ADITYA | SAAD

DriveBerry: Navigating Roads Through Lane and Signage Detection

Introduction

- A **miniature autonomous vehicle system** using **Raspberry Pi** and **Google Edge TPU**, focusing on **lane following** and accurate **signage detection**.
- **Key Components:**
  - **Hardware**: Raspberry Pi, PiCar kit, Google Edge TPU
  - **Software**: Python, OpenCV, TensorFlow, TensorFlow Lite, PyCoral, NumPy
- Systematic approach including hardware assembly, software installation, training and integration.
- Implementation of **real-time lane navigation using OpenCV**.
- Training and deployment of a **Convolutional Neural Network (CNN)** based student network for lane detection.
- **Object detection** for traffic sign recognition using **TensorFlow Lite and Google Edge TPU.**

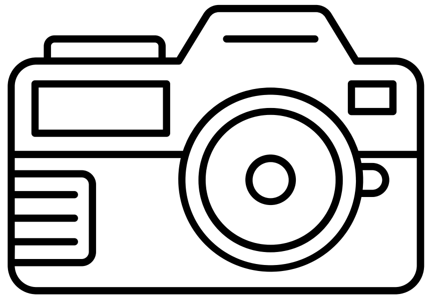
Hardware Design

- **Core Components:**
  - **PiCar Kit**: Foundation of our autonomous vehicle, consisting of a chassis, wheels, motors, Raspberry Pi HAT and servo controls for steering, tilting, and panning the camera.
  - **Raspberry Pi** (4b+): Acts as the **central processing unit**, handling input-output operations, inference, and communication tasks.
  - **Google Edge TPU**: A crucial element for **accelerating deep learning computations**.
  - **Camera Sensor**: A **USB camera attached to the Raspberry Pi**, serving as the vehicle's eyes for capturing video data.
- **Assembled and wired** the Picar kit with steering servo connected to the front wheels and 2 motors connected to the back wheels. Operated using high level PiCar library.

Methodology

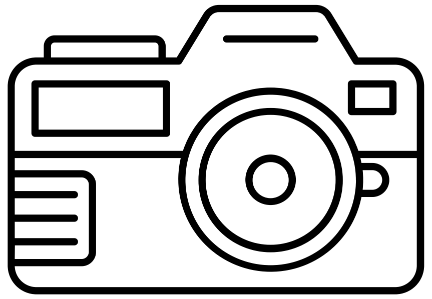
- Lane Navigation using OpenCV
- Lane Navigation using Convolutional Neural Network
- Stop Sign Detection on Edge TPU

Lane **Navigation** Using OpenCV



- **Video Capturing:**
  - Captured **real-time video footage** from a USB camera attached to the Raspberry Pi.
  - Processed video **frame by frame** for lane line detection.
- **Image Processing:**
  - **Conversion** of each frame **from BGR to HSV** color space to handle lighting variability and isolate lane markings.
  - Used **blue tape for lane markings** and **OpenCV's inRange function for color isolation**.
- **Edge Detection:**
  - Utilized **Canny edge detection method** to identify lane line boundaries within each frame.

Lane **Navigation** Using OpenCV

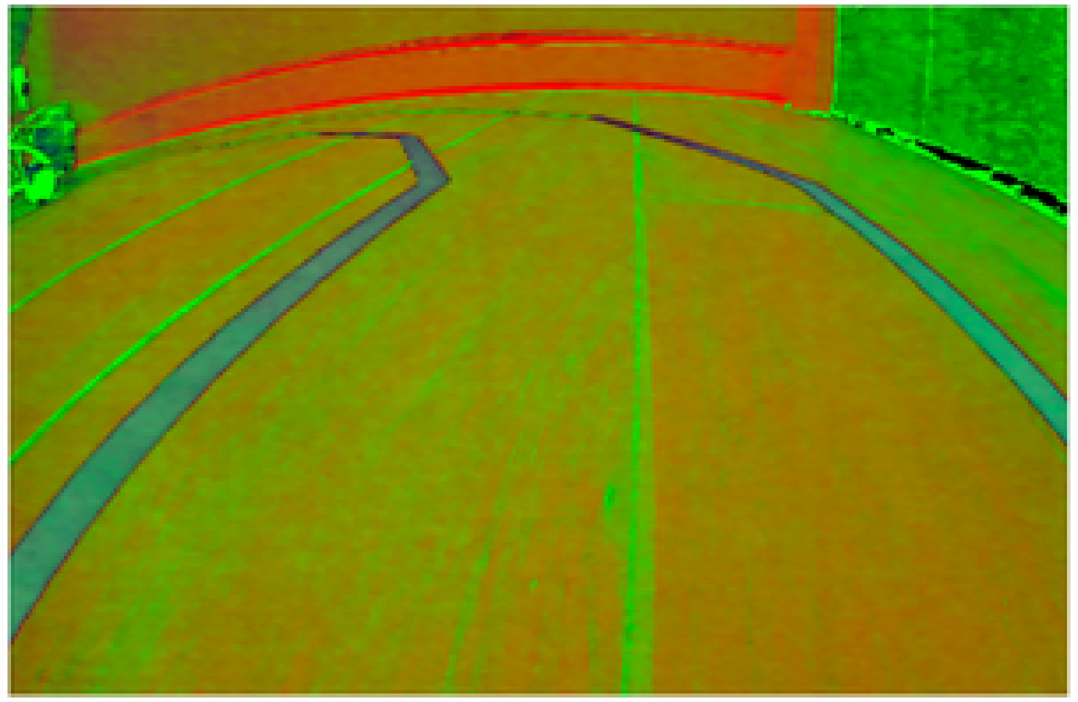


- **Line Detection with Hough Transform:**
    - Converted detected **edges into line representations using Hough Transformation**.
    - This process **interpreted pixel points as potential lane lines** in the image space.
- **Angle Calculation:**
    - Calculated **steering angle** using **arctan of y-offset (half the height of the frame) and x-offset** (midpoint of two lines).
    - Steering angle adjusted against a threshold for smooth navigation and consistent lane-following behavior.
- **Kinematics and Control:**
    - Processed about **20-25 frames per second**, with the car starting at a 90-degree angle and a speed of 35.
    - **Front wheels adjusted based on the calculated steering angle** for seamless lane navigation.

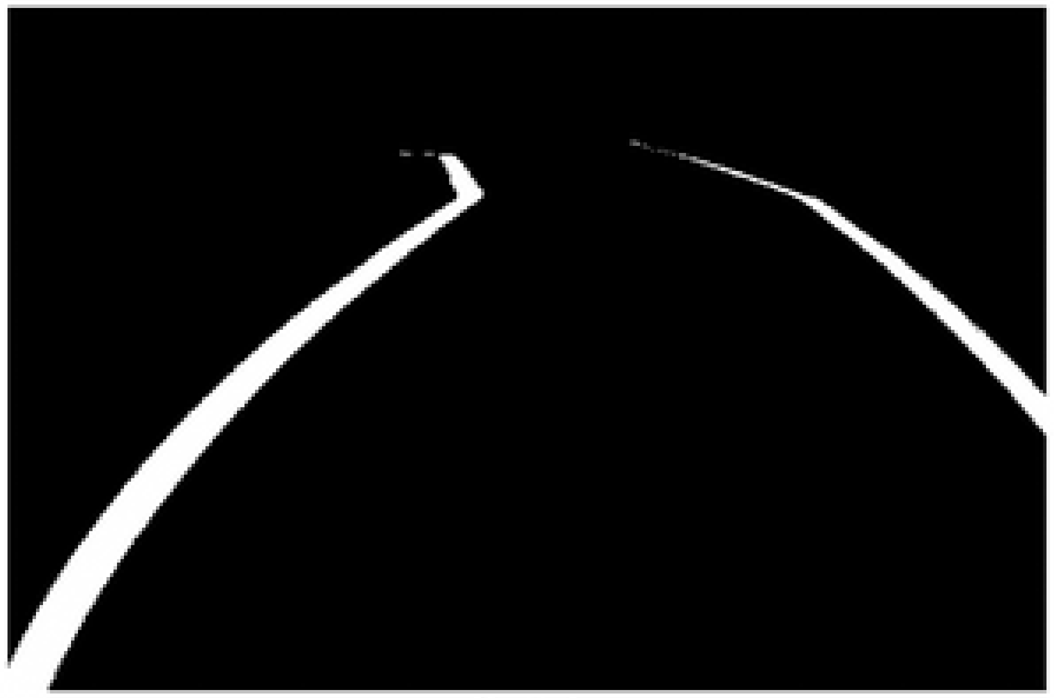
ADITYA | SAAD

Lane Navigation Using OpenCV

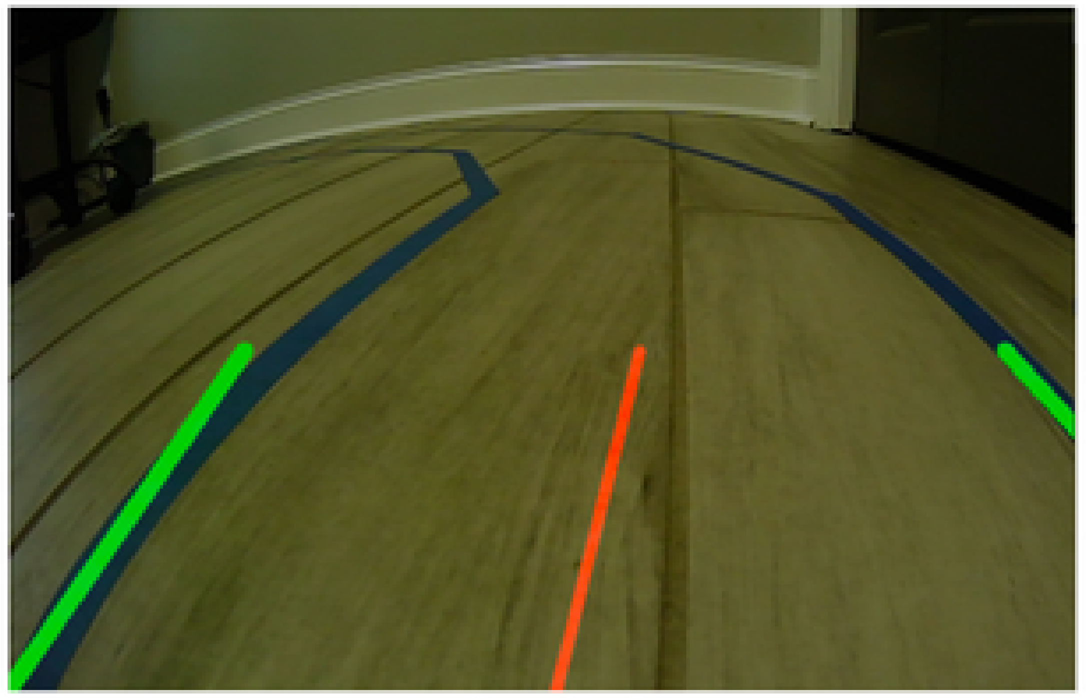
Original Image



HSV Image



Blue Mask



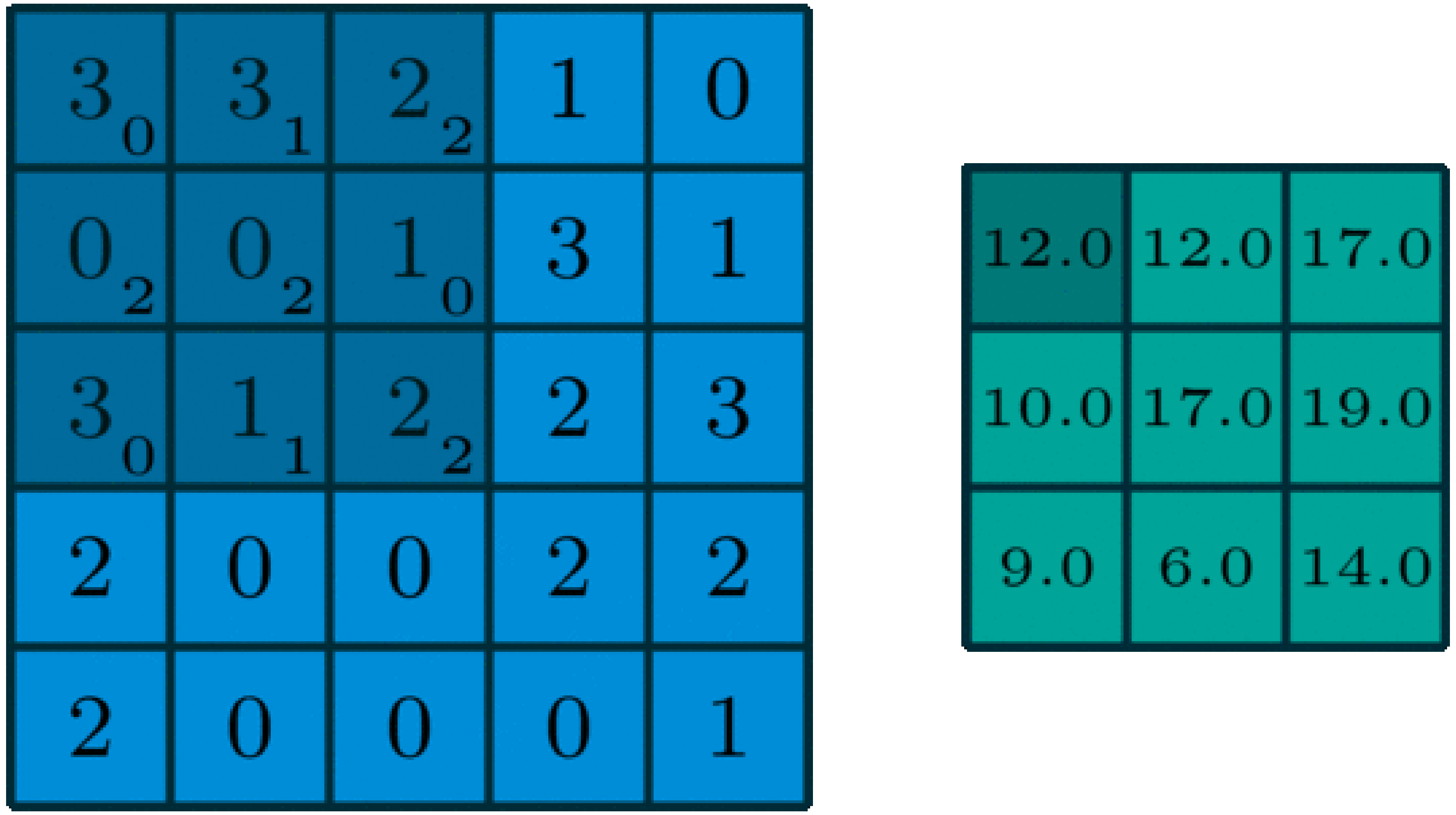
Lane Lines and Heading

Lane **Navigation** Using CNN

- **CNN Architecture (Inspired by NVIDIA):**
  - The model includes **5 convolutional layers** with **Elu activation**.
  - The input size is **66x200x3**.
  - Incorporates a **dropout layer** between the fourth and fifth convolutional layers for **better generalization**.
  - Followed by **4 dense layers**, the last outputting the steering angle.
  - Built and trained using Keras.
- **Data Preparation:**
  - Dataset of **692 distinct frames** captured by the **Raspberry Pi camera** by using the OpenCV based approach.
  - Steering angles predominantly around **80 to 90 degrees**.
  - Image augmentation including zooming, panning, brightness adjustment, blurring, and flipping.
  - Pre-processing to match input dimensions and color requirements of the model.

Lane **Navigation** Using CNN

- **Model Training:**
  - **Regression problem** with steering angle ranging from 0 to 180 degrees.
  - Trained using **Adam optimizer** and **Mean Squared Error (MSE) loss** function.
  - Learning rate set at **1e-3**.
  - Training involved **10 epochs** with a **batch size of 100**.
- **Model Evaluation:**
  - Achieved an **MSE of 8.8** and **R squared of 94.03%** on the test set.
  - Training and validation loss trends displayed on next slide.
- **Integration into DriveBerry:**
  - Real-time processing of camera frames using the **same pre-processing** as in training.
  - **Steering angle computed using the model's prediction**, controlling the vehicle's front wheels.
  - The **CNN-based lane navigation was functional** but **did not surpass the accuracy of the OpenCV-based approach**.



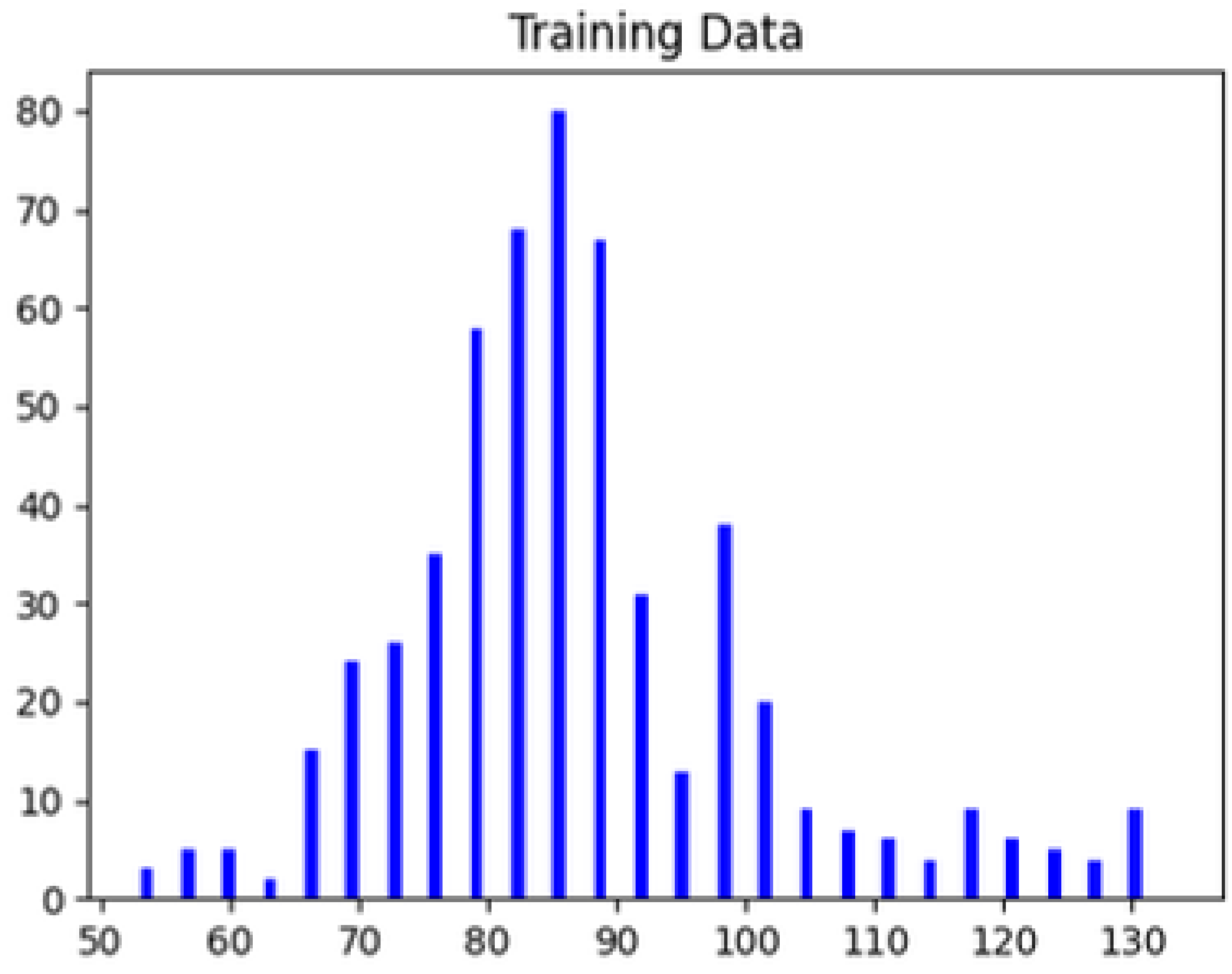
Convolutional Layer

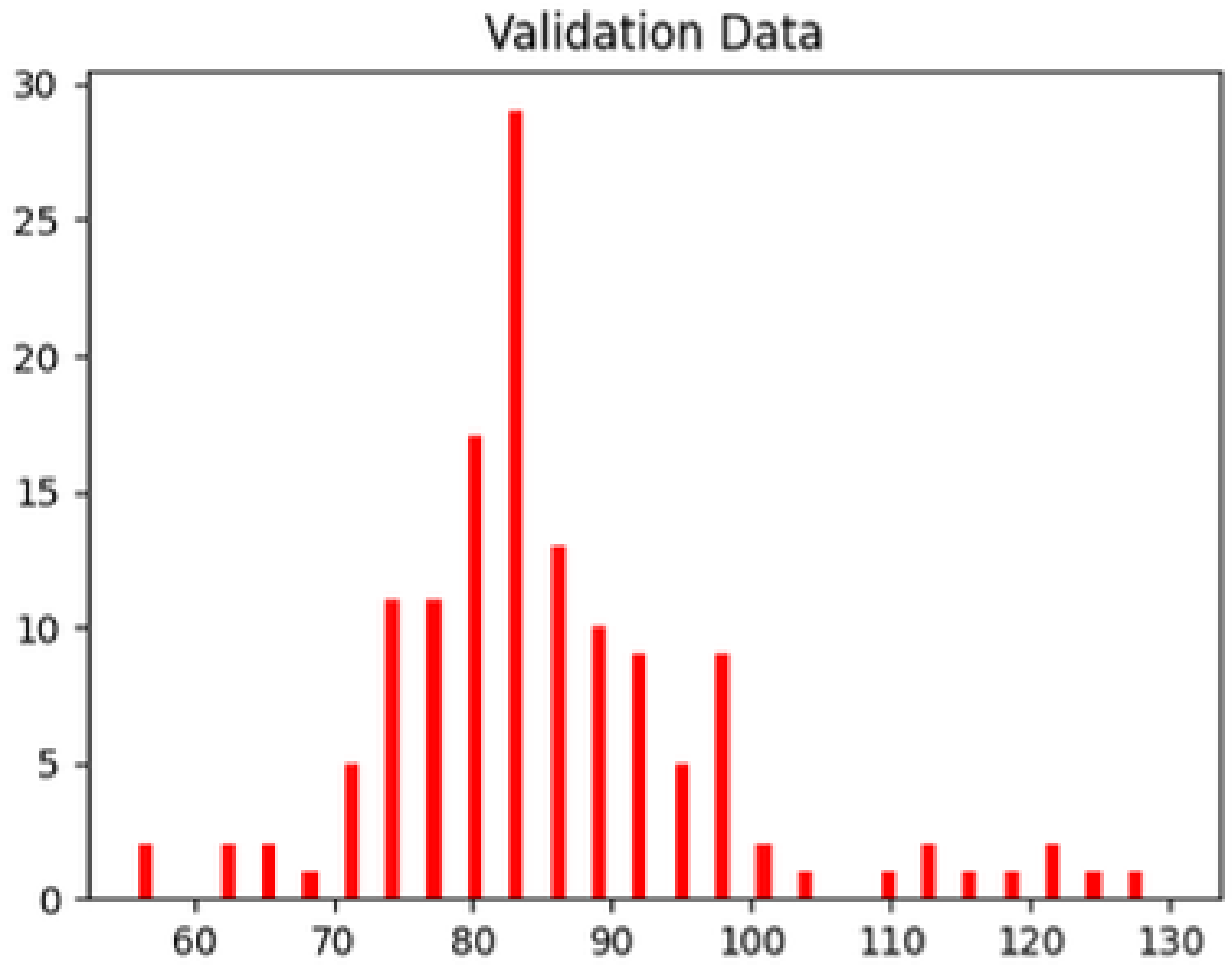
fairly simple operation: Start with a kernel, which is simply a small matrix of weights. **Kernel "slides" over the 2D input data**, performing an **elementwise multiplication** with the part of the input it is currently on, and then **summing up the results** into a single output pixel

Image Source: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

ADITYA | SAAD

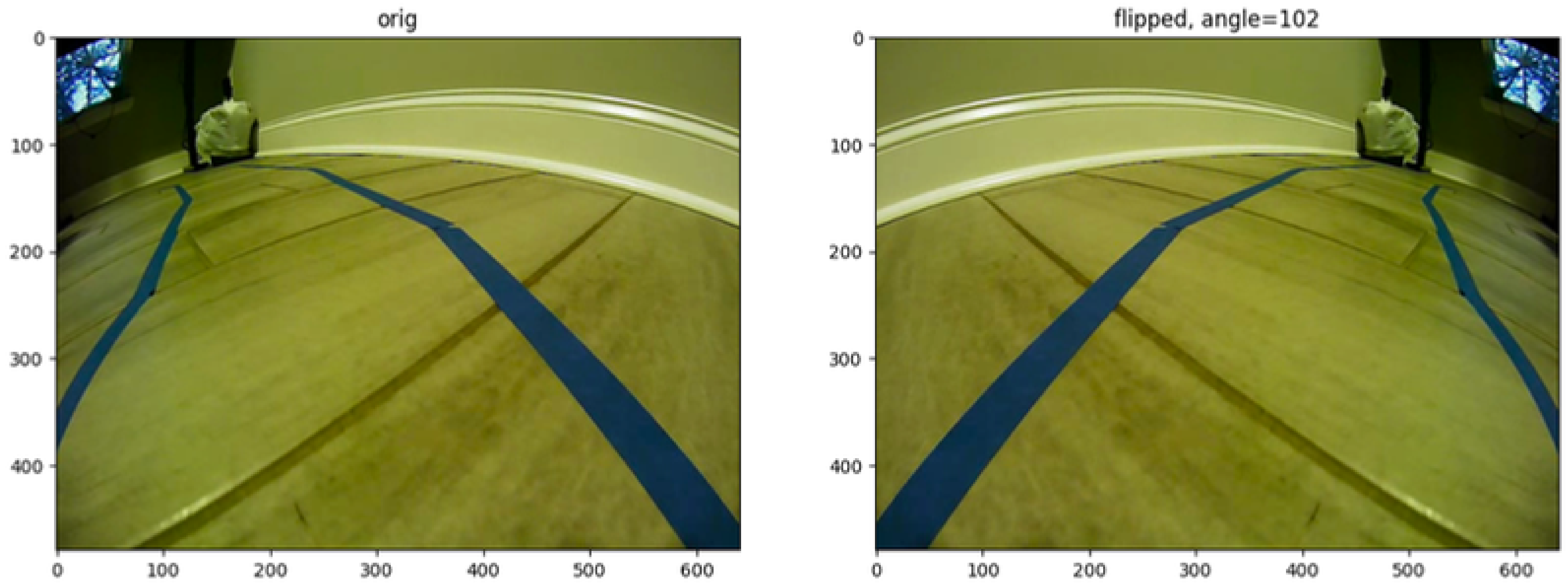
Lane Navigation Using CNN (Angle Distribution)





ADITYA | SAAD

Lane **Navigation** Using CNN (Data Augmentation)

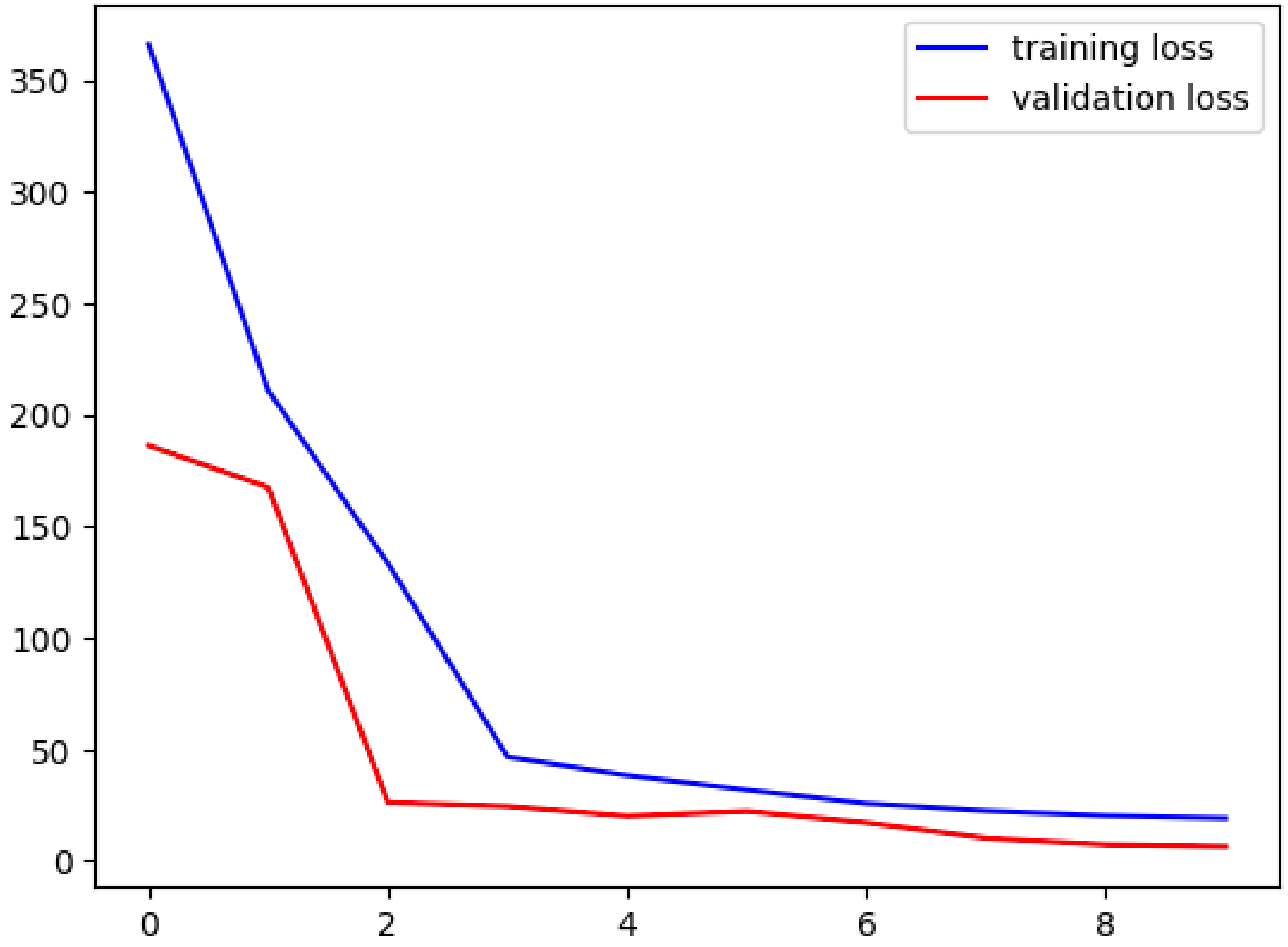


ADITYA | SAAD

Training Outcome

Loss

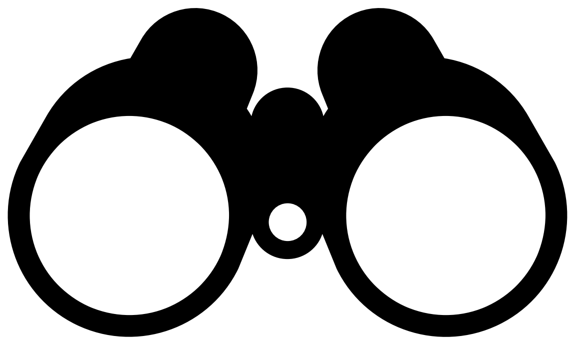
- Training loss and validation **loss significantly went down with each epoch**.
- After Epoch 4, both training and validation loss' rate of decrease **flattened but it still kept going down**, showing no signs of overfitting
- **MSE of 8.8** and **R squared of 94.03%.**



Stop Sign Detection

- Implemented alongside the CNN for lane navigation.
- Used a **pre-trained TensorFlow Lite** (TFLite) model (**SSD MobileNet V2**), optimized for efficiency using **quantization**.
- Deployed using the PyCoral library, **designed for edge devices** like the Google Edge TPU.
- Quantization r**educes model size and speeds up inference** with minimal accuracy loss.
- **Detection Accuracy and Performance:**
  - The **system identified stop signs with a detection accuracy of 100%.**
  - Maintained a **high detection rate throughout different times of the day**.
  - **Consistent frame rate of 30+ FPS**, even after 10+ minutes of continuous operation.

Stop Sign Detection



- **Detection Process:**
  - Load the model on Edge TPU
  - Used PyCoral's **get_objects** method to detect objects and retrieve their bounding box coordinates.
  - **Proximity determination** based on the ratio of the height of the detected object to the height of the frame.
  - A **predefined threshold was used** to decide the immediate relevance of detected objects to the vehicle's path.
- **Response Mechanism:**
  - Upon detecting a stop sign within close proximity, the **vehicle's control system halted the car for three seconds**.
  - This was **achieved by setting the back wheels' speed to zero**, then resuming motion.

Conclusion

Conclusion

- **OpenCV-based lane navigation was more effective** than the CNN approach, adhering to the lane 100% of the time on two different tracks, compared to 83% for the CNN.
- Both methods **showed some performance degradation in low lighting**, with the CNN being more affected.
- OpenCV's superior performance attributed to **meticulous color threshold tuning**.
- Consistent **frame rate of over 30 FPS** even after extended operation of the **object detection model**, demonstrating reliability and speed.
- **Successful integration and application of machine learning and computer vision techniques** in a miniature autonomous vehicle system.

Demo Time!

References

[1] Raja961, “Autonomous Lane-Keeping Car Using Raspberry Pi and OpenCV — instructables.com,” https://www.instructables.com/Autonomous-Lane-Keeping-Car-Using-Raspberry-Pi-and/, [Accessed 04-12-2023].

[2] “Sunfounder picar v kit,” https://docs.sunfounder.com/projects/picar-v/en/latest/, [Accessed 04-12-2023].

[3] “Raspberry Pi Documentation - Getting started — raspberrypi.com,” https://www.raspberrypi.com/documentation/computers/getting-started.html, [Accessed 04-12-2023].

[4] K. Dmitriykovalev, “GitHub - google-coral/pycoral: Python API for ML inferencing and transfer-learning on Coral devices — github.com,” https://github.com/google-coral/pycoral, [Accessed 04-12-2023].

[5] J. Canny, “A computational approach to edge detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[6] L. Chandrasekar and G. Durga, “Implementation of hough transform for image processing applications,” in 2014 International Conference on Communication and Signal Processing, 2014, pp. 843–847.

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” 2016.

[8] “Object Detection with TensorFlow Lite Model Maker,” https://www.tensorflow.org/lite/models/modify/model maker/object detection, [Accessed 03-12-2023].

[9] “Reddit - Dive into anything — reddit.com,” https://www.reddit.com/r/ShinobiCCTV/comments/1761b8n/cant seem to get coral working/, [Accessed 03-12-2023].

[10] “Python 3.10 and 3.11 support?” https://github.com/google-coral/pycoral/issues/85, [Accessed 03-12-2023].

[11] “edgetpu make interpreter fails without printing error,” https://github.com/google-coral/pycoral/issues/57#issuecomment-949997068, [Accessed 03-12-2023].

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2019.

[13] “Models - Object Detection — Coral — coral.ai,” https://coral.ai/models/object-detection/, [Accessed 03-12-2023].