

**Torc Robotics:  
Rain Detection Solution "RainSense AI"**

Alyssa Lowe

Darius Holland

Ethan Weaver

Jason Andrade

Jonathan Samuel

Minho Cho

<b>1. Product Demonstration</b>	<b>3</b>
<b>2. Product Description</b>	<b>3</b>
<b>3. Product Functionalities</b>	<b>3</b>
<b>4. Design</b>	<b>5</b>
4.1 Convolutional Neural Network (Image Prediction)	5
4.2 Anomaly Detection Algorithm (LiDAR Prediction)	8
4.3 Ensemble Learning (Final Product)	12
<b>5. Updates</b>	<b>14</b>
<b>6. Retrospection</b>	<b>15</b>
6.1 Accomplishments, What Went Well	16
6.2 Challenges, What Went Wrong	17
<b>7. Recommendations for the Future</b>	<b>18</b>
7.1 Data Variety and Quality	18
7.2 Additional Sensors	19
7.3 Robust LiDAR Models	19
<b>8. References</b>	<b>20</b>

# 1. Product Demonstration

The following is the video demonstration link to the “RainSense AI” product we have developed for the Virginia Tech Capstone Project. You can click [here](#) to watch a YouTube recording of our demonstration where we are running the production-ready solution within a ROS (Robot Operating System) environment, which is the same environment used by Torc’s autonomous vehicles. The link to the video can also be found as a commit to our GitLab [Wiki](#)

To view a repository of our raw code, as well as historical project deliverables, please refer to our project group’s GitLab repository [here](#).

# 2. Product Description

Torc Robotics makes autonomous vehicles, specifically self-driving trucks. As such, they hope to make their vehicles perform as safely and efficiently as possible. This task becomes more difficult when the vehicles are subjected to adverse weather conditions, such as rain. The first step in making these vehicles perform better in rainy conditions is to give the vehicle the ability to determine whether it is currently “raining” or “not raining”. This information could then help in informing the vehicle to adjust its maneuvers based on the presence of rain.

The machine learning solution we developed can make highly accurate predictions as to whether there is rain present in the vehicle’s localized environment. “RainSense AI” is an ensemble learning system that takes in data from various sensors already present on the vehicle. Namely, it incorporates image data from the vehicle’s front-facing camera and LiDAR point-cloud data from the vehicle’s two LiDAR sensors. This data is passed into our system, which consists of two main parts: a Convolutional Neural Network that processes the image data, and an Anomaly Detection algorithm that processes the LiDAR data. Each of the two processes can make individual predictions regarding the presence of rain.

Our system uses both of these predictions to generate a final prediction as to whether there is currently “rain” or “no rain”. This system can run in real-time, taking in the most current data from the vehicle’s sensors and continuously generating predictions.

# 3. Product Functionalities

Our product performs one main function: generating predictions on whether or not rain is present in the environment. In a real-world scenario, this system would be running as a component of an autonomous vehicle. As such, it has no real “user” (unless you count the autonomous vehicle as a user).

The Torc autonomous vehicles use the Robot Operating System (ROS) in order to share data across various components within the vehicle. Information is shared by publishing data to a ROS topic and/or subscribing to a ROS topic to read data from it. The sensors on board the vehicle publish their readings to ROS topics to make it available to other components within the vehicle.

Our product subscribes to the ROS topics that contain the data published by the forward-facing camera and the two front-facing LiDAR sensors on the vehicle. It uses the most current readings from these three sensors to generate a prediction as to whether the outside environment is raining or not. It outputs this prediction to a separate ROS topic reserved solely for these predictions. As soon as it has output a prediction, it reads the new most current sensor data and generates a new prediction.

The output of our product can easily be subscribed to by any other component within the autonomous vehicle. The information contained in its predictions could be used for a variety of purposes. Theoretically, our solution's predictions would be used to alter the behavior of the self-driving vehicle to maneuver more carefully when the weather is rainy. The data flow of our solution within a ROS environment can be seen in Figure 1.

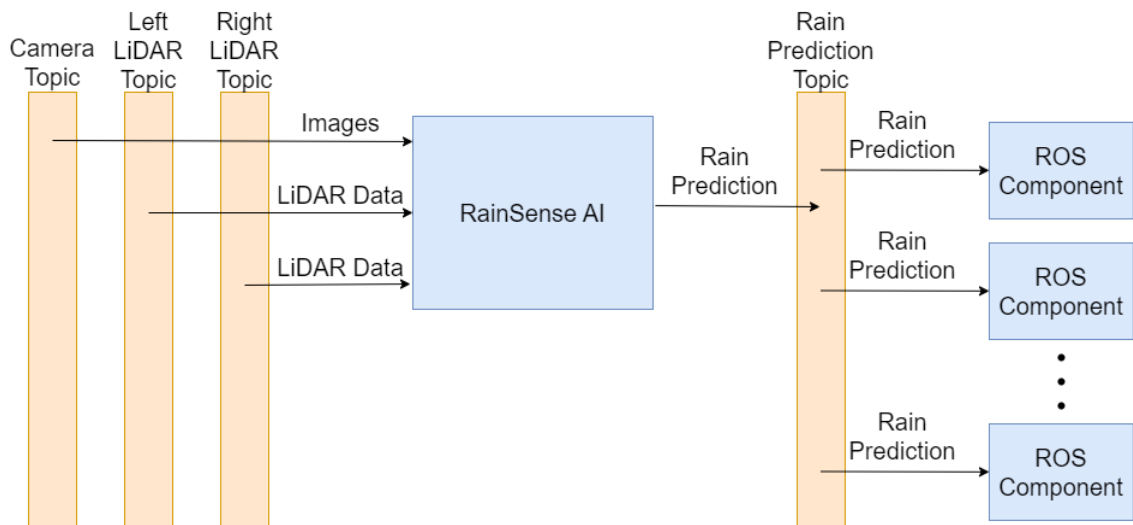


Figure 1: Data Flow of the Product in a ROS Environment

Another important feature of our product is that it is capable of utilizing GPUs to improve its efficiency and speed up its processing time. Torc's autonomous vehicles contain GPUs, so in a real-world environment, our solution would be able to use this hardware to ensure quick and consistent predictions.

Through testing, we have determined that this solution generates predictions with approximately 91% accuracy, meaning its prediction will be correct approximately 91% of

the time. This means that it will sometimes generate incorrect predictions, either failing to detect rain or incorrectly detecting rain. This accuracy could be improved by using further data to retrain the model within the solution. However, due to the nature of the sensor data, an accuracy of 100% is likely impossible to achieve.

## 4. Design

Our product contains two main components. The first is a Convolutional Neural Network (CNN) that takes in image data and generates a prediction as to whether or not rain is present in that image. The second component is a LiDAR anomaly detection algorithm that takes in LiDAR point-cloud scans and generates a prediction as to whether or not rain is present in that scan. We will explain each of these components in depth.

### 4.1 Convolutional Neural Network (Image Prediction)

A CNN is a type of machine learning model that is well-suited for processing image data. We created our CNN by using PyTorch, a Python library used for deep learning. We trained this CNN on a dataset of images that came from real driving sessions recorded by Torc autonomous vehicles.

Figure 2 shows the architecture of our CNN. It consists of 2 convolutional layers and 2 max pooling layers. The convolutional layers break the image up into sections; in our case, both of our convolutional layers break the image into sections of 5 pixels by 5 pixels. The max pooling layers reduce the dimensionality of the input by taking the max values of small sections of the input; in our case, we take the maximum value of each section of 2 pixels by 2 pixels. These sections are then flattened so that they become linear, and are passed into the neural network. Our neural network has 3 linear layers. The first contains 120 nodes, the second contains 84 nodes, and the third contains 2 nodes. These final 2 nodes represent the final output of the CNN model; one node corresponds to a “rain” prediction, and the other corresponds to a “no rain” prediction.

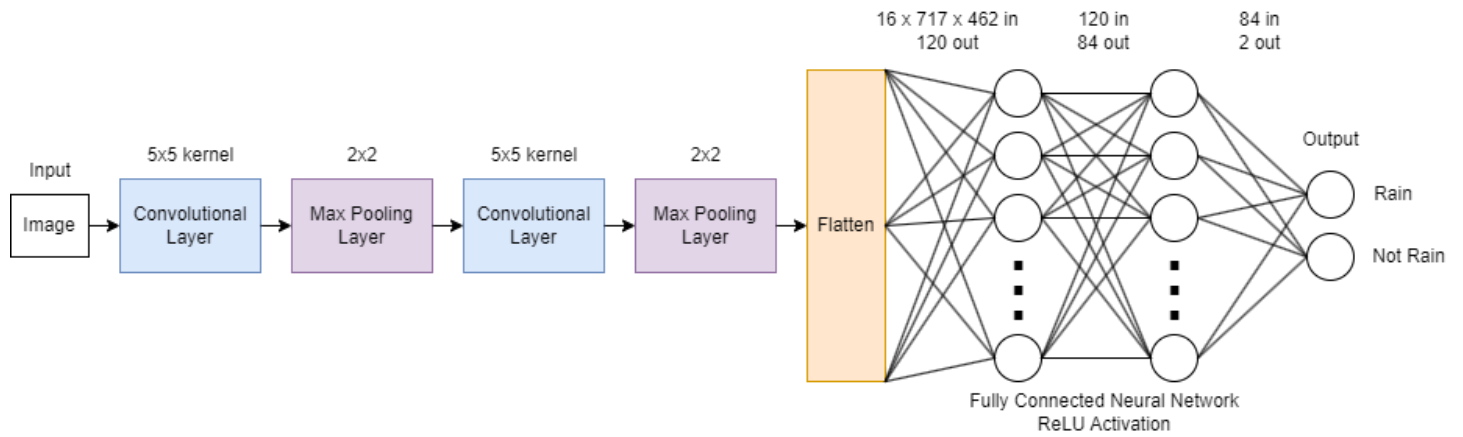


Figure 2: Architecture of the Convolutional Neural Network

When training our CNN, we split our training data into 80% training data, 10% validation data, and 10% testing data. The validation data allowed us to keep an eye on the performance of our model while it was training and allowed us to stop the training early if it looked like the model might be overfitting. The testing data allowed us to gauge the performance of our model after it had finished training. Our entire dataset consisted of 22,084 images, 12,583 of which contained rain and 9,501 of which did not contain rain.

All of this data came from ROSbag files, which each contain a recording of sensor data during a single driving session of a Torc autonomous vehicle. When splitting the data into train, validation, and test sets, it was important to keep all of the data from each individual bag together. In other words, we split the data on bags, not on individual images.

The reason for this is that all images in a bag come from the same recording, and subsequent images are taken fractions of a second apart from each other. This makes many of the images within a bag closely resemble each other. If some of the images from a bag were placed in the training set, the CNN would learn to recognize those images as either “rain” or “not rain”. Then, when it saw the other images from that bag in the test set, it would recognize that the images looked similar to the ones it saw in training, and know whether or not they were “rain” or “not rain”. This would lead to overfitting of the model, and it would only perform well on images that were extremely similar to the images in its train set. When testing the model, we would see incredibly high performance, but this is only because our test set is so similar to the train set. To get an accurate gauge of the model’s true performance, it needs to be tested on data it has never seen before. So, to avoid overfitting and to get accurate results when testing, we need to ensure all of the images from each bag go into either the train, validation, or test set and are not split between multiple sets.

When training this model, we attempted to use image augmentation as part of the training process. This is a process that involves transforming some of the training images to artificially increase the variety of training data. These transformations can

include rotating, flipping, and color-shifting of images, just to name a few. We tried performing several different types and combinations of transformations on our training set. However, each model we trained on transformed images performed worse than our initial model that was trained on untransformed images. So, in the end, we determined that image augmentations were not helpful for the training of this specific model.

We can see the results of our final CNN model in Figure 3 and Table 1. Figure 3 depicts its confusion matrix, which shows how often the model predicted either “rain” or “no rain” compared to what the correct prediction should have been. Table 1 shows multiple statistics regarding our model’s performance, represented as percentages. The most important of these is Accuracy, which is 84%. This means our model generates the correct prediction 84% of the time. This is very good for this type of model.

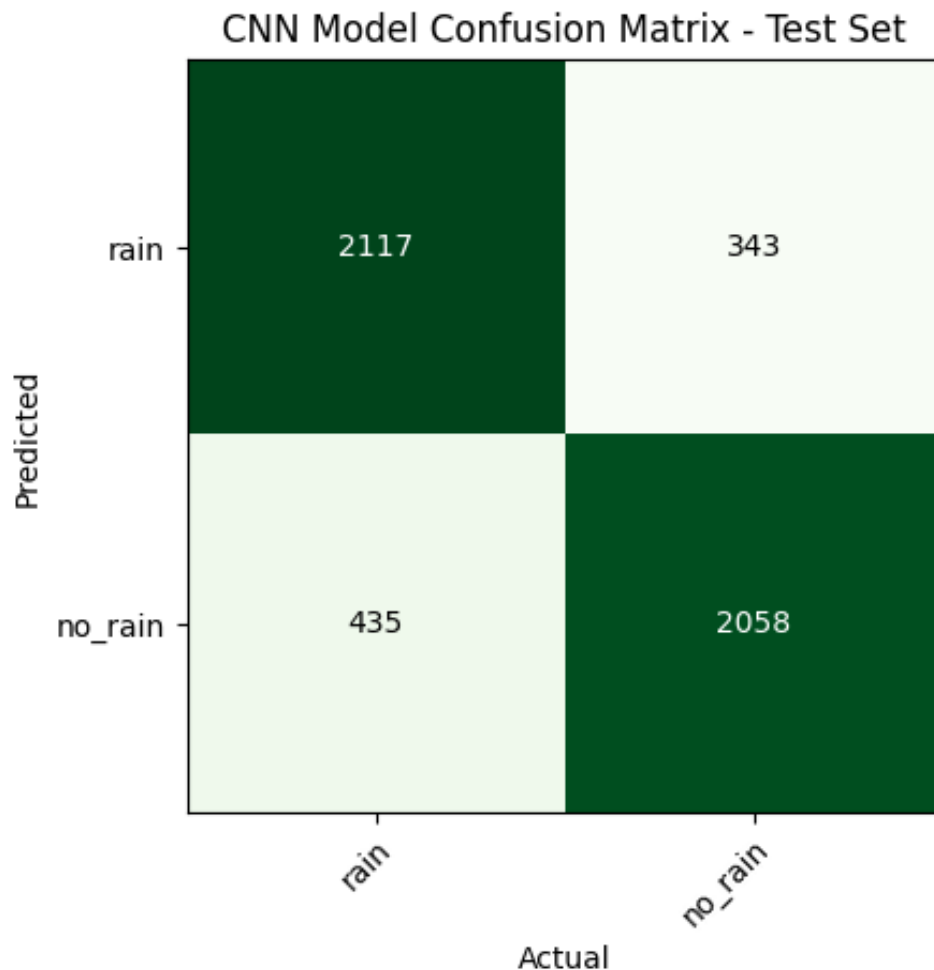


Figure 3: Confusion Matrix for the CNN Model

Accuracy	84%
Sensitivity	83%
Specificity	86%
Precision	86%

Table 1: Performance Statistics for the CNN Model

## 4.2 Anomaly Detection Algorithm (LiDAR Prediction)

Rather than attempting to train a machine learning model using LiDAR data, our team instead opted to use an algorithmic approach. The main idea for this algorithm comes from a paper published by Zhang, C. et al [1]. This paper describes a mathematical approach for determining whether LiDAR data contains an “anomaly”, or abnormal patterns. As we have discovered, rainy conditions appear as an anomaly that this algorithm can detect.

The LiDAR data we obtain from the autonomous vehicle includes x, y, and z coordinates and intensity values. To begin this algorithm, we convert these values into spherical coordinates (range, azimuth, and elevation). Post-conversion, we discretize the data into 2.5 by 2.5 azimuth-elevation grid cells.

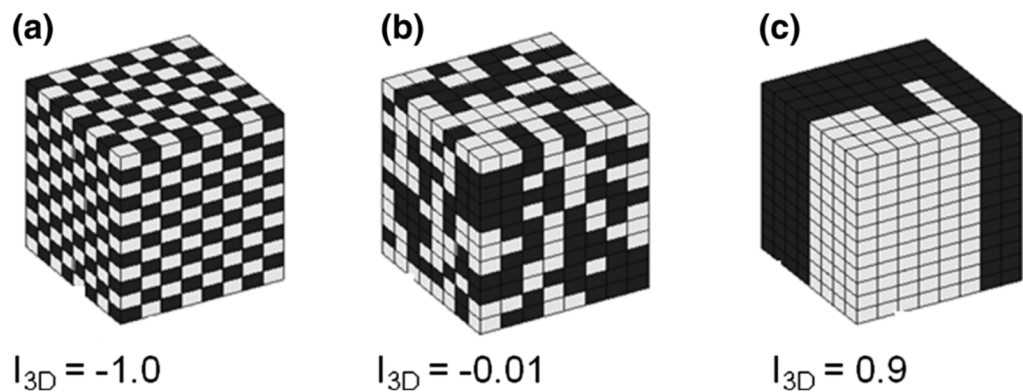


Figure 4: Spatial autocorrelation for different voxel states:  
 (a) negative spatial autocorrelation for a perfectly uniform pattern (b) no spatial autocorrelation for a random pattern (c) a strong positive correlation for a clustered pattern [2]

In each of these cells, we assess the spatial dispersion of points. LiDAR point clouds can generate random, sparse false positives in rainy conditions. These anomalies can be

identified by their distribution in three-dimensional space. To tackle this, we use spatial autocorrelation, which measures the degree of spatial dispersion of LiDAR points, helping to distinguish between genuine objects (like vehicles or pedestrians) and noise [1]. Genuine objects tend to produce clustered distance values due to their reflective surfaces, whereas noise-dominated segments appear more dispersed [1,2].

Furthermore, LiDAR point clouds in rainy conditions often show reduced intensity values, which can be due to signal attenuation or hardware issues, leading to abnormally low or high-intensity readings [1]. Especially in conditions like heavy rain or dense fog, LiDARs can generate clustered noise points that might be mistaken for real objects. These noise points usually have extremely low intensity due to partial laser reflections. Most cases of high average intensity in LiDAR point clouds are typically due to retro-reflective targets, such as road signs, being in close proximity [1]. To address this, we integrate an intensity weight multiplier with the spatial autocorrelation. This multiplier, based on intensity metrics such as mean or standard deviation, helps in identifying and adjusting for abnormally low-intensity values. Essentially, this scales spatial autocorrelation values where average intensities are less than some reference nominal value. By definition, if average intensities are above the nominal value, this weight is 1 and greater than 1 otherwise thereby intentionally disregarding high average intensities.

The overall anomaly metric for a LiDAR data frame is calculated by summing up the weighted spatial autocorrelation values across all grid cells. This total is then averaged by the number of grid cells to provide a comprehensive measure of the data frame's anomalies.

$$\textit{Spatial Autocorrelation: } I = \begin{cases} \frac{N \sum_{i=1}^N \sum_{j=1}^N w_{ij} (r_i - r) (r_j - r)}{W \sum_{i=1}^N (r_i - r)^2} & N > 1 \\ -1 & N = 1 \end{cases}$$

$$\textit{Average Distance: } r = \frac{1}{N} \sum_{i=1}^N r_i$$

$$\textit{Inverse Angular Distance: } w_{ij} = \begin{cases} \left\| (\theta_i, \phi_i), (\theta_j, \phi_j) \right\|^{-2} & i \neq j \\ 0 & i = j \end{cases}$$

$$\textit{Sum of Weights: } W = \sum_{i=1}^N \sum_{j=1}^N w_{ij}$$

$r_i$ : range,  $\theta_i$ : azimuth,  $\phi_i$ : elevation,  $\gamma_i$ : intensity

$$\textit{Intensity Weight Multiplier: } K_\gamma = \exp \left( k \cdot \frac{\max(0, \gamma_{ref} - \bar{\gamma})}{\gamma_{ref}} \right)$$

$k$ : constant scale factor,  $\gamma_{ref}$ : reference intensity,  $\bar{\gamma}$ : average intensity

$$\textit{Anomaly Metric: } s = \frac{1}{VH} \sum_i^V \sum_j^H K_{\gamma,ij} \cdot I_{ij}$$

We developed this anomaly detection algorithm in Python, capable of both single and parallel operations. Initially, it processed spatial autocorrelation and weighted intensity multipliers for each grid cell using CPU-based vectorization. However, we have since transitioned to PyTorch modules, allowing this algorithm to be run with GPU acceleration for more efficient computation.

The anomaly class offers various adjustable parameters for users, including the gamma reference (a nominal intensity value), a scale factor (which adjusts the intensity multiplier), azimuth minimum and maximum window (defining the horizontal field of view or HFOV), elevation minimum and maximum window (for the vertical field of view or VFOV), and the size of the azimuth-elevation grid cell. To optimize these parameters for our dataset, we performed an extensive parameters assessment, supplemented by a literature review.

For instance, the gamma reference, which could be sourced from Torc or the LiDAR manufacturer, was instead determined by calculating the mean of all non-rain intensity values, approximately 407.5. The HFOV and VFOV parameters were derived based on the specifications of the LUMINUS IRIS LiDAR model, showing +/-60 degrees and +/-15 degrees for azimuth and elevation, respectively. The remaining parameters – scale factor and grid cell size – were established through empirical grid search assessment. Note the grid search operation generally is conducted by selecting parameters that minimize the loss or objective function. Rather than loss, our goal is to maximize the separation between non-rain and rain mean anomaly scores. However, the loss would be too complex to incorporate programmatically but can be added in the future. Given this, the assessment was concluded qualitatively rather than quantitatively.

We performed a grid search assessment for grid cell size and scale factor. The results of this search led us to use a grid cell size of 2.5 and a scale factor of 5 in our final implementation of this algorithm. These parameters resulted in the largest discernible difference between rain and non-rain data. With these parameters, we determined that we could set a score threshold of 10, when combining both left and right sensor data, to best differentiate between rain and non-rain cases. This means we consider a score larger than 10 to indicate the presence of rain and a score less than 10 to indicate non-rain.

We can see this behavior in Figure 5. The majority of rain cases lie above the threshold of 10, while the majority of non-rain cases lie below the threshold of 10. However, it is worth noting that there are still a sizable number of rain cases that result in a score of less than 10.

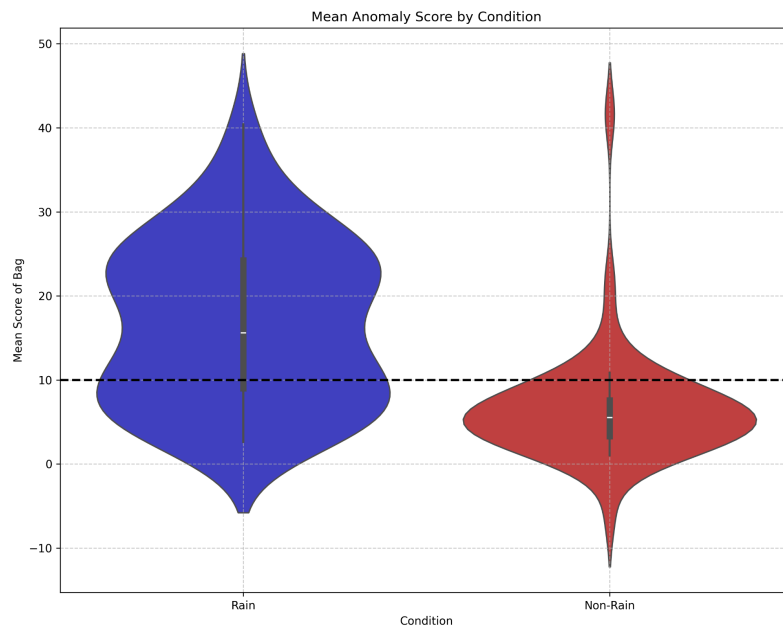


Figure 5: Graph Showing Anomaly Scores of Rain and Non-Rain Data

By using this algorithm with a threshold of 10 as a classifier, we can evaluate the performance through accuracy. Because this is not a model and instead a metric itself, we can apply this over our entire dataset rather than sacrificing a portion of our dataset for training. Figure 6 shows the confusion matrix when applying our algorithm to the entire dataset. The performance statistics are detailed in Table 2. It is worth noting that using this algorithm as a classifier generates relatively few false positives (predicting rain incorrectly), but quite a few false negatives (failing to predict rain). This leads to a relatively low sensitivity score, but relatively high specificity and recall scores. Overall, this algorithm performs worse than our image classifier at detecting rain, with an accuracy of 74%. However, this model is generalizable as it's not fitting to any specific training dataset, a possible downfall in our image classifier.

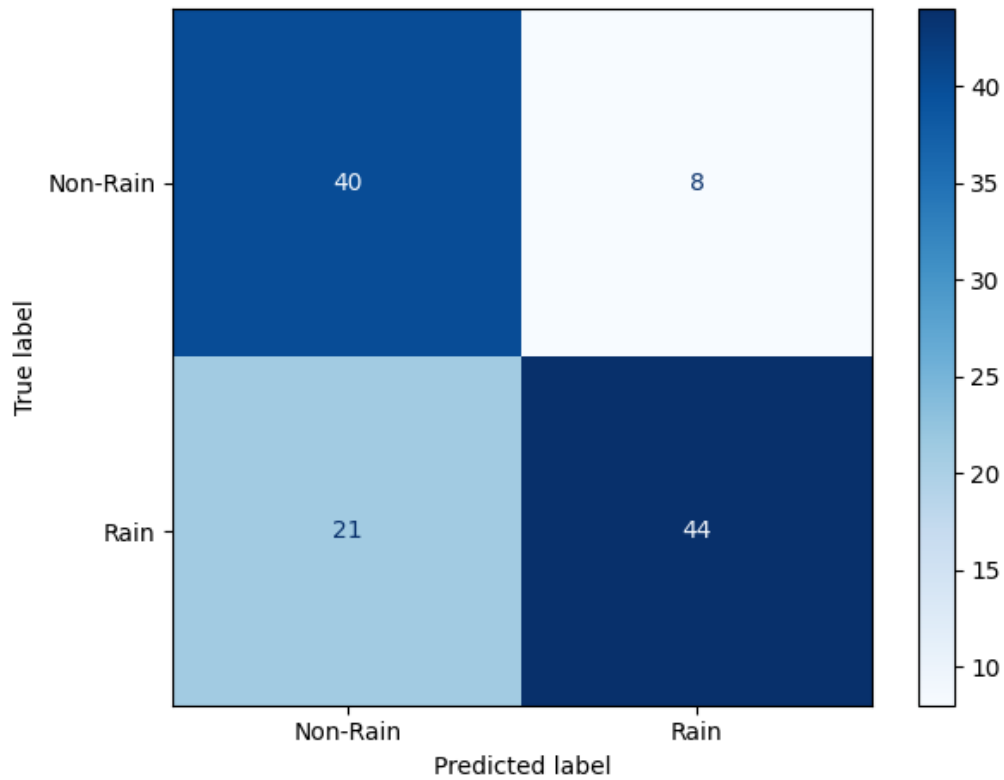


Figure 6: Confusion Matrix for LiDAR Anomaly Detection Classifier

Accuracy	74%
Sensitivity	67%
Specificity	83%
Precision	84%

Table 2: Performance Statistics for LiDAR Anomaly Detection Classifier

### 4.3 Ensemble Learning (Final Product)

With our CNN model and anomaly detection algorithm, we have two different ways of detecting rain. However, by combining the two solutions together, we were able to achieve even better results.

As mentioned earlier, our LiDAR anomaly detection algorithm produces few false positives, but many false negatives. As such, if it predicts “rain”, we can be reasonably

certain that that is a correct prediction. We can even adjust its score threshold in order to minimize its false positives, to make us even more confident in its “rain” predictions. On the other hand, if it predicts “no rain”, we are not very confident that that is a correct prediction. So, in that case, we can instead default to the CNN model's prediction, as it has a higher overall accuracy.

This is precisely the process we implemented in our final solution, as can be seen in Figure 7. It first uses LiDAR anomaly detection. If the algorithm predicts “rain”, our solution outputs that as the final prediction. However, if the algorithm outputs “no rain”, we query the CNN model, and whatever it predicts is used as the final prediction.

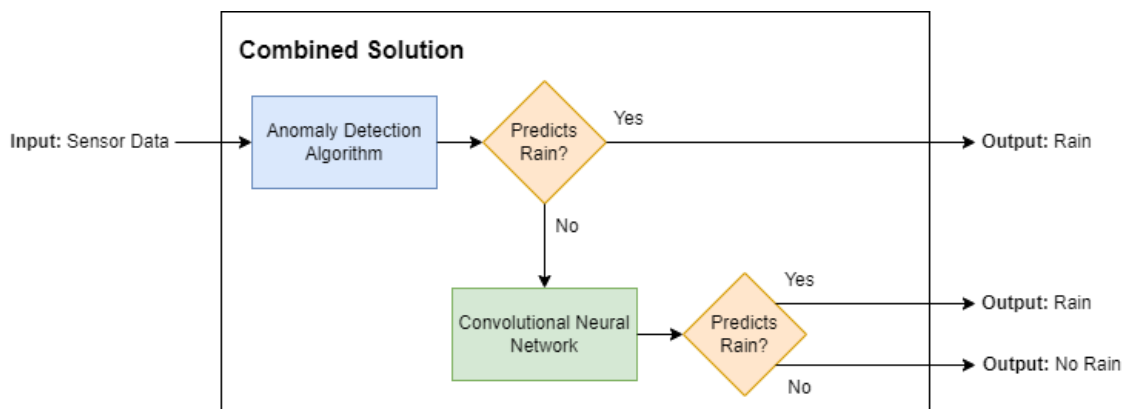


Figure 7: Functionality of the Combined Solution

Keep in mind that this combined solution is what is running in the ROS environment described in Section 3. It will continuously use the most up-to-date data from the vehicle's sensors to make consistent predictions.

The overall performance of this combined solution can be seen in Figure 8 and Table 3. The combination of the two classifiers results in better performance than either one alone. We achieved an overall accuracy of 91%, with very few false negatives, which we are very satisfied with.

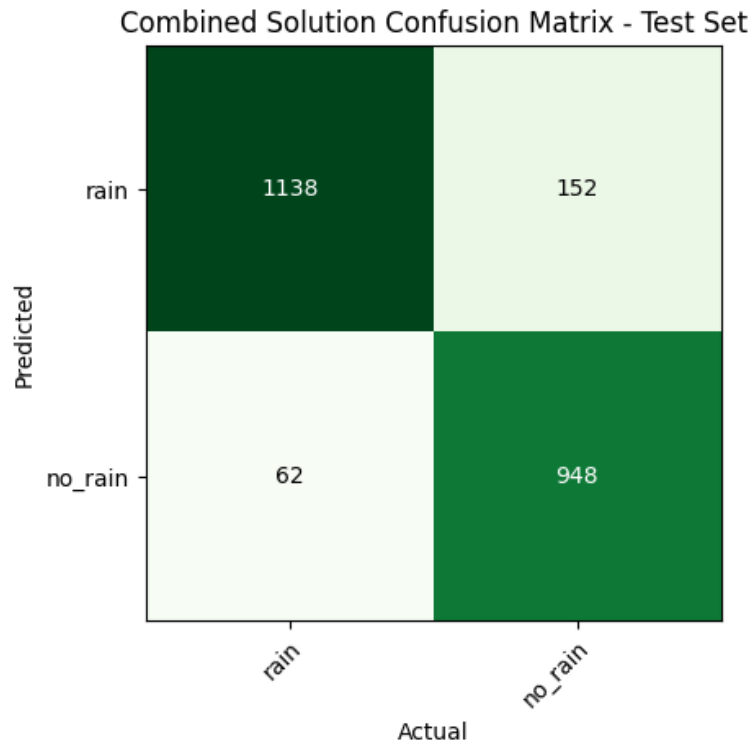


Figure 8: Confusion Matrix of Combined Solution

Accuracy	91%
Sensitivity	95%
Specificity	86%
Precision	88%

Table 3: Performance Statistics for Combined Solution

As mentioned previously, a video demonstration of this final product in action can be seen [here](#).

## 5. Updates

Our third and final sprint consisted of three major tasks. First, we combined our existing CNN and LiDAR detection solutions into a combined solution. In addition to

implementing a combined solution, we also tuned its parameters to optimize its accuracy and tested its performance. This resulted in a better-performing solution than either the CNN or anomaly detection algorithm on their own. This task is described in detail in Section 4.3

Second, we implemented this solution within a ROS environment. This implementation reads from the ROS topics corresponding to image and LiDAR data, and writes its rain predictions to a separate ROS topic. It uses the most up-to-date sensor data to continuously generate predictions. This will allow Torc to easily integrate our solution into their autonomous vehicles. This solution is described in detail in Section 3.

Finally, we refactored our LiDAR Anomaly Detection algorithm to use PyTorch rather than NumPy. This change will allow the anomaly detection algorithm to run on GPU rather than CPU. This results in more efficient computation and faster processing times, which will allow for more frequent predictions in a real-world scenario. Additionally, this abides by the stakeholder's need to run a classifier on GPUs only. We originally expected this task to be a small one, but it ended up being much more tedious and time-consuming due to the lack of one-to-one functions from NumPy to PyTorch.

We concluded Sprint 3 having closed out all outstanding tasks. There are no remaining tasks to be maintained in our backlog. As is expected when working in an Agile environment, we completed the last Sprint having also performed a Review at the Retrospective Meeting with our project stakeholders.

## 6. Retrospection

The third and final sprint of RainSense AI resulted in a successful, timely project that satisfied the final user story:

*As a **Data Scientist** at Torc,*

*I want to deploy an application that uses **an image** data classifier and an algorithm for **LiDAR** spatial data to determine the current state of the vehicle as **raining or not raining***

*So that I can give the autonomous vehicle **useful information**.*

During this sprint, the team worked concurrently to meet project deadlines, explore hyperparameters for the most accurate solution, and reframe our code base to meet project requirements. In the following sections, we will discuss the successes and challenges faced throughout the project as a whole.

## 6.1 Accomplishments, What Went Well

Throughout this project, our team has completed our original goal of developing a joint CNN and LiDAR rain detection solution that passed our expectations for accuracy. This goal was accomplished by maintaining a steady flow of communication with the Torc team, who is very involved with our task prioritization, and maintaining sprint deadlines. This has allowed the team to communicate our needs and concerns, as well as request feedback every week.

During this project, several accomplishments allowed us to be successful and work according to the schedule:

### 6.1.1 Communication

Communication was an integral part of the success of this project. Frequent communication with advisors, stakeholders, and within the team allowed us to thoroughly explore our implementation options, discover the needs of Torc Robotics, and meet deadlines. Specifically, weekly meetings with the Torc Stakeholders were used to contextualize use cases and discover solution requirements that were not initially described by Torc. For example, in the last sprint, GPU support was added to the LiDAR algorithm, and ROS environment support was added. These features are essential for Torc to have the ability to integrate our solution into their pipeline.

### 6.1.2 Concurrent Work

Concurrent work allowed our group of six people to maximize the resources that we put toward development and meet course requirements simultaneously. This required that team members work in parallel with significant communication.

### 6.1.3 Research

During the first two sprints of this project, much of our time was spent researching and testing different implementation options. This includes external data sources, different image resolutions, image augmentation, ML with LiDAR, voxelization, hyperparameter adjustments, etc. This research allowed us to feel confident that we chose the best implementation for this project.

### 6.1.4 Minimum Viable Accuracy and Beyond

The baseline goal of this project was to reach an accuracy of at least 80%. This minimum viable accuracy was reached with the CNN and was blown away with the Ensemble solution with 91% accuracy.

### 6.1.5 Well Presented

Throughout this project, our team spent considerable effort in presenting and demonstrating our project which is an integral part of development with stakeholders. Our project was well presented and received by our peers, advisors, and stakeholders.

## 6.2 Challenges, What Went Wrong

Every project faces challenges and during this project, our team faced several challenges that resulted in adjusting our expectations, timeline, and goals for the project. During the first sprint, we experienced significant delays in data access and a learning curve for the data that we received. Our first sprint was primarily focused on collecting requirements and creating tasks because we did not have access to client-specific data that could not be replicated. Once we received this data, we were faced with a concern that our goal of determining Rain or No Rain from image data would be impossible because the images we were given were difficult to differentiate. During Sprint 1, we also struggled to manipulate the LiDAR data type because the research related to LiDAR is not robust or mature. These challenges caused our team to re-scope and drop some of our “reach goals”. During this timeline setback, the team was struggling to adjust to the agile framework required for the coursework but resolved our project management struggle with communication and advice.

During Sprint 2, our team finally had access to data but was frustrated with the lack of variety. At this stage, we experienced a large learning curve for creating a CNN especially due to the lack of data variety. As a result, we found overfitting to be problematic because of the insignificant rain indicators in the images provided. Through communication, our team was able to resolve some of the data variety issues with the Torc stakeholders much faster than our original data access challenge. During this time we began to work more with LiDAR data but continued to find the data type challenging. We explored several algorithms to approach LiDAR anomaly detection but the limited LiDAR research led us to use the solution demonstrated in this project. During sprint 2, our team struggled the most with working simultaneously and meeting deadlines but we took note of our difficulties and learned from these mistakes.

Lastly, During Sprint 3, our team finally resolved many of the major setbacks during this project. With a re-scoped goal, we met all of our project requirements by the deadline. In Sprint 3, we spent more time focusing on making a viable, usable solution for Torc and after probing for missing project requirements we found several tasks that we had not planned for. These missing requirements are a demonstration of our successful communication throughout the project but are a sign that we should have spent more time in the planning stage of this project. Fortunately, we had the time available to add these tasks to our last sprint and create a usable product for Torc.

Overall, the challenges we faced during this project caused a large setback in our goals for this project but we recovered well from the challenges.

## 7. Recommendations for the Future

The image classifier and algorithm we have developed perform extremely well as mentioned throughout the document, but they still provide false positives from time to time. For instance, when it deals with data that contains a gray sky and wet ground, it falsely predicts rain even in situations when it is not raining.



Figure 9: False Positive Case Image

To further improve our developments, we suggest the following recommendations.

### 7.1 Data Variety and Quality

Several efforts have been made throughout the semester to increase the dataset with meaningful data. We believed that the initial dataset we received was skewed toward daylight conditions. Because of this, we had requested data that was taken during the night so that the classifiers/detectors were not making decisions based on other commonalities such as bright skies equaling no rain, etc. We believe that more diverse data from different periods of the day will help improve the prediction rate. We also re-attempted data augmentation to further diversify the data we had using the image augmentation feature provided in PyTorch. This was meaningful in the sense that we were able to train the model without having to request additional data from Torc while still

seeing improvements. More precise transformation and augmentation may lead to improvements as well. Lastly, along with the datasets that Torc provided us with, we believe introducing external datasets will help improve the prediction rate of the current model/algorithm we have even further. However, finding datasets that as such may be challenging as specific criteria (such as image size and resolution) have to be met to be used for the model and algorithm.

## 7.2 Additional Sensors

Some modern cars are equipped with infrared (IR) sensors near the rearview mirror that are used to detect rain and automatically activate windshield wipers. The IR sensor sends out a beam of light to detect any water droplets on the windshield and determines if it is raining or not by the different angles it receives back. It also obtains information regarding the intensity of precipitation so that it can adjust the wiper speed depending on the vehicle's current speed. Similar to how we have combined the image prediction method using CNN with the anomaly detection algorithm using LiDAR data to bring up the prediction rate, we believe adding data from an IR sensor to our solution will greatly improve the performance of our current solution.

## 7.3 Robust LiDAR Models

It would be interesting to attempt to create a Machine Learning that takes in LiDAR data and attempts to predict the presence of rain in a LiDAR scan. If it shows better performance, a model such as this could replace the anomaly detection algorithm in our current solution. This type of model would require significant preprocessing of the LiDAR data. Voxelization, which imparts structure to the inherently unordered and unstructured point cloud data, has emerged as a significant area of interest. In future work, we aim to investigate the integration of diverse voxelization methods as a preliminary processing step in conjunction with CNNs, following the paradigm established by VoxNets [3]. Additionally, we plan to examine alternative strategies that engage directly with point cloud data, inspired by the methodologies of PointNet and PointNet++ [4, 5].

## 8. References

[1] Zhang, Chiyu et al. "Detecting the Anomalies in LiDAR Pointcloud." ArXiv abs/2308.00187 (2023): n. Pag.

[2] Jjumba, Anthony and Suzana Dragičević. "Spatial indices for measuring three-dimensional patterns in a voxel-based space." Journal of Geographical Systems 18 (2016): 183-204.

[3] Maturana, Daniel and Sebastian A. Scherer. "VoxNet: A 3D Convolutional Neural Network for real-time object recognition." 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2015): 922-928.

[4] Qi, C. et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 77-85.

[5] Qi, C. et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." Neural Information Processing Systems (2017).