P. Ruiz, "Understanding and visualizing ResNets," *Medium*. [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8.

[22] Cityscapes Dataset, "The cityscapes dataset," [Online]. Available: https://www.cityscapes-dataset.com/.

[23] Ultralytics, "Ultralytics/yolov5: Yolov5 in PyTorch ¿ ONNX ¿ CoreML ¿ TFLite," *GitHub*. [Online]. Available: https://github.com/ultralytics/yolov5.

[24] COCO, "Common objects in context," [Online]. Available: http://cocodataset.org/.

[25] Aaron-Moses, "Aaron-Moses/streetsight-computervision-vehicles," *GitHub*. [Online]. Available: https://github.com/Aaron-Moses/StreetSight-ComputerVision-Vehicles.