Industry Project - Road Safety Monitoring System

Shayel Penuela-Celis, Lauren Corbeil, Skyler Serwa, and Holly Young
Department of Computer Science
Brock University
St. Catharines, Canada
sp21ge@brocku.ca, wf22dy@brocku.ca, ss22tp@brocku.ca, hy21zt@brocku.ca

***Abstract*—When drivers are presented with a hazard on the road, their response affects everyone around them. Without a given warning, rash decisions can occur and lead to dangerous accidents, or roads can be neglected and develop long-term problems. Consequently, contemporary infrastructure has brought with it the need for real-time, reliable, mass information updates. Providing correct and immediate information to drivers is crucial for improving the flow of traffic and minimizing risks on the road. Fewer accidents also allow everyone to get to where they need to faster, including emergency vehicles. Luckily, the majority of today's population is equipped with a phone that can receive instant status updates. All that is required is an application that can quickly and accurately report on local or trip-related concerns to warn users. To learn how to curate an application that achieves this, we built one using different path finding algorithms applied to real road data taken from Calaguiro Estates, Niagara Falls. Our experiments helped show the importance of quantity and quality of data in machine learning.**

I. Introduction

In 2023, road-related incidents were the leading cause of death for young people, accounting for 1.19 million deaths globally [2]. Recent advancements in artificial intelligence (AI) must be leveraged to combat this daunting statistic. Modern computer vision systems have opened the doors to a diverse set of solutions for many common tasks, given their ability to analyze large datasets and automate repetitive work. Road monitoring systems are one of many tools that have been enhanced with AI over the past few years.

While AI has been shown to greatly improve data analysis, there are still many factors that should be considered when building an intelligent system. For example, there are a diverse set of methods that can be used to calculate routes, each with their own unique outcomes. Tailoring a route-calculating program to always find the best option even with applied restraints is a process that can be approached from all angles. From data acquisition to search algorithms, each decision should be carefully considered when designing a system.

To test the efficiency of different road-monitoring systems, we built a route-calculating web application based on the Calaguiro Estates neighborhood in Niagara Falls, Ontario. We applied two different search algorithms to a dataset of 521 images that were fed through a computer vision model for analysis. Our goal was to test the quality of our computer vision model, and compare the functionality of both of the search algorithms.

II. Background

To separate unsafe roads from safe roads, we trained **YOLOv8** (You Only Look Once, version 8) on the image data of Calaguiro Estates that we collected. YOLOv8 is a computer vision model that uses convolutional neural networks to detect and localize objects in images [1]. Once the computer vision model was properly trained, we were able to apply search algorithms to a graph representation of the images at their real life locations. The search algorithms that we used to find the best routes were Dijkstra's and A*.

Dijkstra's algorithm is a greedy path finding algorithm used to determine the shortest path between nodes in a weighted graph. The graph is composed of nodes, edges, and non-negative weights, which represent the cost of traveling between connected nodes. By utilizing a priority queue, the algorithm systematically explores nodes in order of increasing path cost, ensuring that the shortest known distance to each node is processed first.

In the context of our GPS application, nodes represent specific geographic coordinates within the selected region, while edge weights correspond to the distances between these coordinates. The algorithm begins by accepting user input that specifies both the starting location and the desired destination, allowing the system to identify the appropriate start and end nodes within the graph.

Dijkstra's algorithm has a time complexity of O(ElogV) when implemented using a priority queue, which makes it suitable for large-scale maps and dense road networks. The performance of Dijkstra's algorithm was comparable to that of the A* search algorithm in our case, due to the relatively small search area. However, it is expected that A* would demonstrate superior performance when applied to larger datasets, such as city-wide or regional street networks, due to its use of heuristic guidance.

Algorithm 1 Simple Dijkstra Implementation

```
while Destination is not visited do
    Move to the location with the shortest overall path
    Mark the new location as visited
    for all unvisited neighbors do
        Update shortest path
    end for
end while
```

A* is similar to Dijkstra's, with the exception of using a **heuristic** to determine which node to traverse to rather than just picking the shortest overall path. A heuristic uses information on its environment to determine if some routes are simply not worth taking, despite having the shortest distance. This helps increase program efficiency and decrease memory usage.

Algorithm 2 Simple A* Implementation

```
while Destination is not visited do
    Move to the location with the shortest path + best heuristic evaluation
    Mark the new location as visited
    for all unvisited neighbors do
        Update the best path using the shortest value + best heuristic evaluation
    end for
end while
```

III. Application Setup

The first step to creating the web application was selecting a location to perform the route-calculations on. We chose Calaguiro Estates due to its proximity and our familiarity with the area. Once we measured a 1.36 km^2 area, we selected 92 points to use as nodes for the graph-representation. Each node was photographed six times total, with three images from each direction and two images from the three most recent available years. Every image was then annotated with the year it was taken and the road's condition.

The nodes within the code stored the directions and distances to their adjacent nodes, as well as their images and image metadata. The computer vision model was trained on the dataset to determine safe roads, while the search algorithms found the best paths to traverse based on the distances and road safety.

IV. Results

The core of the computer vision system is a fine-tuned **Convolutional Neural Network** (CNN) based on the **YOLOv8** architecture. This model was selected for its balance of high accuracy and computational efficiency, making it suitable for deployment in resource-constrained smart city environments.

The model was trained with transfer learning, which leverages pre-trained weights to accelerate convergence. The final model weights are saved in a .pt format, which contains the optimized parameters for distinguishing between "Safe" and "Hazardous" road conditions. To find the file, you need to navigate through the following directories within the project:

```
\road_safety_project\hazard_classifier2\
weights\best.pt
```

The development pipeline consists of three primary Python scripts designed to ensure reproducible results. The first is **prepare-data**. This script handles raw data collection and annotation requirements. It parses the project's metadata, applies binary classification logic (Severity = 0 vs. Severity > 0), and creates a stratified split of the dataset. The split consists of a 70:15:15 ratio of images, with 70% as a training set, 15% as a validation set, and 15% as a testing set.

The next script is **train-model**. This script initializes the YOLOv8 engine. It configures the training parameters, including the number of epochs and image sizing, and manages the transfer learning process to prevent overfitting. We initialized our parameters as:

```
data = 'dataset_cls',
epochs = 20,
imgsz = 224,
project = 'road_safety_project',
name = 'hazard_classifier'
```

Where: **data** points to the folder created by prepare_data.py, **epoch** refers to the number of iterations through the entire dataset, **imgsz** resized all images to 224x224 pixels, **project** names the output folder, and **name** names the specific run of the program.

The final script is **cv-inference**. This script loads the trained *best.pt* weights and runs inference on the unseen data. It then converts the model's raw probability outputs into a standardized dictionary format, used by the graph generation component.

A. Performance Metrics and Analysis

The model's performance was evaluated using the held-out validation set. The metrics below demonstrate the model's ability to correctly identify hazardous road features (cracks, potholes, etc.) versus safe roads.

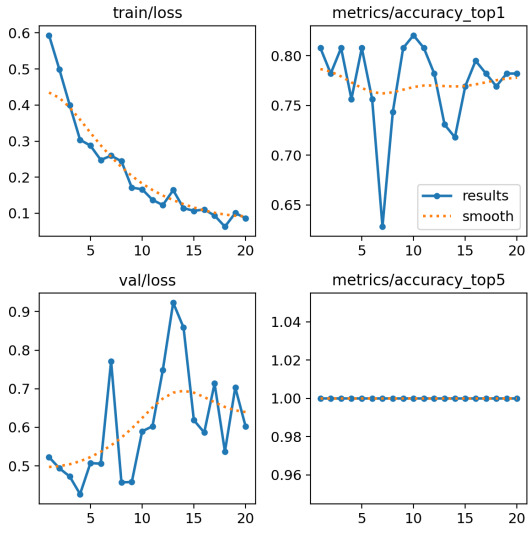

Fig. 1. Training Loss and Validation Accuracy over 20 Epochs. The graph demonstrates model convergence with significant overfitting.

Our training shows the model clearly improving each epoch in regards to overall accuracy, and yet, due to our small sample size, it's tendency to overfit leads to some rough patches for our top 1 accuracy and loss. We cover some reasons for this further below, but in general our model is struggling with our low sample of hazardous images to correctly guess them right compared to our high sample of safe images. That is why our training is still overall good while our accuracy lags behind.

B. Quantitative Metrics

Metric	Value
Top-1 Accuracy	0.8205
Macro Average	0.70
Weighted Average	0.82
Precision (Hazardous)	0.54
Recall (Hazardous)	0.47
F1-Score (Hazardous)	0.50
Precision (Safe)	0.88
Recall (Safe)	0.90
F1-Score (Safe)	0.89

TABLE I
FINAL MODEL PERFORMANCE METRICS

This data was collected from the sample run of our GitHub files, located at 'dataset-cls/val'. Our precision in identifying hazardous images was fairly poor. We estimate that this is caused by hazardous images only making up 15 images from the validation set. In contrast, the 64 safe images significantly increased the precision in identifying safe roads. Weighting our average accuracy in favour of safe images brings our total back to our standard accuracy average of 0.82. For our next steps, we would fix the 0.47 recall percentage on hazardous images through rigorous training with many more example photos.

C. Confusion Matrix

The confusion matrix below illustrates the model's classification performance, highlighting false positives and false negatives.

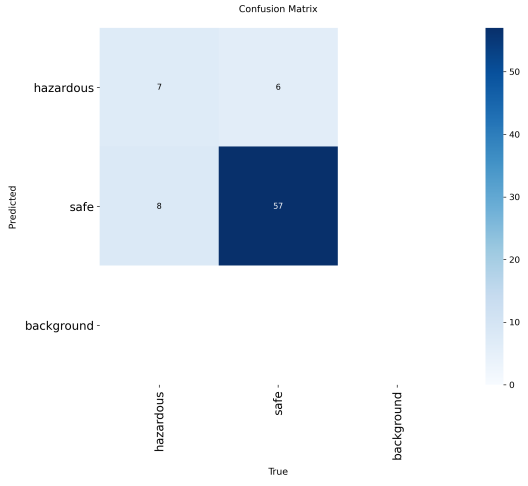

Fig. 2. Confusion Matrix for Safe vs. Hazardous Classification.

The model correctly identified 57 roads and 7 hazardous ones. It only missed 14 roads in total. Looking at the actual set of all our images, it becomes easier to understand why. Each of the four team members had slightly different personal computers of which to capture these photos, and were unclear on how obvious a hazard might be for the system to identify. With only 500 images confined to a relatively safe neighborhood, we also did not possess the ability to train for specific hazard types. In repeating our steps, with more time and funding, we could accurately create a much larger matrix for our model to not only guess if an image contains a hazard, but also identify which kind of hazard it contains.

D. Model Inference Pipeline

The inference pipeline is encapsulated in the `RoadScanner` class. This component integrates the computer vision model directly into the geospatial graph construction. It accepts raw image data, preprocesses it to meet the model's input requirements, and returns a **Hazard Score** from 0.0 to 1.0.

This score is used to dynamically weigh the edges of the navigation graph, effectively discouraging hazardous routes so that the path finding algorithms (Dijkstra and A*) favour safer alternatives.

1. Example Prediction Output for a Hazardous Image:

```
[caption=Inference Pipeline Output Log]
Test Result for images/h_pics/h2_1.png:
{
'is_hazardous': True,
'hazard_score': 0.69,
'prediction': 'hazardous'
}
```

V. CONCLUSION

Our final web application was able to accurately find a safe route between two points within our chosen location. A hurdle we faced in the initial stages of data collection was realizing that we had chosen an upper-class neighborhood with minimal road issues. Looking back, it would have been ideal to select a neighborhood that was in greater need of road repair, so that the machine learning model could experience a more diverse data set. Despite this, our app was still able to function with the provided data and correctly avoid unsafe routes. The Dijkstra and A* search algorithms also both performed well in traversing the location, although A* is presumed to allow more scalability for larger locations with it's ability to eliminate suboptimal routes.

REFERENCES

[1] Kundu, R. (2023, January 17). YOLO: Algorithm for Object Detection Explained [+Examples]. YOLO Algorithm for Object Detection Explained [+Examples]. https://www.v7labs.com/blog/yolo-object-detection

[2] World Health Organization. (2023). Global status report on road safety 2023. Geneva: World Health Organization; 2023. Licence: CC BY-NC-SA 3.0 IGO. https://iris.who.int/server/api/core/bitstreams/46275f9f-ef66-4892-8ddd-a496ef8c1b74/content.