DOI: 10.37943/20MNZJ4935

Akmaral Kuatbayeva
Ph.D. in Computer science, assistant-professor at the Computing and Data Science department
a.kuatbayeva@astanait.edu.kz, orcid.org/0000-0002-2143-3994
Astana IT University, Kazakhstan

Muslim Sergaziyev
Head of Computing and Data Science department
muslim.sergaziyev@astanait.edu.kz, orcid.org/0009-0004-5332-3441
Astana IT University, Kazakhstan

Daniyar Issenov
Master student ADA educational program, Computing and Data Science Department
231902@astanait.edu.kz, orcid.org/0009-0009-3572-9647
Astana IT University, Kazakhstan

Didar Yedilkhan
Head of the "Smart City" Research center, PhD, Associate Professor
d.yedilkhan@astanait.edu.kz, orcid.org/0000-0002-6343-5277
Astana IT University, Kazakhstan

DEEP NEURAL NETWORK AND CNN MODEL OF DRIVING BEHAVIOR PREDICTION FOR AUTONOMOUS VEHICLES IN SMART CITY

Abstract: This research applies deep neural networks (DNN) and convolutional neural networks (CNN) to the modeling and prediction of driving behavior in autonomous vehicles within the Smart City context. Developed, trained, validated, and tested within the Keras framework, the model is optimized to predict the steering angle for self-driving vehicles in a controlled simulated environment. Utilizing a training dataset comprised of image data paired with steering angles, the model achieves autonomous navigation along a designated track. Key innovations in the model's architecture, including parameter fine-tuning and structural optimization, contribute to its computational efficiency and high responsiveness. The integration of convolutional layers facilitates advanced spatial feature extraction, while the inclusion of repeated layers mitigates information loss, with implications for potential future enhancements. Clustering algorithms, including K-Means, DBSCAN, Gaussian Mixture Model, Mean-Shift, and Hierarchical Clustering, further augment the model by providing insights into driving environment segmentation, obstacle detection, and driving pattern analysis, thereby enhancing complex decision-making capabilities amid real-world noise and uncertainty. Empirical results demonstrate the efficacy of Gaussian Mixture and DBSCAN algorithms in addressing environmental uncertainties, with DBSCAN displaying robust noise tolerance and anomaly detection capabilities. Additionally, the CNN model exhibits superior performance, with lower loss values on both training and validation datasets compared to an RNN model, underscoring CNN's suitability for visually driven tasks within autonomous systems. The study advances the field of autonomous vehicle behavior prediction through a novel integration of neural networks and clustering algorithms to support sophisticated decision-making in autonomous driving. The findings contribute to the development of intelligent systems within the Smart City framework, emphasizing model precision and computational efficiency.

Received: 26.09.2024 Accepted: 27.10.2024 Published: 30.10.2024

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 20, OCTOBER 2024

Keywords: self-driving cars; machine learning; Mean-shift clustering; Udacity car simulator; Gaussian mixture model; K-means clustering; DBSCAN; hierarchical clustering.

Introduction

Self-driving cars have become a trending subject with significant improvement in technologies in the last decade. The purpose of the study is to train a neural network to drive an autonomous car agent on the tracks of Udacity's Car Simulator environment. Udacity has released the simulator as an open-source software and enthusiasts have hosted a competition (challenge) to teach a car how to drive using only camera images and deep learning. Autonomously driving a car requires learning to control the steering angle, throttle, and brakes. In the practice mode on the track, driving behavior is imitated by a behavioral cloning technique. In the simulator, a dataset is produced by a user-driven car in training mode, and the deep neural network model then operates the vehicle autonomously. In the end, the automobile was able to perform admirably on Track 1. The research hopes to eventually achieve the same precision on real-time data.

To simulate a real-world setting, Udacity published an open-source simulator for self-driving cars. The task is to use a model built by deep neural networks to simulate human driving behavior on the simulator. To replicate how a human would drive, the idea is known as behavioral cloning. Two tracks and two modes–training mode and autonomous mode–are included in the simulator. The user creates the dataset while operating the simulator while operating the vehicle in training mode. The «good» driving data is another name for this type of data. The deep learning model is then tested on the track to see how it does after being trained using the user data. The problem is solved through the following steps.

1. Data collection: Use a simulator to drive the car in training mode with a joystick or keyboard, generating a driving log and a set of images.
2. Machine learning model: Develop a machine learning model using Deep Neural Networks in Keras, trained on the collected data.
3. Autonomous driving: Once trained, the model provides steering angles and throttle commands to drive the car autonomously in the simulator.
4. Simulation: The model's outputs are sent back to the simulator to keep the car on track autonomously.

Literary review

The Smart city concept nowadays comes true with self-driven vehicles which will come soon one of the most sustainable parts of Smart cities all over the world. In the Republic of Kazakhstan since 2019 a huge amount of works are devoted to Smart city development processes like Methodical rules for Smart city in RK, National and International standards accredited by Kazakhstan government and many scientific papers related to Smart city development in RK [12] [13] like Almaty development strategy till 2025 in short-term and till 2030 long-term (Aim 6. Smart city) and many projects for Smart city like Smart city Akkol, Ikomek 109 and others [16], [17], [18]. The self-driven vehicles will give opportunities to develop Smart Roads [10], [11] strategy in everyday life and will provide modern complex data mining technologies for smart city urban mobility [19], [20], road safety, traffic and passengers flows predictive models more accurately build for smart city, manage parking in city more effectively and using Greentech with reduce traditional energy resources [1], [2].

SDV could be fully integrated with Smart city infrastructure using multi domain data technologies and ML, AI [21],[22], CNN and RNN algorithms because it has local edge infrastructure computing system to the central cloud. 4G, 5G technologies and 6G in future prospective and their success implementations are used by SDV for sharing the data and collecting with smart city infrastructure. SDV can send speed parameters, location, weather, road conditions back to the cloud, for traffic jam model prediction, with complex analysis for SDV routes optimization in Smart city. SDV car itself could be implemented and modified as the green energy flow node for sustainable smart city development [3], [4], [5]. For example, in Almaty in 2025 it is planned to use aero taxi VTOL vehicles to move between Almaty and Alatau city [13], [14]. For Astana, using models for SDVis also actual and novel task for passenger flow prediction model and for evaluation time of SDV coming due to the driving behavior prediction [15].

There are several studies investigating SDV simulators with the help of various gaming tools [23], [24], [25]. These studies utilize the game engine to simulate various driving environments and scenarios, enabling researchers to test autonomous driving algorithms, assess vehicle responses under different conditions, and gather data for improving SDV functionalities. Through such tools, challenges such as object detection, path planning, and obstacle avoidance, thus contributing to advancements in autonomous vehicle safety and performance can be addressed and solved. For example, in [23] the authors used Epic Gaming's Unreal Engine 4.

Methods and Materials

The project employs various technologies for implementation, each chosen for specific reasons. TensorFlow is an open-source library known for dataflow programming and widely used in machine learning. In this project, TensorFlow serves as both a math library and a platform for large-scale computations. Additionally, Keras, a user-friendly high-level API built on top of TensorFlow, is utilized for better model building. NumPy is a Python library that provides high-level mathematical functions and supports multi-dimensional matrices and arrays. It is used in this project for efficient computations involving neural network weights (gradients). Another significant library, scikit-learn is a Python machine learning library featuring various algorithms and function packages. It enhances the performance of the project by offering diverse machine learning capabilities. Open-Source Computer Vision Library OpenCV is designed for efficient computation, especially for real-time applications. In this project, OpenCV is employed for image preprocessing and augmentation techniques, contributing to improved results. Conda is an open-source Python distribution that simplifies package management and deployment. It is well-suited for large-scale data processing, providing an organized environment for project development. The project is developed on a personal computer, making use of these technologies to facilitate efficient data processing and model training.

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for processing sequences of data. They are particularly well-suited for tasks that involve sequential data, such as natural language processing (NLP), speech recognition, time series analysis, and more [7], [8], [9]. RNNs are used for modeling sequences of data where each element in the sequence depends on previous elements. This makes them suitable for tasks like predicting the next word in a sentence, generating text, or forecasting future values in a time series [6]. CNNs are feed-forward neural networks used for learning from data input. They adjust their weights through training to match desired outputs. They excel at capturing hierarchical and spatial information from images by using filters that scan the input with a defined window size and stride, gradually recognizing details like lines, shapes, and objects. CNNs are well-suited for classifying images into distinct categories. The model predicts the steering angle, θ, for autonomous vehicle navigation based on image data using a CNN. The CNN model follows these stages:

a. *Image Input Representation*

Each input image I is represented as a tensor with dimensions $\mathrm{W} \times \mathrm{H} \times \mathrm{C}$, where W and H are width and height, and C is the number of color channels.

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 20, OCTOBER 2024

b. *Convolutional Layers*

Feature maps F are calculated by applying convolutional filters K across the input image or preceding feature map. For each filter k, the feature map F_k is:

$$F_k = f(I * K_k + b_k) \tag{1}$$

where $*$ denotes convolution, b_k is the bias, and f$(\cdot)$ is a non-linear activation function such as ELU.

c. *Pooling and Flattening*

After pooling layers reduce spatial dimensions, the flattened output is fed into fully connected (dense) layers.

d. *Fully Connected Layers for Regression*

The flattened feature map Flattened is processed through dense layers. The final layer predicts the steering angle θ as follows:

$$\theta = W_{out} \cdot F_{dense} + b_{out} \tag{2}$$

where W_{out} and b_{out} are weights and bias, and F_{dense} is the dense layer output.

e. *Loss Function.* The model minimizes a Mean Squared Error (MSE) loss function.

In addition to the primary neural network model, our project leverages three clustering algorithms to enhance autonomous driving capabilities. K-Means clustering algorithm partitions the data into K clusters by assigning each point to the nearest centroid. It is simple, fast, and efficient for large datasets. However, it assumes that clusters are spherical and equally sized, which may not always be the case in real-world data. For a self-driving car project, it might be useful for segmenting similar traffic patterns or clustering static objects.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm finds core samples of high density and expands clusters from them. It is good at separating high-density clusters from noise and can find clusters of arbitrary shapes. This could be particularly useful in a self-driving context for identifying outliers or 'noise' such as pedestrians or unexpected obstacles.

Hierarchical Clustering creates a tree of clusters called a dendrogram, which can be useful for understanding the data structure at multiple scales. It's good for small to medium datasets but can be computationally expensive for large datasets. In self-driving cars, it could be used for understanding hierarchical relationships in road features.

Mean-Shift Clustering aims to discover blobs in a smooth density of samples without assuming any prior knowledge of the number of clusters. It is not as scalable to high dimensionality as K-means but can handle arbitrary shape clusters. In self-driving cars, it could be used for tracking applications or identifying vehicle groupings.

Gaussian Mixture Model (GMM) Clustering uses soft clustering to assign probabilities to each point for being in a particular cluster, which can be interpreted as cluster memberships. It's more flexible than K-means because it doesn't assume clusters have equal size or spherical shapes. This can be used for probabilistic modeling of traffic situations for self-driving cars.

When comparing these algorithms for a self-driving car project, the choice depends on the specific use case. For instance, if the task is to identify vehicles on the road, a density-based algorithm like DBSCAN might be preferred due to its ability to handle noise. Gaussian Mixture Clustering might offer the nuanced probabilistic approach needed if the task is to categorize different driving environments.

Ultimately, the decision would depend on the specific needs of the project, such as:

- The size and dimensionality of the data.
- The computational resources available.
- The need for speed versus accuracy.
- The type of data and expected clustering structure.

DOI: 10.37943/20MNZJ4935

For instance, if the project requires real-time analysis, a faster algorithm like K-Means might be more suitable. On the other hand, if the project can afford more computational time for a thorough analysis and the data is expected to contain noise or irregular cluster shapes, algorithms like DBSCAN or Gaussian Mixture Clustering might be more appropriate.

The Training Process

Opened a new python3 notebook and git clone the repo, after that imported all the libraries needed for the training process. It will use TensorFlow backend and Keras at the front end. A datadir as the name given to the folder itself and takes the parameters itself.

Using head, showed the first five values for the CSV in the desired format.

As this is picking up the entire path from the local machine, needed the use of the ntpath function to get the network path assigned. Finally, declared a name path_leaf and assigned accordingly.

Data Normalization & Data Information

Binned the number of values where the number will be equal to 25 (odd number aimed to get center distribution). Then, it got the histogram using the np.histogram option on the data frame 'steering', divided it by the number of bins. Kept samples at 400 and then drew a line. At the end the data is centered along the middle which is equal to 0.

Balancing Steering Angle Data for Improved Uniformity

To achieve a more uniform distribution of steering angle data, created a variable called removelist. Used a loop to iterate through each bin, examining the steering data. After shuffling the data to ensure a more uniform structure, certain samples were removed from the remove_list. This process addresses the bias toward driving straight by reducing the significant number of left and right steering angle data points.

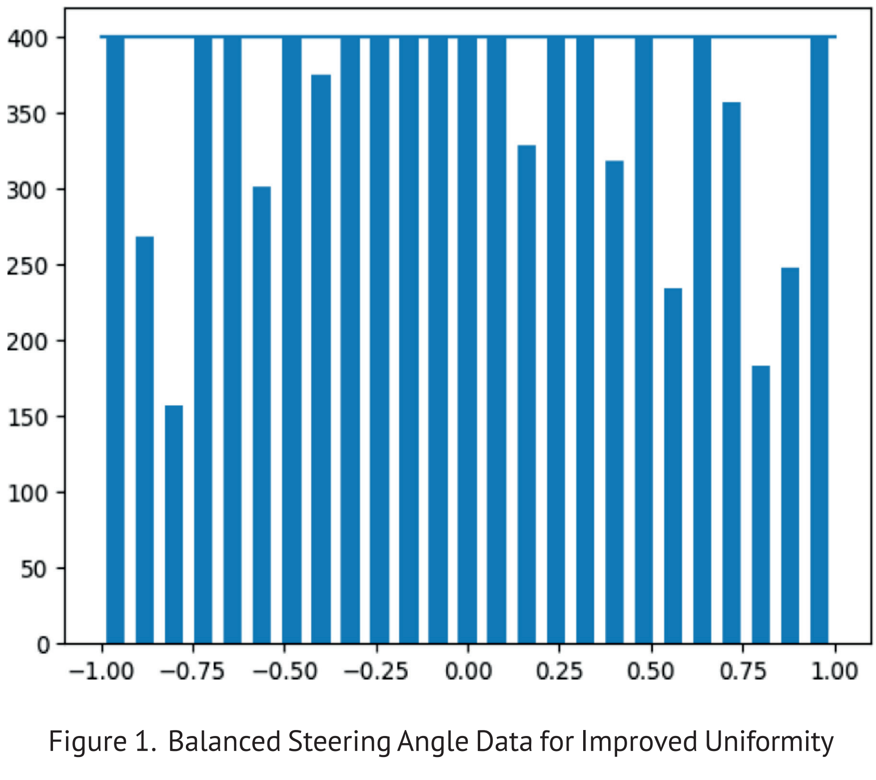

Figure 1. Balanced Steering Angle Data for Improved Uniformity

Loading and Manipulating Images with Steering Data

For this process loaded the image into an array to manipulate them accordingly. Defined a function named locd_img_steering. After, it was got an image path as an empty list and steering as empty list and then loop through. It was used as an iloc selector as a data frame based on the specific index, it will be used to cut data for now.

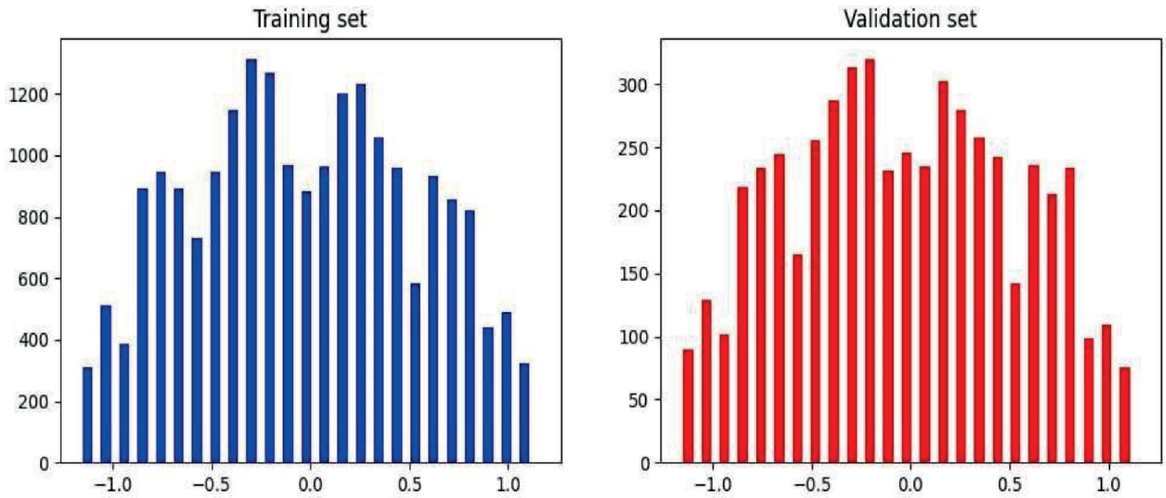

Figure 2. Histograms of training and validation set

Data Augmentation and image pre-processing

Cropped and resized the images from the dataset to focus on the relevant road features. About 30% of the top portion of each image was removed, and this modified image was used in the training set.

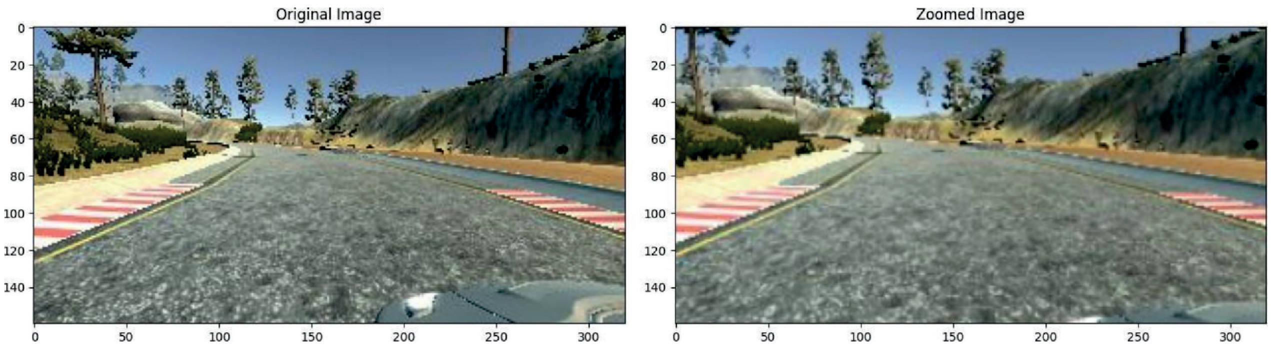

Figure 3. Zoomed version of the original picture after data augmentation

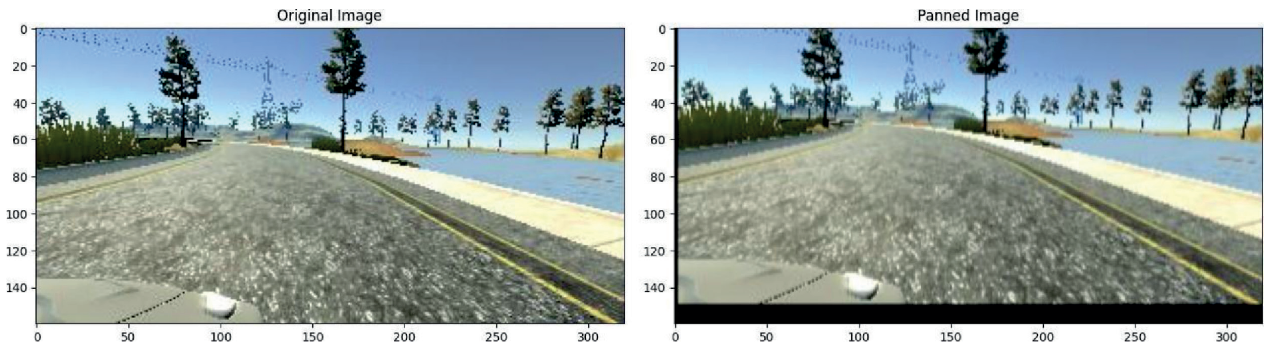

Figure 4. Shifted version of the original picture after data augmentation

To adapt to various weather conditions, such as bright sunny days or low-light situations, in the project brightness augmentation applied.

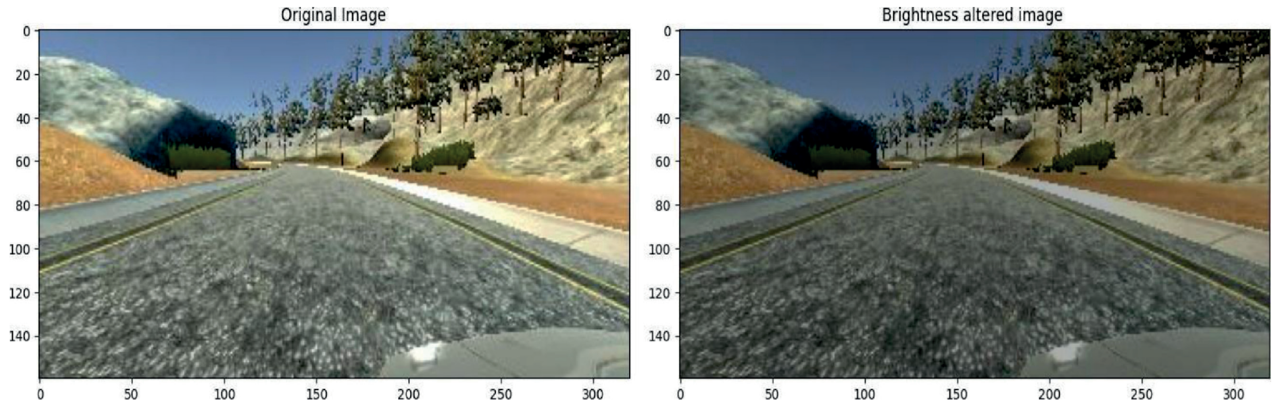

Figure 5. A darkened version of the original image after data augmentation

The horizontally flipped format of the image (created a mirror image) and included it in the dataset to train the model for both left and right turns.

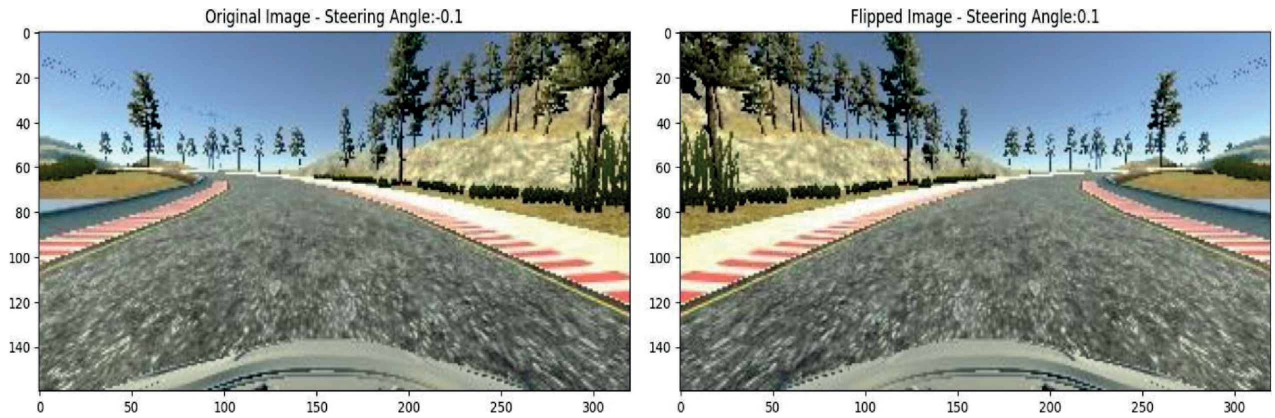

Figure 6. Flipped version of the original image after data augmentation

Model Training and Architecture

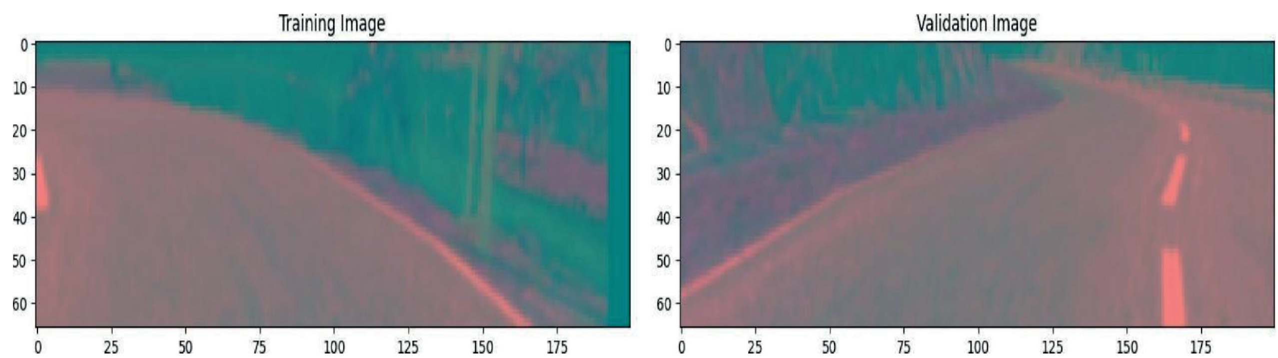

Figure 7. Pre-processed training and validation set images

Experimental configurations

When building the model, researchers tried their best to experiment with parameter settings to achieve the best combination. As a result, we came to the following configuration combination:

- When training the data, we used sequential models built on Keras with deep neural network layers.
- 80% of the data set is used for training, and 20% for testing.
- Epochs = 10, i.e. number of iterations.
  It was also tried with a larger number of epochs, but the model was "overfitted."
- Batch size = 100.
- Learning rate = 0.0001, i.e. how the coefficients of weights or gradients in the network change.

Network architectures

The network design is based on the NVIDIA model, which NVIDIA has used for the end-to-end self-driving test. As such, it is well suited for the project. It is a deep convolutional network that works well with supervised image classification/regression problems. As the NVIDIA model is well documented, work kept focus on how to adjust the training images to produce the best result with some adjustments to the model to avoid overfitting and adding non-linearity to improve the prediction.

In research the following adjustments added to the model:

- Used the Lambda layer to normalize input images to avoid saturation and improve gradients.
- Added an additional dropout layer to avoid overfitting after the convolution layers.
- Included ELU for the activation function for every layer except for the output layer to introduce non-linearity.

In the end, the model looks like as follows:

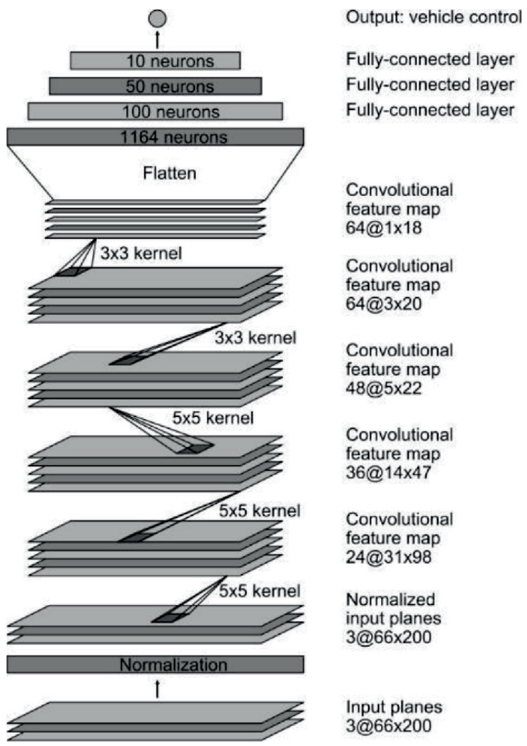

Figure 8. Adjusted NVIDIA model architecture

DOI: 10.37943/20MNZJ4935

As per the NVIDIA model, the convolution layers are meant to handle feature engineering, and the fully connected layer is meant to predict the steering angle. Overall, the model is very functional to clone the given steering behavior. Below is a model structure output from the Keras which gives more details on the shapes and the number of parameters.

Have developed the Model architecture and are moving from Lenet 5 to the NVIDIA Model for traffic sign classification. The behavioral cloning dataset is more complex than MNSIT, with images of size (200.66). 5386 images for training are had, while MNSIT has about 60,000 images. Our behavioral cloning code solves the regression problem by returning the rotation angle. For this, a more advanced NVIDIA model was used.

For model architecture:

1. Definition of the model object.
2. Skip the normalization state since it is already normalized.
3. Add a convolution layer with 24 filters and a 5x5 kernel. Use 2x2 subsampling due to large images. The input form of the model is (66, 200, 3) and the activation function is «elu».
4. The second layer has 36 filters, 5x5 kernel, and 2x2 subsampling, with an «elu» activation function.
5. Three additional convolutional neural network layers with 48, 64, and 64 filters respectively, using 3x3 kernels. Remove subsampling in layers 4 and 5 due to size reduction.
6. Add a flattening layer to convert the output array of the previous convolutional neural network into a one-dimensional array.
7. The last convolution layer with array shape output (1, 18) and 64 filters.

Completed the NVIDIA model architecture with a single dense output layer that predicts the turning angle of a self-driving car. To compile the model, model.compile() is used with the mean square error metric and the Adam optimizer. A low learning rate promotes accuracy. To prevent overfitting of the data, a dropout layer is used that randomly sets the input nodes to 0 during updating, thus generating different combinations of nodes for training. The elimination layer appliesa factor of 0.5 to convert 50% of the input data to 0. The model is defined as the NVIDIA model.

To train the model, model.fit() is used, which passes the training data X_Train and y_train. Given the limited amount of data, more epochs are required for effective training. It also uses data for verification and sets the packet size.

Results

Value loss or Accuracy

The first evaluation parameter considered here is "Loss" over each epoch of the training run. To calculate value loss over each epoch, Keras provides "val_loss", which is the average loss after that epoch. The loss observed during the initial epochs at the beginning of the training phase is high, but it falls gradually, and that is evident by the screenshots below which show the run of Architecture in the training phase.

```
Epoch 1/10
300/300 [==============================] - 384s 1s/step - loss: 0.3379 - val_loss: 0.2234
Epoch 2/10
300/300 [==============================] - 360s 1s/step - loss: 0.2557 - val_loss: 0.2441
Epoch 3/10
300/300 [==============================] - 374s 1s/step - loss: 0.2504 - val_loss: 0.1998
Epoch 4/10
300/300 [==============================] - 361s 1s/step - loss: 0.2417 - val_loss: 0.1837
Epoch 5/10
300/300 [==============================] - 361s 1s/step - loss: 0.2369 - val_loss: 0.1987
Epoch 6/10
300/300 [==============================] - 361s 1s/step - loss: 0.2314 - val_loss: 0.1796
Epoch 7/10
300/300 [==============================] - 357s 1s/step - loss: 0.2248 - val_loss: 0.1896
Epoch 8/10
300/300 [==============================] - 370s 1s/step - loss: 0.2225 - val_loss: 0.1888
Epoch 9/10
300/300 [==============================] - 365s 1s/step - loss: 0.2175 - val_loss: 0.1777
Epoch 10/10
300/300 [==============================] - 365s 1s/step - loss: 0.2136 - val_loss: 0.1704
```

Figure 9. Loss and val_loss evaluation parameters over each epoch of the training run

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 20, OCTOBER 2024

Why do we use ELU over RELU?

Possibility of having a dead RELU – this is when a node in a neural network essentially dies and only feeds a value of zero to nodes that follow it. That is why, it changed from RELU to Elu. Elu function has always a chance to recover and fix its errors, which means it is in the process of learning and contributing to the model. Plotted the model and then save it accordingly in h5 format for a Keras file.

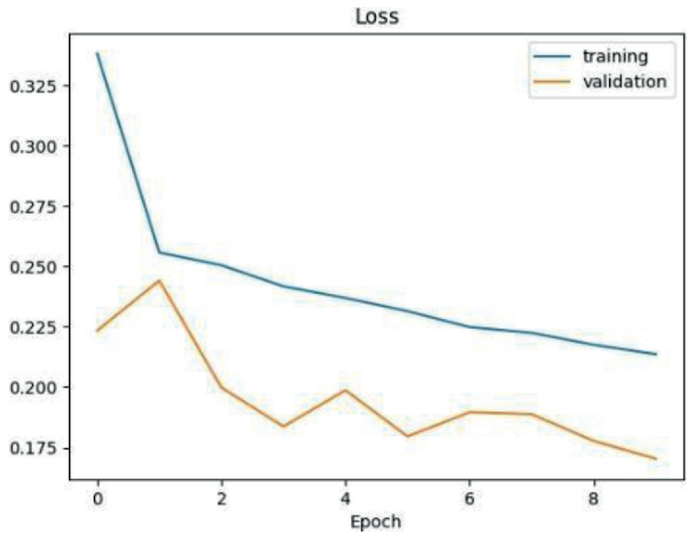

Figure 10. Loss value during model training and validation

Evaluating Clustering Algorithms **[4-8]**

Upon implementing the three clustering algorithms, we observed distinct behaviors:

K-Means showed clear segmentation of road conditions but required prior specification of the number of clusters.

- Clustering Time: **16.08** seconds

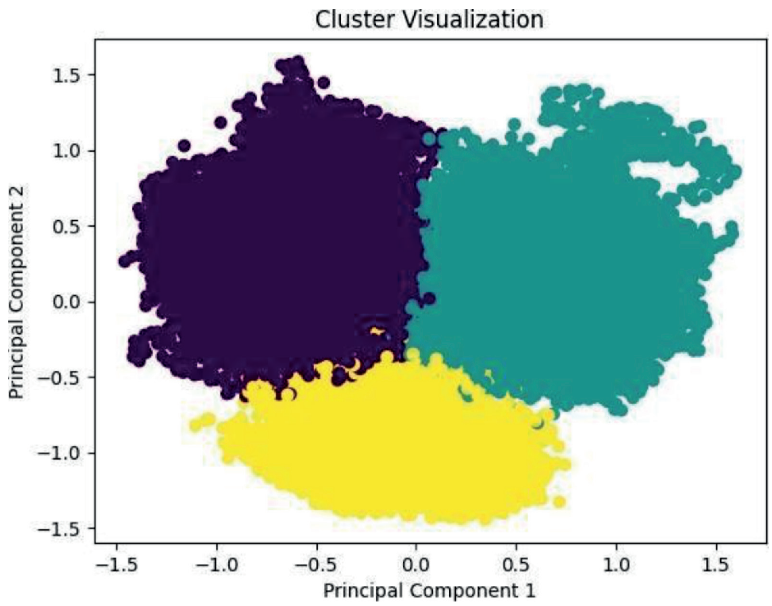

Figure 11. K-Means Clustering Visualization

Similar to Hierarchical Clustering, **K-means** has identified three distinct groups. However, the clusters here are more «spread out» across the steering angle range, which may suggest a more generalized grouping. K-means clustering is sensitive to the initial placement of centroids and can sometimes result in different outcomes on different runs unless the random state is fixed.

DBSCAN excelled in identifying outliers and unusual patterns in driving data, showcasing its utility in detecting unexpected road conditions.

- Clustering Time: **42.32** seconds

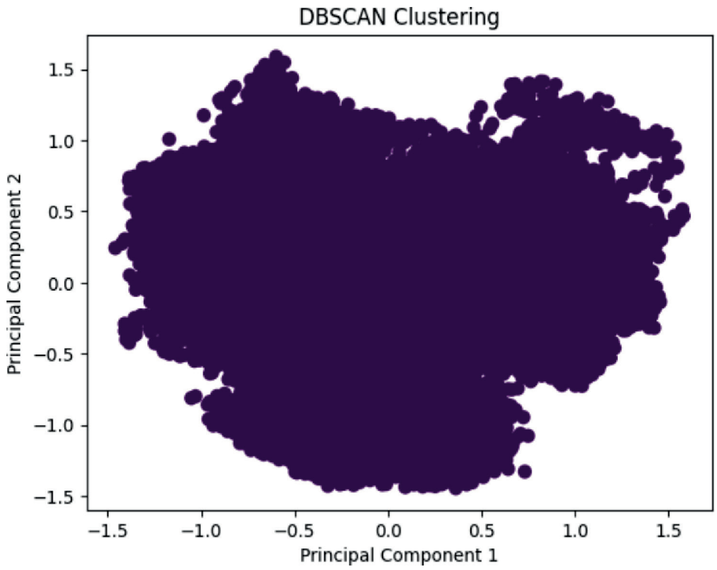

Figure 12. DBSCAN Clustering Visualization

This plot shows that DBSCAN has classified almost each data point as its own cluster or noise. This is indicative of an eps value that is too small, causing the algorithm to fail to group nearby points into the same cluster. There's little to no meaningful clustering here, as DBSCAN is very sensitive to the choice of eps and min_samples. The plot shows data points distributed across two principal components, suggesting a reduction in dimensionality, like PCA. The clustering isn't visually differentiated by color or shape, making it hard to discern distinct clusters or outliers.

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 20, OCTOBER 2024

Hierarchical Clustering provided an insightful dendrogram, illustrating the relationship between different driving behaviors.

- Clustering Time: **0.05** seconds

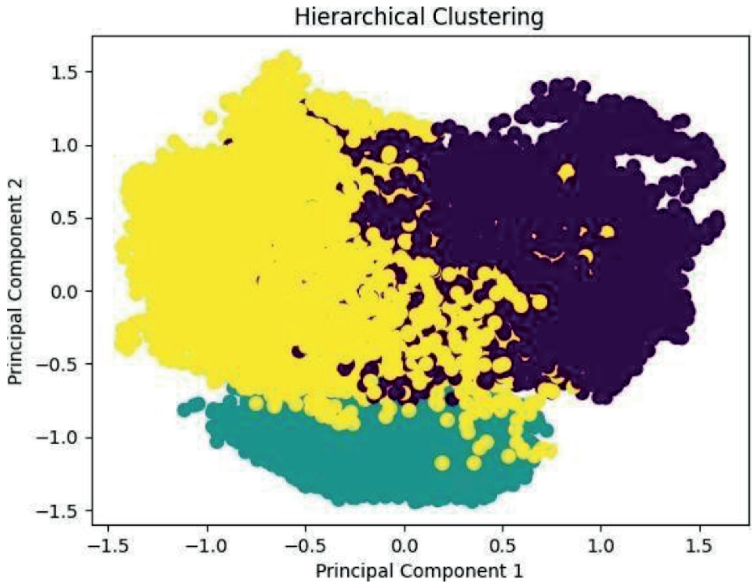

Figure 13. Hierarchical Clustering Visualization

The data seems to be divided into three main clusters with distinct ranges of steering angles. The transitions between clusters are clear and well-defined. This suggests that hierarchical clustering has found a meaningful structure in the data, potentially corresponding to different types of turns or driving patterns.

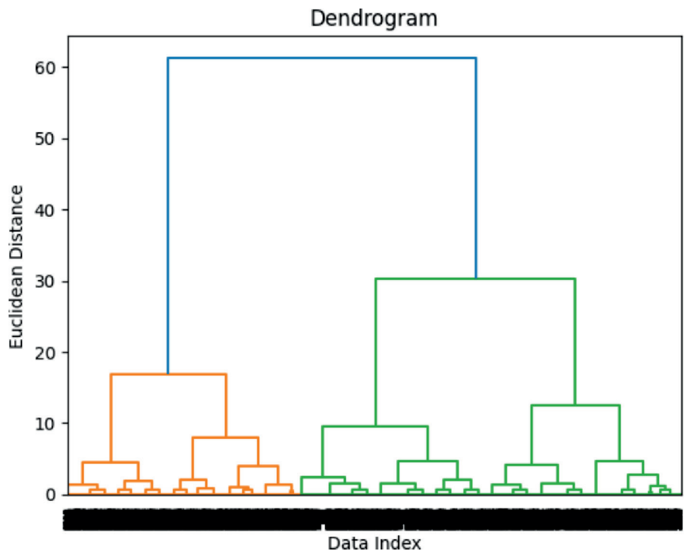

Figure 14. Dendrogram (Hierarchical) Clustering Visualization

The dendrogram provides a visual representation of the clustering process. The height of the dendrograms indicates the distance (or dissimilarity) between clusters. The large vertical distances without horizontal lines cutting through them suggest that a smaller number of clusters could be considered.

Mean-Shift Clustering provided an insightful dendrogram, illustrating the relationship between different driving behaviors.

- Clustering Time: **2902.64** seconds

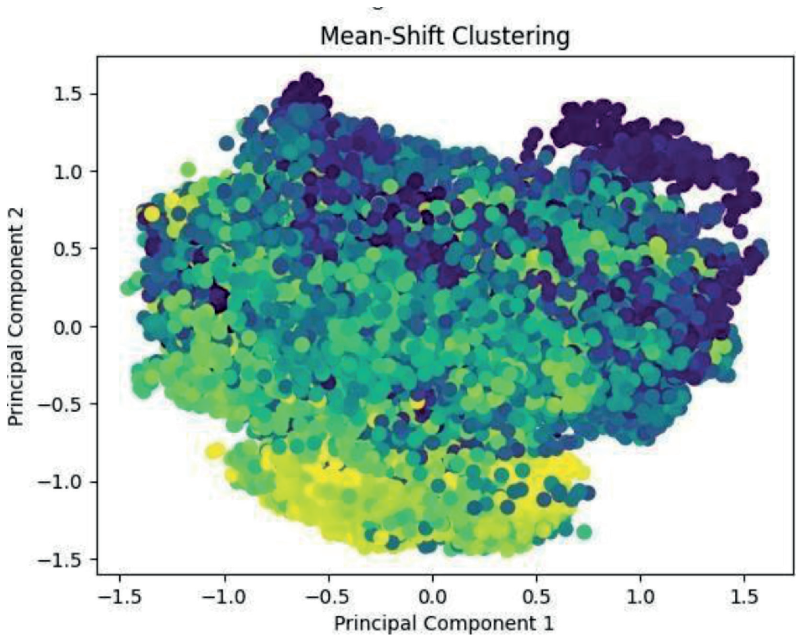

Figure 15. Mean-Shift Clustering Visualization

The image displays a color-coded scatter plot for Mean-Shift Clustering, with data points across two principal components, indicating clusters by color gradients from yellow to dark blue. The plot visually distinguishes clusters but lacks a legend for full clarity.

Gaussian Mixture Model Clustering provided an insightful dendrogram, illustrating the relationship between different driving behaviors.

- Clustering Time: **223.71** seconds

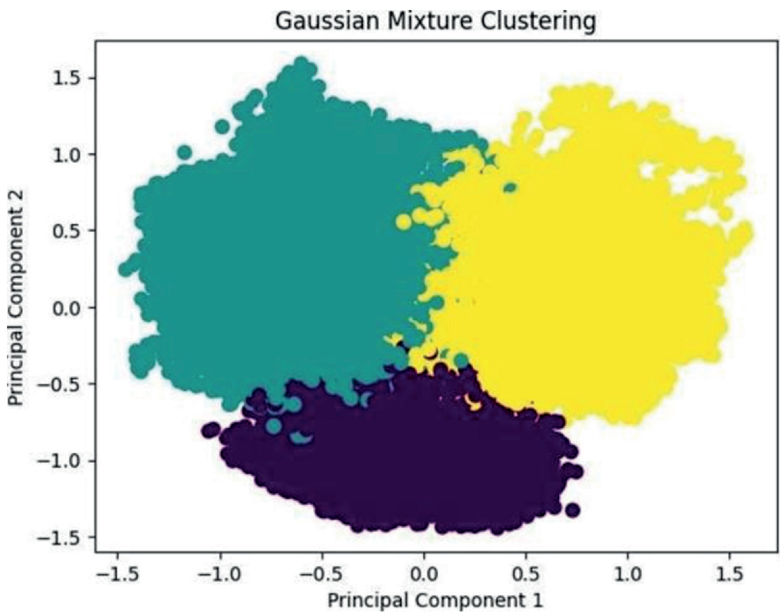

Figure 16. Gaussian Mixture Model Clustering Visualization

Scientific Journal of Astana IT University
ISSN (P): 2707-9031 ISSN (E): 2707-904X
VOLUME 20, OCTOBER 2024

This image shows a Gaussian Mixture Clustering scatter plot with clearly separated clusters in different colors, indicating the grouping of data points by the algorithm. The axes suggest a PCA reduction.

RNN (Recurrent Neural Network):

- Learning Epochs: 10
- Loss on the training set: 0.3992 (on the last epoch)
- Losses on the validation set: 0.3208 (on the last epoch)

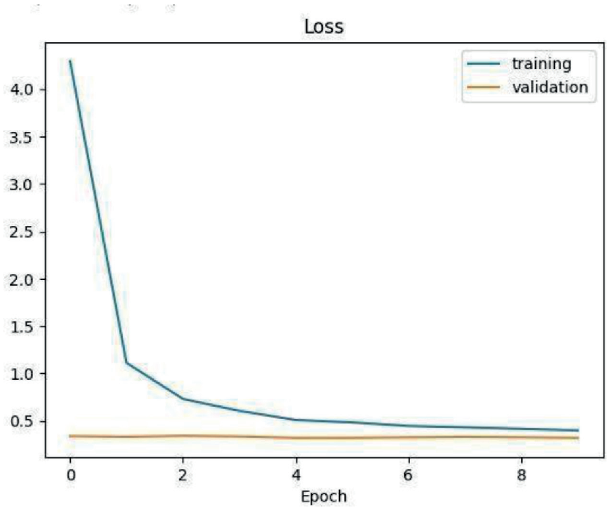

Figure 17. RNN

CNN (Convolutional Neural Network):

- Learning Epochs: 10
- Loss on the training set: 0.2075 (on the last epoch)
- Losses on the validation set: 0.1597 (on the last epoch)

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the lega
Epoch 1/10
210/210 [==============================] - 223s 1s/step - loss: 4.2922 - val_loss: 0.3395
Epoch 2/10
210/210 [==============================] - 204s 973ms/step - loss: 1.1123 - val_loss: 0.3318
Epoch 3/10
210/210 [==============================] - 197s 939ms/step - loss: 0.7317 - val_loss: 0.3423
Epoch 4/10
210/210 [==============================] - 193s 919ms/step - loss: 0.6061 - val_loss: 0.3359
Epoch 5/10
210/210 [==============================] - 205s 976ms/step - loss: 0.5085 - val_loss: 0.3205
Epoch 6/10
210/210 [==============================] - 203s 966ms/step - loss: 0.4836 - val_loss: 0.3209
Epoch 7/10
210/210 [==============================] - 206s 981ms/step - loss: 0.4464 - val_loss: 0.3262
Epoch 8/10
210/210 [==============================] - 211s 1s/step - loss: 0.4318 - val_loss: 0.3312
Epoch 9/10
210/210 [==============================] - 204s 970ms/step - loss: 0.4166 - val_loss: 0.3272
Epoch 10/10
210/210 [==============================] - 207s 988ms/step - loss: 0.3992 - val_loss: 0.3208
```

Figure 18. CNN loss and val_loss evaluation parameters result

Discussion

Analysis of Clustering Algorithms in Autonomous Driving.

The integration of clustering algorithms with neural networks in autonomous driving presents a novel approach. While K-Means offered straightforward segmentations, its requirement for predetermined cluster numbers was a limitation. DBSCAN's ability to detect anomalies aligns well with the unpredictability of real-world driving conditions. Hierarchical Clustering contributed to a deeper understanding of driving data but was computationally intensive for larger datasets.

Comparison of Clustering algorithms

1. K-Means Clustering:

- It has clearly defined, non-overlapping clusters which could represent distinct groups like different types of vehicles, road conditions, or traffic scenarios.
- The sharp boundaries could be used for making clear decisions when distinct options are available.
- The algorithm is fast, as indicated by the relatively short computation time.

2. Hierarchical Clustering:

- It provides a dendrogram which can be useful for understanding the data at multiple levels of granularity.
- This could be used to determine the level of detail required for a decision or to classify objects hierarchically (e.g., vehicle types, then specific models).
- The scatter plot indicates overlapping regions which might represent gradual transitions in scenarios, potentially useful for complex decision-making.

3. DBSCAN:

- It identifies core points and expands clusters from them. This method can identify outliers which might be critical in avoiding unexpected obstacles.
- Its clusters have no fixed shapes, which is beneficial for recognizing a variety of object forms and groupings on the road.

4. Gaussian Mixture Clustering:

- It provides soft clustering with probabilistic cluster assignments, which could be useful for ambiguous situations where decisions are not clear-cut.
- This method may offer a probabilistic understanding of different scenarios, which is valuable for predictive modeling.

5. Mean-Shift Clustering:

- It automatically determines the number of clusters and can find clusters of arbitrary shapes.
- The overlapping and dense regions might be indicative of the flow of traffic or crowded areas.

Comparison of RNN and CNN

1. Loss: Both types of neural networks have achieved low losses on both training and validation datasets. However, CNN has lower losses compared to RNN in the last epoch.

2. Overall Performance: CNN demonstrated better performance compared to RNN based on loss values. This may indicate that CNN is better suited for this task.

3. Training: Both networks have gone through 10 training epochs, which means they had the same number of iterations to train.

4. Validation: CNN also showed better results on the validation set, indicating its ability to generalize data better than RNN.

Conclusion

Considering the specific goal of determining where to turn and by what degree, the best clustering algorithm would likely be one that can handle noise and provide probabilistic outcomes to adapt to the uncertainties inherent in driving scenarios. DBSCAN is excellent for its robustness to outliers, which is essential in dynamic environments. Gaussian Mixture could provide the nuanced probabilities needed for making graded decisions (like degree of turning) in uncertain conditions.

In contrast, K-Means might be too rigid due to its assumption of spherical clusters, and mean-shift could be computationally intensive for real-time applications, as suggested by its long computation time. Hierarchical Clustering is informative but may not be practical for real-time decisions due to its complexity.

For a self-driving car project, a Gaussian Mixture might be the most suitable if the environment contains a lot of uncertainties and probabilistic decisions are required. However, DBSCAN could be a strong candidate for its ability to handle complex and noisy urban environments. The final decision should also consider the computational constraints and the specific context in which the self-driving system operates. Integration with other decision-making systems and sensor inputs would also be crucial for translating these clustering results into steering controls.

Acknowledgement

This research has been funded by the Ministry of Science and Higher Education of the Republic of Kazakhstan, grant number BR24992852 «Intelligent models and methods of Smart City digital ecosystem for sustainable development and the citizens' quality of life improvement».

References

[1] Dilmegani, C. (2023). *Top data augmentation techniques: Ultimate guide for 2023*. AIMultiple.https://research.aimultiple.com/data-augmentation-techniques/

[2] Na, Z. (2019, April 6). *Self-driving car simulator*. Kaggle. https://www.kaggle.com/datasets/zaynena/selfdriving-car-simulator/data

[3] Nawaz, M. (2023, January 20). *Clustering algorithms in machine learning with python*. Python Code. https://thepythoncode.com/article/clustering-algorithms-in-machine-learning-with-python

[4] Pal, M. (2022, September 9). *Deep learning for self-driving cars*. Medium. https://towardsdatascience.com/deep-learning-for-self-driving-cars-7f198ef4cfa2

[5] Saini, A. (2023, October 27). *Guide on Support Vector Machine (SVM) algorithm*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

[6] Zuccolo, R. (2017, April 18). *Self-driving cars – opencv and SVM machine learning with scikit-learn for vehicle detection on the...* Medium. https://medium.com/@ricardo.zuccolo/self-driving-cars-opencv-and-svm-machine-learning- with-scikit-learn-for-vehicle-detection-on-the-bf88860e055a

[7] Askhatuly, A.,Berdysheva, D., Yedilkhan, D., Berdyshev, A. Security Risks of ML Models: Adverserial Machine Learning SIST 2024 – 2024 IEEE 4th International Conference on Smart Information Systems and Technologies, Proceedings, 2024, p. 440–446

[8] Yedilkhan, D., Smakova, S. Machine Learning Approaches for Smart Home Device Recognition from Network Traffic. Procedia Computer Science, 2024, 231, p.709–714.

[9] Amirgaliyev, B., Kuchanskyi, O.,Andrashko, Y.,Yedilkhan, D. A Dynamic Model of Profit Maximization for Carsharing Services: Astana, Republic of Kazakhstan. Urban Science, 2023, 7(3), 74.

[10] Yedilkhan, D., Kyzyrkanov, A.E., Kutpanova, Z.A. ,Aljawarneh, S. ,Atanov, S.K. Intelligent obstacle avoidance algorithm for safe urban monitoring with autonomous mobile drones. Journal of Electronic Science and Technology, 2024, 22(4), 100277.

[11] Alikhanovna, Kuatbayeva Akmaral, Kuatbayeva Gulnar Kuangaliyevna, and Bektemessov Amanzhol Tokhtyamovich. «Logistics cluster in Smart city.» *Scientific Research and Experimental Development* 1 (2022).

[12] Angelidou, Margarita. «Smart city policies: A spatial approach.» Cities 41 (2014): S3-S11.

[13] Methodical recommendations for building smart city in the Republic of Kazakhstan 2022 https://www.gov.kz/memleket/entities/mdai/documents/details/361341?lang=ru
[14] Chen, Zefeng, et al. «Metaverse for smart cities: A surveys.» Internet of Things and Cyber- Physical Systems (2024).
[15] Fadhel, Mohammed A., et al. «Comprehensive systematic review of information fusion methods in smart cities and urban environments.» Information Fusion (2024): 102317.
[16] Prakash, J., et al. «A vehicular network based intelligent transport system for smart cities using machine learning algorithms.» Scientific reports 14.1 (2024): 468.
[17] Naeem, Awad Bin, et al. «Smart road management system for prioritized autonomous vehicles under vehicle-to-everything (V2X) communication.» Multimedia Tools and Applications 83.14 (2024): 41637-41654.
[18] Jain, Pritesh. A Smart Transportation System for Existing Vehicles and Roads Infrastructure to Ease Traffic and Toll Problems in India. No. 2024-26-0181. SAE Technical Paper, 2024.
[19] Hazarika, Anakhi, et al. «Edge ML Technique for Smart Traffic Management in Intelligent Transportation Systems.» IEEE Access (2024).
[20] Elassy, Mohamed, et al. «Intelligent transportation systems for sustainable smart cities.» Transportation Engineering (2024): 100252.
[21] Ullah, Amin, et al. «Smart cities: The role of Internet of Things and machine learning in realizing a data-centric smart environment.» Complex & Intelligent Systems 10.1 (2024): 1607-1637.
[22] Jagatheesaperumal, Senthil Kumar, et al. «Artificial intelligence of things for smart cities: advanced solutions for enhancing transportation safety.» Computational Urban Science 4.1 (2024): 10.
[23] Kaviratna, Neal. «Testing self-driving cars with game development tools.» Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks. (2020).
[24] Zhou, Rui, et al. «Artificial intelligence in engineering education in the case of self-driving vehicle curriculum.» 2022 IEEE 25th international conference on intelligent transportation systems (ITSC). IEEE, 2022.
[25] Priya, M. Deva, et al. «Intelligent navigation system for emergency vehicles.» Proceedings of the 4th International Conference on Smart City Applications. 2019.