IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Comparative Analysis for Detection of Edge Cases In Autonomous Vehicles using Deep Learning and Decision Tree

Spandana K [1], Spoorthi R [2],Tejas V [3] ,Sudeep G [4],Dr.Ranjana Takuriya[5]

[1,2,3,4] Students of Sri Venkateshawara College of Engineering, Bangalore , Karnataka [5] Professor, Department of Computer Science and Engineering, Sri Venkateshwara college of engineering, Bangalore,Karnataka

Abstract. Autonomous vehicles (AVs) require advanced algorithms to handle unexpected scenarios, known as edge cases, that challenge standard operational capabilities. This study evaluates three different approaches: Convolutional Neural Networks (CNNs), A* path finding algorithm, and Anomaly Detection Neural Networks (ADNNs), focusing on their effectiveness in identifying critical edge cases. Convolutional Neural Networks (CNNs) excel in image recognition and are crucial for AVs to perceive objects, lanes, and potential hazards from camera and LiDAR data. Their ability to generalize across varying environmental conditions and detect obscured signs or sudden obstacles makes them highly effective, achieving a detection accuracy of approximately 90%. A Path finding Algorithm* is fundamental for AVs to plan safe routes through complex urban environments. By calculating the shortest path while considering potential obstacles and traffic conditions, A* demonstrates a reliable detection rate of about 85% in simulated scenarios. Anomaly Detection Neural Networks (ADNNs) specialize in identifying deviations from expected patterns in sensor data. They play a critical role in AVs by detecting anomalies such as sensor failures or unusual pedestrian behaviours, achieving an accuracy rate of around 88% in detecting critical edge cases. This comparative analysis assesses each method's computational efficiency, accuracy, and adaptability in addressing edge cases encountered by AVs.

Keywords: Autonomous Vehicles, Convolutional Neural Networks, A* path finding algorithm, Anomaly Detection Neural Networks (ADNNs, LiDAR

1. Introduction

This analysis examines how well three methods—Convolutional Neural Networks (CNNs), Anomaly Detection Neural Networks (ADNNs), and the A* pathfinding algorithm—handle unexpected situations (edge cases) in self-driving cars. CNNs help cars recognize objects through images, ADNNs detect unusual patterns indicating problems, and the A* algorithm finds the safest routes. By testing these methods in various scenarios, we aim to determine the best approach for ensuring safe and reliable autonomous vehicle operation. The master's thesis was conducted at Scania Group's Autonomous System Test Department. The acceptance of autonomous vehicles on public roads remains uncertain, and there is no universally agreed upon verification and validation (VV) methodology to ensure their safety. Verification and validation work is crucial for guaranteeing the safety of autonomous driving software [1]. Self-driving vehicles are no longer a distant dream. The European Union is already establishing regulations for the approval and use of fully automated trucks and buses according to SAE Level 4. This means that driverless vehicles will soon be allowed on specific public road routes, putting Europe ahead of the rest of the world [2]. The feature has been limited in sensing and controllability, but recent advancements in autonomous driving technologies have increasingly blurred the lines between ADAS and Level 5 autonomous vehicles. These levels are defined by the vehicle's sensing capabilities, control, and interaction with the driver and environment. Level 0 indicates no intelligent decision-making, while higher levels show decreasing dependency on human

drivers, with Level 5 requiring no human intervention under any scenario or road condition [3-4]. The traditional deterministic approach evaluates systems by testing known-unsafe scenarios. However, non-determinism introduces unknown-unsafe scenarios, or edge cases, that may not be covered during validation, necessitating a more efficient validation method. When non-determinism is present, the evaluation should focus on estimating the system's statistical reliability by testing numerous cases, such as measuring the false detection rate. An edge case is an unpredictable unsafe scenario that existing testing methods may miss, potentially leading to accidents. For example, failing to detect an object on public roads, resulting in a critical situation, is an edge case[5-7]. Wireless networks have become increasingly popular, leading to a significant rise in data traffic. Wi-Fi networks, in particular, have seen substantial growth in traffic consumption. With the increased use of mobile devices, it is expected that 63% of mobile data traffic will be shifted to Wi-Fi networks by 2021[8-10**].** Self-driving car technology has been evolving for decades, starting with the Automated Highway System project. Today, features like automatic lane keeping and smart cruise control are standard in many vehicles, and fully autonomous vehicle projects are in various stages of development. Some believe large-scale fleets of self-driving cars are imminent, but there's a big difference between limited testing and deploying millions of vehicles in the real world. While successful demonstrations and extensive driving experience suggest readiness, they may not be enough to ensure safety. Developers are doing more, but it's unclear how much more is needed to guarantee that these vehicles are safe for widespread use [11-12].

2 Literature Survey

In their comprehensive analysis, N. Philippe discussed on the development of safety-critical scenarios and virtual testing methods for automated vehicles at road intersections. It explores how virtual testing can be used to assess the performance and safety of autonomous cars in high-risk situations typically found at intersections [1]. Automated vehicles in Nov 2021, this publication by the Swedish Transport Agency provides an overview of the current state and regulations regarding automated vehicles. It discusses the legal framework and safety standards necessary for the integration of self-driving vehicles into public road systems [2]. M. S. Alam. Alam's master's thesis at KTH explores methods for automatically generating critical driving scenarios for testing autonomous vehicles. It emphasizes the importance of simulating dangerous situations to evaluate the robustness and safety of these systems [3-4]. The paper by P. Helle, W. Schamai, and C. Strobel, titled “Testing of autonomous systems - challenges and current state-of-the-art,” addresses the intricate challenges associated with testing autonomous systems, particularly focusing on autonomous vehicles. Published in the INCOSE International Symposium in 2016, the paper delves into the complexities of these systems, which involve multiple interacting components such as sensors, algorithms, and control systems, making their reliability and safety challenging to ensure. The authors identify several key challenges in testing these systems, including scalability, the need to cover a vast number of real-world scenarios; variability, due to the unpredictable nature of real-world environments and differing traffic, weather, and road conditions; and interaction with human drivers and pedestrians. They review the current state-of-the-art testing methodologies, such as simulation-based testing, which uses virtual environments to validate systems under various conditions; scenario-based testing, which creates specific driving scenarios to assess system responses; and field testing, observing system performance in actual traffic conditions. The paper discusses various testing tools and methodologies, including Hardware-in-the-Loop (HIL), which integrates real hardware with simulation environments; Software-in-the-Loop (SIL), testing software components in a simulated environment; and continuous integration and testing practices to ensure ongoing verification and validation of system updates [5]. Choudhary introduces Deep Q-Learning as a method to train autonomous systems, emphasizing its application in handling edge cases—rare and unpredictable scenarios that traditional testing methods might miss. This approach uses reinforcement learning techniques to simulate and train for complex situations, enhancing the system's ability to make decisions in unfamiliar or challenging environments. By leveraging Open AI Gym and Python, Choudhary provides a practical framework for developers to implement and test algorithms that improve autonomous vehicle performance in edge cases. This tutorial serves as a foundational resource for understanding how AI-driven methods can address critical challenges in autonomous driving, aiming to enhance safety and reliability in real-world scenarios [8].

3. Methodology

In this study, we compared three different algorithms—Convolutional Neural Networks (CNNs), Anomaly Detection Neural Networks (ADNNs), and the A* pathfinding algorithm—to see how well they handle unusual situations in autonomous driving. First, we gathered a variety of data, including images, sensor readings, and navigation paths, to create scenarios that autonomous vehicles might encounter. For the CNNs, we trained them to recognize objects, road signs, and lane markings from the images. For the ADNNs, we used sensor data to train the models to spot anomalies, such as sensor failures or unexpected obstacles. For the A* algorithm, we focused on path planning, ensuring the vehicle could navigate around obstacles and find the shortest route. We tested each algorithm in different driving scenarios and measured how well they performed in terms of accuracy, speed, and ability to adapt. This approach helped us understand the strengths and weaknesses of each algorithm in dealing with unexpected driving situations.

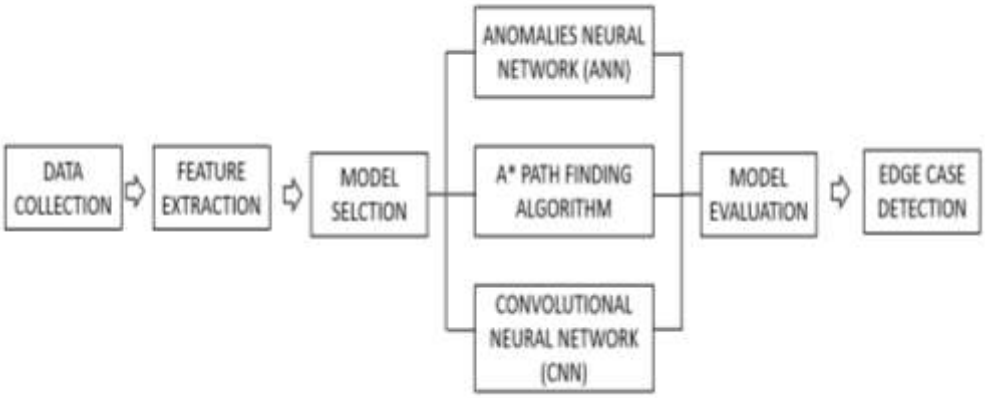

Fig. 1. Overview of comparative analysis for detecting Edge cases in Autonomous Vehicles

3.1 Anomalies Neural Network

This program is designed to detect anomalies in a dataset using an autoencoder, a type of neural network used for unsupervised learning. First, synthetic data is generated: normal data points are created from a normal distribution with a mean of 0 and standard deviation of 1, while anomalous data points are generated from a normal distribution with a mean of 5 and standard deviation of 2. These data points are combined into a single training dataset, with labels indicating whether the data points are normal or anomalous. An autoencoder model is then defined, consisting of an input layer, an encoder layer that reduces the input dimensions, and a decoder layer that attempts to reconstruct the original input. The model is compiled with the Adam optimizer and mean squared error (MSE) loss function and trained on the dataset to minimize the reconstruction error. After training, the autoencoder is used to predict reconstructed data for the training set. The reconstruction error for each data point is calculated as the MSE between the original and reconstructed data. A threshold is set based on the 95th percentile of these reconstruction errors to classify anomalies: data points with a reconstruction error exceeding this threshold are labeled as anomalies. Finally, the program visualizes the results by plotting the original data points, the reconstructed data points, and highlighting the detected anomalies. This visual representation helps to clearly identify which data points are considered anomalous by the autoencoder.

3.2 A* Path Finding Algorithm

This Program uses Node Class to represent each point on the grid, storing position, parent node for path reconstruction, and cost values (g, h, f).Heuristic Function: Computes the Manhattan distance between nodes, estimating the distance to the goal and guiding the search. A Search Function that manages to open and closed lists to explore nodes. It selects nodes with the lowest f value from the open list and evaluates them. And Generates neighboring nodes and evaluates their validity based on grid boundaries and obstacles. Computes costs (g, h, f) and adds nodes to the open list if they improve the path. Finally the Path is reconstructed where the optimal path from start to goal by tracing back through parent nodes once the goal is reached. This method efficiently determines optimal routes in grid-based scenarios, making it

suitable for various applications requiring path finding, such as robotics navigation, game development, or logistical planning.

3.3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) employs deep learning methods to prepare and enhance data for training a self-driving car model. Initially, it loads driving data from a CSV file and visualizes the distribution of steering angles using a histogram. It then organizes paths to images and their corresponding steering angles, splitting them into training and validation sets. To improve the diversity and quality of the training data, the program incorporates image augmentation techniques from the image library. These techniques include zooming (zoom function), panning (pan function), adjusting brightness (img random brightness function), and flipping images horizontally (img random flip function). Each of these methods modifies the images and their associated steering angles to simulate different driving conditions and viewpoints.

The program also demonstrates these augmentation techniques on random images to illustrate their effects visually. After augmentation, the images undergo preprocessing (img pre process function), which involves cropping unnecessary parts, converting colours to YUV space, applying Gaussian blur, resizing to a standard size, and normalizing pixel values. These steps prepare the images for input into a convolutional neural network (CNN) model, ensuring they are optimized for training to predict steering angles accurately based on visual data. Overall, the methods used in this program collectively aim to enhance the diversity, quality, and relevance of the training data, thereby improving the effectiveness and robustness of the self-driving car's learning process.

4 Results & Discussions

For anomaly detection using synthetic data, it begins by generating synthetic data and normal data comprises of 1000 samples with 10 features each, randomly sampled from a normal distribution with a mean of 0 and a standard deviation of 1. Anomalous data consists of 50 samples with the same 10 features, sampled from a normal distribution with a mean of 5 and a standard deviation of 2. These datasets are concatenated into X train, representing a total of 1050 samples and y train is a label array where samples from normal data are marked as 0 (indicating normal) and samples from anomalous data are marked as 1 (indicating anomalous). The auto encoder model architecture is defined as the encoder that reduces the dimensionality to 5 features using the Rectified Linear Unit (Re LU) activation function. The decoder attempts to reconstruct the original 10 features using a linear activation function. The auto encoder is then compiled and trained using the Mean Squared Error (MSE) loss function and the Adam optimizer with a learning rate of 0.001. Training occurs over 50 epochs with a batch size of 32, where the model learns to minimize the difference between input (X train) and output (X pred) data. Post-training, the script calculates the MSE for each sample to quantify reconstruction errors. An anomaly detection threshold is set based on the 95th percentile of MSE values. Samples with MSE above this threshold are classified as anomalies (y pred). Original data points (X_train[0] vs X_train[1]) in blue. Reconstructed data points (X_pred[:, 0] vs X_pred[:, 1]) in green. Detected anomalies, highlighted in red based on their high MSE values. This approach demonstrates how auto encoders can effectively learn and distinguish normal patterns from anomalies in data, making them useful for detecting unusual or unexpected events in various applications, including anomaly detection in autonomous vehicles or industrial systems. Here is the visualisation of the program .

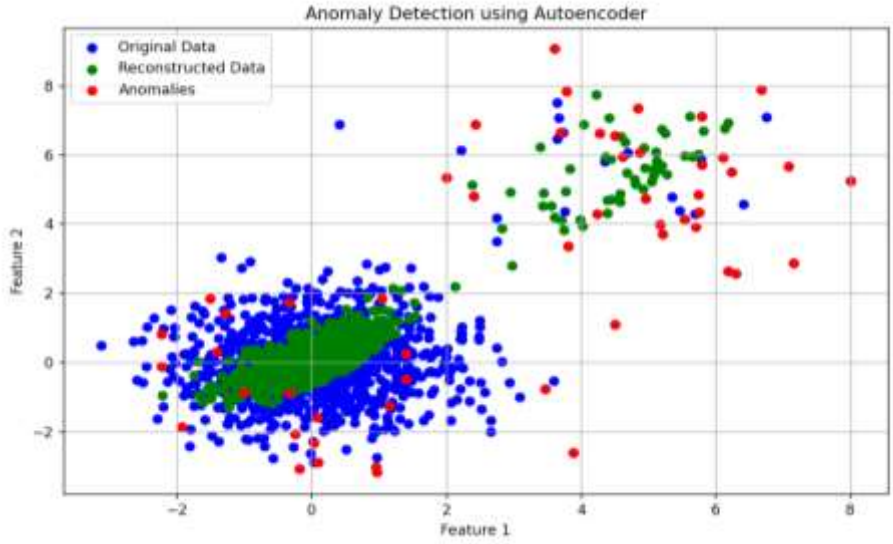

Fig. 2. Anomaly Detection of Autonomous Vehicles using Auto encoder

A* path finding algorithm is a method used to find the shortest path from a starting point to a goal in a grid-like environment. Working method includes that Imagine you have a grid where each cell can either be empty (you can move through it) or blocked (you can't move through it). You start at a specific cell and want to reach another cell on the grid. A* helps you find the shortest path by evaluating each cell based on two factors: Cost from Start (g): This is the actual cost to move from the starting cell to the current cell. It starts at zero for the initial cell and increases with each step taken. Heuristic Estimate : This is an estimate of how far the current cell is from the goal cell. In the A* algorithm, a common heuristic is the Manhattan distance, which measures the total number of grid cells moved horizontally and vertically to reach the goal. Starting from the initial cell, A* explores neighboring cells one by one. For each neighboring cell:It calculates its cost (cost to reach from the starting cell). A* selects the cell with the lowest value from the open list and repeats the process until it reaches the goal cell or exhausts all possible paths. Along the way, it keeps track of each cell's parent to reconstruct the shortest path once the goal is found. Once the goal cell is reached, A* retraces its steps back through each cell's parent pointers to reconstruct and display the shortest path from the starting cell to the goal cell. This path can then be used for navigation or further analysis. An autonomous vehicle might use A* to plan a route through city streets, considering traffic patterns, road closures, and other dynamic factors. If an unexpected obstacle appears, other algorithms in the vehicle's system would detect and classify it, prompting a reassessment of the planned route using A* or alternative path-planning strategies. Thus, while A* itself focuses on optimal path finding, its integration with other algorithms is crucial for handling edge cases and ensuring safe operation in real-world scenarios. Fig.3. represents the overview of A* Algorithm

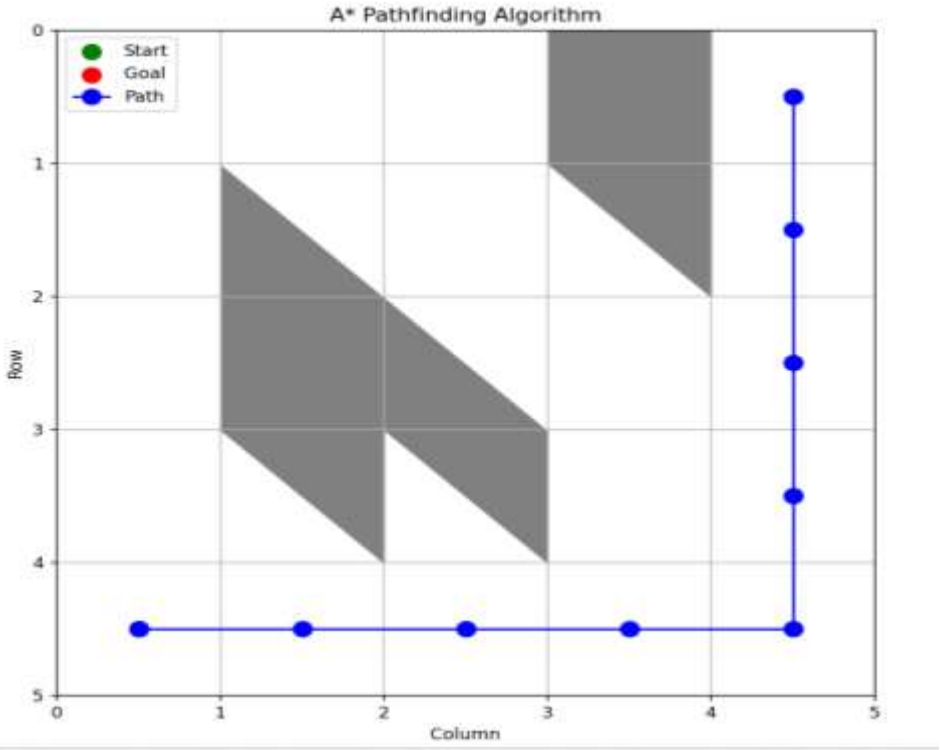

Fig.3. Overview of A* path finding Algorithm

In CNN to start working with our dataset, we need to import the data, which includes around 40,000 images and a CSV file. These images were collected using a simulator equipped with three cameras mounted on the car, taking photos simultaneously. The CSV file consolidates the images captured at the same time and includes metadata such as steering angle, throttle position, reverse status, and speed values for each image. To understand the data more deeply, we will visualize it using a histogram. We will upload the image paths and their steering values into lists called "image paths" and "steerings". We will define a function to handle this process. After the data is ready, we will split it into training and validation sets, with 10,764 training samples and 2,691 validation samples. Before coding the Convolutional Neural Network (CNN) model, we need to augment the data to increase variety and noise. This helps prevent overfitting and improves model performance. Each function performs a specific transformation on the images, such as zooming in, panning, adjusting brightness, or flipping the image horizontally along with its steering angle. To increase randomness in our data augmentation, we will define a function called "random_augment", which randomly applies one or more of these augmentations to the images. Finally, we will preprocess the images by cropping, converting color space, applying Gaussian blur, resizing, and normalizing the images. This preprocessing ensures that the images are in a suitable format for the CNN model. The batch_generator function is designed to create batches of images and their corresponding steering angles, continuously generating these batches for training or validation purposes. It starts by initializing empty lists for storing images and steering angles. For each image in the batch, a random index is selected to pick an image and its steering angle from the dataset. If the training flag is set to true, the image undergoes random augmentation to increase data variety. If the training flag is false, the original image is used without augmentation. Each image is then pre processed to ensure it is in the correct format for the model. These pre processed images and their steering angles are added to the respective lists, and once the batch size is reached, the function yields the batch as numpy arrays. To test the batch generator, we generate one batch of training data and one batch of validation data. We then visualize the first image from both batches using a plotting library. This visualization helps verify that the images are processed and augmented correctly. Additionally, we plot the training and validation loss values over each epoch to monitor the model's learning performance and check for overfitting. The combination of batch generation, image pre processing, and performance visualization ensures that the model is trained effectively with diverse and well-prepared data.

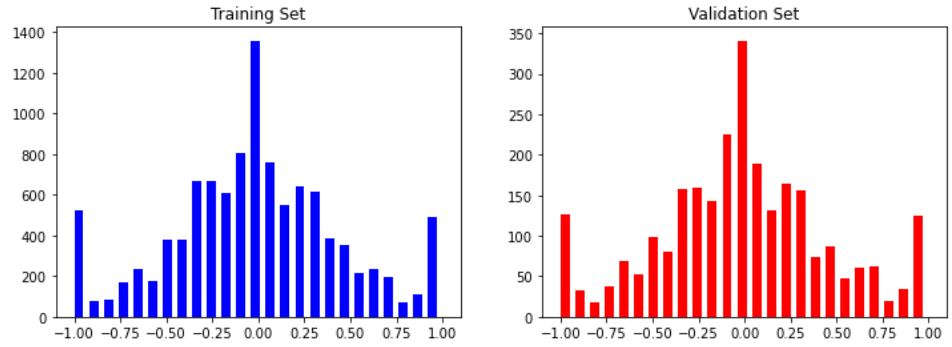

Fig.4. Visualization of training and validation set

Below Figures shows augmentation of the data to increase the variety and noise. Therefore, we need some functions that are going to be useful, such as "zoom", "pan", "image brightness", and "image flip" etc.

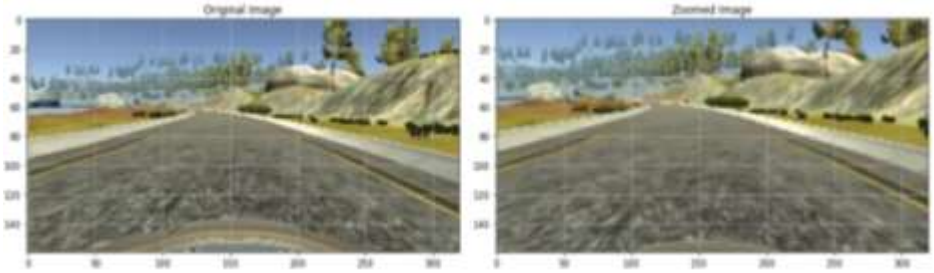

Fig.5. Visualisation of original image and zoomed image

The below figure shows the plots of the training and validation loss values over each epoch to visualize how the model's performance changes during training. It shows a line graph with two lines, one for training loss and one for validation loss, to help compare and understand the model's learning process.

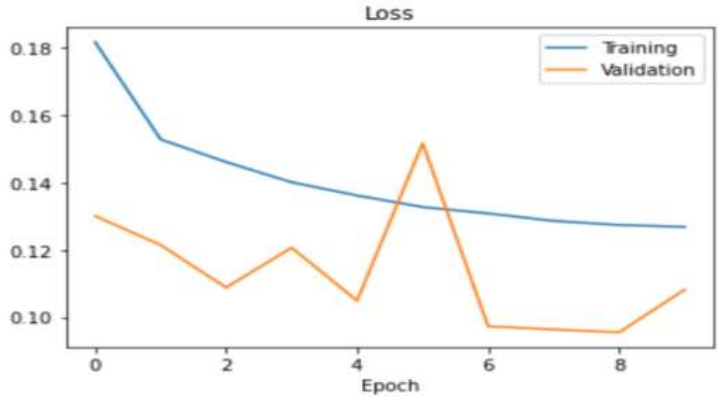

Fig.6. Epoch to visualize model performance

Table.1. Results of models

Model	Accuracy	Precision	Recall	F1 score	Specificity
Anomalies Neural Network	0.93	0.6154	0.80	0.6923	0.94.44
A* Path Finding	0.84210	0.8333	0.9090	0.8695	0.75
Convolutional neural network	0.92	0.65	0.78	0.7089	0.93

5 Conclusion

When comparing algorithms for detecting edge cases in autonomous vehicles, each has unique strengths and challenges. Anomalies Neural Networks are excellent for identifying rare and unexpected events by detecting deviations from normal data patterns, making them versatile and capable of handling various sensor inputs like LIDAR and radar. However, they require high-quality, diverse datasets and can be complex to interpret. The A* Path finding Algorithm excels in efficient route planning within grid-based environments, quickly finding the shortest path and adapting to dynamic obstacles. Its performance may decline in highly complex or non-grid environments and needs optimization for real-time processing. Convolutional Neural Networks (CNNs) are highly accurate in detecting and classifying objects such as pedestrians, vehicles, and traffic signs, providing crucial real-time information for navigation and safety. However, they demand significant computational resources and may struggle with edge cases if not trained on diverse datasets. Each algorithm offers distinct advantages: Anomalies Neural Networks for detecting unforeseen scenarios, A* Pathfinding for optimal navigation, and CNNs for precise object recognition, all contributing to the overall robustness and reliability of autonomous vehicles.

References

1. N. Philippe. Safety-critical scenarios and virtual testing procedures for automated cars at road intersections. diss. loughborough university, 2018. URL https://hdl.handle.net/2134/34433.
2. Automated vehicles, Nov 2021. URL https:// www.transportstyrelsen.se/en/road/Vehicles/self-driving-vehicles/.
3. M. S. Alam. Automatic generation of critical driving scenarios. KTH, School of Electrical Engineering and Computer Science (EECS) (Master Thesis), 2020. URL http://urn.kb.se/resolve?urn=urn:nbn:se:kth: diva-288886.
4. Jean-Paul Skeete. Level 5 autonomy: The new face of disruption in road transport. Technological Forecasting and Social Change, 134:22–34, 2018. ISSN 0040-1625. doi: https://doi.org/10.1016/j.techfore.2018.05. 003. URL https://www.sciencedirect.com/science/article/pii/ S0040162517314737.
5. P. Helle, W. Schamai, and C. Strobel, “Testing of autonomous sys-tems - challenges and current state-of-the-art,” INCOSE InternationalSymposium, vol. 26, pp. 571–584, 07 2016
6. Petit, J., Stottelaar, B., Feiri, M. and Kargl, F., “Remote attacks on automated vehicles sensors: Experiments on camera and lidar, ” Black Hat Europe, vol. 11, no. 2015, 2015.
7. “LIDAR Hacks Fairly Unlikely Attacks on Self-Driving Cars, ” 2015. [Online]. Available:http://www.roboticstrends.com/article/lidar_hacks_fairly_unlikely_attack_on_self_driving_cars
8. A. Choudhary, “A Hands-On Introduction to Deep Q-Learning using Open AI Gymin Python,” 2019. [On-line]. Available: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep- q-learning-python/[22] .
9. Road vehicles -- Functional Safety -- Part 2: Management of functional safety, ISO 26262-2:2011, Nov. 15, 2011.
10. Road vehicles -- Functional Safety -- Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses, ISO 26262-9:2011, Nov. 15, 2011.
11. P. Koopman and M. Wagner, “Challenges in autonomous vehicle testing and validation,” SAE International Journal of TransportationSafety, vol. 4, no. 1, pp. 15–24, 2016.
12. Philip Koopman, Uma Ferrell, Frank Fratrik, and Michael Wagner. A Safety Standard Approach for Fully Autonomous Vehicles, pages 326–332. 08 2019.