TinyML-Empowered Line Following for a Car Robot

Evaluating the Capabilities of Various Lane Detection Models on Microcontrollers

Adrien Carton[1]

Supervisors: Qing Wang[1]**, Ran Zhu** [1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
27 June 2025

Name of the student: Adrien Carton
Final project course: CSE3000 Research Project
Thesis committee: Qing Wang, Ran Zhu, R.R. Venkatesha Prasad[1]

Abstract

This research explores the feasibility of implementing lane detection on lightweight microcontrollers using a combination of traditional image processing and compact machine learning methods. With the aim of enabling real-time inference under strict hardware constraints, several models were trained and evaluated against a custom image processing pipeline. Each approach was tested for accuracy, speed, and resource usage on the Raspberry Pi Pico 0 microcontroller. While these solutions fall short of cutting-edge accuracy and cannot process as much information as state of the art models, their low cost, minimal power consumption, and real-time performance highlight their potential. These findings suggest that lightweight lane detection is a viable direction for further research in embedded autonomous systems.

1 Introduction

Autonomous cars and robots can assist humans and allow them to drive their focus on more entertaining tasks. Lane following is an essential component that would allow full autonomy to a diverse class of robots. Their applications range from warehouse and industrial robots to delivery systems and self-driving private vehicles. The ability to compute lane following on board with a microcontroller presents many advantages. It allows independence from heavy on-board hardware or cloud-based solutions, hence, decreasing dependence on internet connection and dedicated CPUs and GPUs. TinyML models [1] aim to enable edge computing by reducing the computational needs of a model. This approach would allow for smaller, lighter, cheaper robots. This increases the scalability and independence of robots and autonomous cars.

Many diverse approaches to the problem of lane detection exist and have been studied. Prompting literature reviews [2; 3; 4] that show accuracies of up to 97% are achievable today. The trend divides in two main approaches: Image processing (IP), Machine Learning (ML), but also a combination of the two. However, the common trait of all approaches is the computational intensity, leading to a hardware-to-computation time trade-off. These real-time lane detection algorithms require bulky, expensive hardware, beyond the scope of this project. Another important note is that the cutting-edge technologies are proprietary to individual auto manufacturers to keep a competitive edge. These approaches are inaccessible and therefore not considered in this study.

Given the diversity of approaches and our constraints, a key question arises: *What is the best solution, focusing on efficiency and accuracy, to detect lanes dynamically on a lightweight microcontroller?* This research will focus on finding and adapting existing solutions to the hardware.

The main contributions of this research include the compilation of different academic approaches to the issue of lane finding, the process of selection and creation of algorithms compatible to the selected hardware, and lastly, a methodical comparison of the resulting algorithms.

The paper starts with an overview of the existing research in section 2. Section 3 then outlines the scientific process followed in the duration of the project. The experimentations and their result are outlined in section 4. Section 5 outlines the ethical limitations of this research, followed by a contextualization of the results and concluding remarks in Sections 6 and 7.

2 Related Work on Lane Detection

As mentioned previously, given the strong trend of autonomous vehicles, many computationally intensive solutions exist to the problem of lane finding. We will classify the different types of solutions to understand the underlying mechanics and what can be used in an implementation customized to the hardware. The existing solutions will be sorted in the order of, IP methods, ML methods and combined methods. Understanding the researched solutions will help us conclude what will be further researched in this study.

Image Processing Approaches

In controlled setups, it suffices to have a lane with a different color from the background and apply thresholding. We consider that this solution is too rigid and limiting, therefore not relevant to this research. Nonetheless, it is important to note that it is a practical and simple to use in controlled environments such as factories.

Other IP methods [5] include camera calibration to undistort images, gradient thresholding, Region of Interest (ROI) masking, edge detection and finish with Lane Pixel Detection and Polynomial Fitting. Given the "artisanal" nature of IP processing each implementation can include additional optimization steps such as sliding windows. The description of a complete simple pipeline can be found in [6] and in Figure 1. For the edge detection, several approaches need to be studied, Canny edge filters [7] are the most precise but more computationally expensive. Hodge-Laplacian [8] and Sobel [9] edge filters offer a less precise solution for a much smaller cost. Using only traditional methods it is possible to achieve "detection rate of 96.78 per cent" [6], although we must note the accuracy test exclude examples from nighttime.

`OpenCV` is the most commonly used framework for image processing applications. This is largely due to OpenCV's convenience; it abstracts many algorithmic details, provides optimized implementations, and supports multi-threading to enhance performance. OpenCV remains the dominant choice for C++ and Python implementation despite evidence suggesting that it is not the most efficient solution for all image processing applications [10]. However, OpenCV's computational demands exceed the capabilities of ultra-low-power microcontrollers such as the `Raspberry Pi Pico 0`. As a result, adapting OpenCV's implementations will be necessary to use existing IP approaches on our hardware.

Machine Learning Approaches

Many diffrent approaches of ML have been used to attempt to find lanes efficiently:

- CNN [11] - A lightweight encoder–decoder first produces a binary "lane-edge" mask; a second branch lo-

calises each individual lane via low-dimensional clustering.

- BNN [12] - Binary Neural network, a subset of CNN, simplifies the weights of the neural network to binary values +1 and -1. Much more compact and efficient to calculate at the cost of precision.
- U-Net [13] - U-Nets create "skip connections", caches of the state while encoding that is reused at decoding time. These connections help preserve spatial detail during reconstruction. It derives its name after the "U" shape the process takes. This does mean that at inference time, the model has to simultaneously cache intermediate results, increasing the total necessary flash memory.
- SCNN (Spatial CNN) [14] - can give information from row to row, column to column. SCNNs are designed to capture horizontal and vertical spatial dependencies, providing very efficient results.
- UFDL (Ultra Fast Structure-aware Deep Lane Detection)[15] - treats lane finding as a classification problem. For Y rows the lane edge can be in one of X classes. This can be set up as a last layer after a classical CNN.
- RNN (Reccurent Neural Networks) - use sequential data to make a more precise time-invariant estimation. However, sufferers of the same issue as the U-Net, the need to cache more frames. It also necessitates more context about the vehicle's speed and direction, which is not commonly available.

An approach we have not encountered in our research would be to use previous predictions to accelerate the next prediction, using the labels rather than the input, as in the RNN. Examples of such models would modify the search space using the previously computed inputs. They are interesting to consider as the labels are much smaller than the actual images, but like the RNN, they require additional context about speed and direction.

All of the given approaches require between 3 and 16 GB of RAM in their proposed implementations. Given the limit of the hardware, 264KB of RAM, we would need to prune them to a magnitude of 10^4. Given the resource constraints, it may be more prudent to derive compact models inspired by the original architectures rather than relying on aggressive pruning of existing large-scale networks.

To that goal, we can use the `Tensorflow` and `keras` libraries provided by Google. They can be considered a toolbox that provides users with the components to create very compact and efficient ML models, such as the `MobileNetV1` architecture [16], also developed by Google. These tools allow us to create minimal models that are able to run on microcontrollers.

Combined Approach

It is important to note that the two methods can work hand in hand. To increase the accuracy of a ML model we need to preprocess it. This can include a normal Gaussian blur but could also use edge filters and ROI masks to combine the benefits of traditional IP and ML. In this paper, we will explore the gains and losses of Sobel edge filters as a preprocessing method.

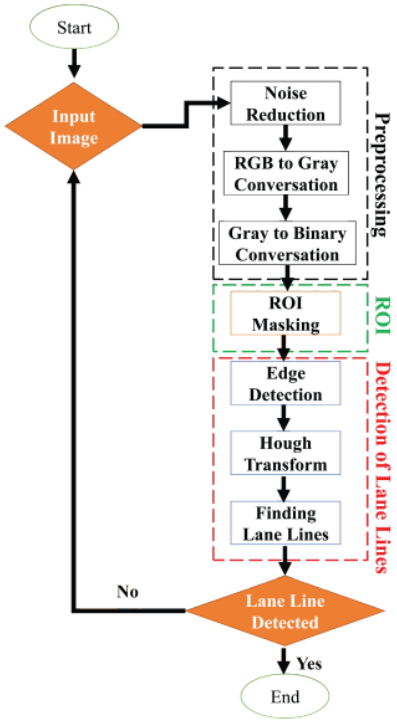


Figure 1: Outline of a simple Image Processing pipeline from [6]

Lastly, it is important to note that more advanced solutions exist. Tesla can create a 360° feature space unmatched in academia due to the resources poured into intellectual property, as detailed in Appendix A. This research will not aim to match the performance of Tesla's, but rather explore a smaller-scale alternative. In the next section, we will further define the constraints set in this research and the steps necessary to complete this goal.

3 Research Constraints and Metodology

This section outlines the constraints and experimental methods used to evaluate lane detection on low-power microcontrollers. With strict limits on memory, processing power, and cost, both image processing and lightweight machine learning models are explored. Section 3.1 briefly defines the research constraints, while the following sections elaborate on the training and evaluation strategies aimed at balancing accuracy and efficiency on constrained microcontrollers.

3.1 Lane Finding on Limited Hardware

The purpose of this research is to bring Lane finding to the edge. In that goal, constraints on power draw, RAM, and cost were prioritized. Three microcontrollers were considered: the Raspberry Pi Pico 0 and 2, and the Teensy 4.1, as can be seen in Table 1. The goal is to create a flexible solution that uses minimal resources. In this study, to explore the full capabilities of lightweight models, we opted to focus on the most constraining microcontroller: the Raspberry Pi Pico 0.

Microcontroller	Cost (€)	SRam (kB)	Flash Memory (MB)	Peak Current Draw (mA)
Raspberry Pi Pico 0	5	264	2	93
Raspberry Pi Pico 2	6	520	4	93
Teensy 4.1	40	1024	8	100

Table 1: Considered Microcontrollers

Having studied IP and ML solutions, this study will focus on comparing different ML models to a traditional IP pipeline, weighing the advantages and disadvantages of each approach. Given that the final goal is to have a solution with accurate estimations, the research will focus on finding and creating solutions, measuring their necessary hardware, and estimating their accuracy and efficiency.

3.2 Experimental Process: ML Models

In this section, we will outline the steps necessary to complete the ML portion of this research. The process is outlined in two main parts; part 1, *Existing ML models*, outlines how existing models are treated. The following parts, *Training Models*, *Datasets and Loss function*, and *Final Preprocessing*, will explicit how new models were trained and created for the research.

Existing ML Models

All existing models are considered an option for the research. The first step is to analyse the resource requirements. They are then filtered into two sections: models that can work on the pico or be pruned enough to, and models intended for use as industry benchmarks. However, since all models were trained to require gigabytes of RAM, none met the criteria for the first category. As a result, the pruning process is not discussed further.

Training Models

Of the Models described in the Section 2, four architectures will be considered: CNN, BNN, U-Net, and UFDL. These models were selected on two criteria: feasibility and probability to work on the pico 0. This discounted RNN architectures and SCNN due to their complexity.

To train and quantize these models, the primary tools will be the Torch and Tensorflow Python libraries. In Tensorflow, the MobileNetV1 architecture from Google will be the primary tool explored.

To achieve the goal of running the models on a microcontroller like the pico 0, the models will have to be as minimal as possible. To do so, the input will have to be reduced. The target hardware camera is the Himax HM01B0, supporting up to 320x320 grayscale images. This means images would occupy 102KB, 39% of the SRAM. To increase the available space for the models, smaller input images of 80x80 and 40x40 will be considered. Downsampling the image to that degree makes the lanes unrecognizable; therefore, image cropping will be considered.

Datasets and Loss function

First, the key to a working model is the dataset it is trained on. In this goal, two datasets were selected. TUSimple [17] is a collection of clips from American highways. It will be used as the simplest training layer, as the lanes are rarely obstructed and the weather is uniformly sunny. To test the models further, the CULane dataset [14] will be used. The dataset was selected due to its diverse urban settings, weather, and lighting conditions. This will test the model's resilience to nighttime, rainy conditions, and roads obstructed by surrounding traffic. For each dataset, a new model will be trained with the same parameters. It is necessary as the camera angles, dimensions, and warps are incompatible, and we were unable to unify them. Larger models can compensate for these flaws, but it is not uncommon to see larger models have separate training configurations per dataset, such as in the implementation of UFLD [15]. In this case, the scale of the models makes retraining unavoidable.

Through experimental trial and error, the dice similarity coefficient [18] was selected as the most efficient loss function:

$$\text{DSC}(p, g) = 1 - \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} g_i}$$

Where N is the number of pixels in the input, p_i is the predicted value for pixel i, and g_i is the ground truth label for pixel i, with 1 indicating the positive class and 0 the negative class.

The dice coefficient was complemented with the focal loss function, which helps underrepresented labels be recognized. The α in the following function will be experimented with to tune the correction:

$$\text{FL}_i = \frac{1}{N}\sum_{i=1}^{N}\left(-\alpha(1-p_i)^{\gamma}\log(p_i)\right)$$

With the same N, i, and p_i. α and γ are constant hyperparameters, γ reducess the loss value for well classified labels while giving a high weight to misclassified examples in the loss result. α offsets the class imbalance by penalizing misclassified minority elements, assigning them a higher weight. This encourages the model to focus more on learning from the underrepresented class.

The number of training epochs is determined by the stabilization of the model's loss; the training will be interrupted once the loss stops decreasing with a patience of 5 epochs if it has 50 epochs scheduled or 10 if 150. UFLDs are scheduled for more epochs due to the model's slower learning rate. The stopping mechanism was implemented to reduce overfitting.

Final Preprocessing

To understand the effect of data preprocessing, each model was trained identically on differently preprocessed databases. The differences include the sampling of the image, one with a simple blur, while the other with top pixel sampling. More

Algorithm 1 Sobel Edge Detection with Thresholding [9]

procedure SOBELEDGEDETECTION(I, T)
  Define Sobel kernels:
    $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
  Initialize output image E with zeros
  for each pixel (i, j) in image I **do**
    Extract 3×3 patch P centered at (i, j)
    $S_x \leftarrow G_x * P$ ▷ Convolve patch with G_x
    $S_y \leftarrow G_y * P$ ▷ Convolve patch with G_y
    $M \leftarrow |S_x| + |S_y|$ ▷ Common Gradient
    if $M \geq T$ **then**
      $E[i, j] \leftarrow 255$ ▷ Mark as edge
    else
      $E[i, j] \leftarrow 0$ ▷ Suppress as non-edge
    end if
  end for
  return E
end procedure

Figure 2: Input 80x80 image before and after Sobel edge filter

importantly, each model will be trained on the raw image and an edge filter. The Sobel edge filter [9] described above in Algorithm 1 is used due to its simplicity and efficiency. The Sobel filter works by emphasizing the edges in an image, which correspond to regions of rapid intensity change. It does this by convolving the image with two 3×3 kernels that approximate the gradient of the image intensity in the horizontal (Gx) and vertical (Gy) directions. The resulting gradients are then combined to produce the final edge map. Edge filters are typically combined with a Gaussian blur to reduce noise in the image, however, given the scale of the input images, too much information would be lost. Moreover, the algorithm used on the microcontroller is slightly modified to include in-place computation, using a buffer to store the current and previous row; it also uses simple thresholding instead of using a binary search square root. A result of the algorithm can be found in Figure 2. We can observe that the contrast between the relevant features and the background is much more pronounced.

For models with smaller input sizes, the same region of interest will be considered. To keep a maximal amount of information in the input, several downsizing methods will be considered. First, a normal Gaussian blur will be considered. For edge detection inputs, the Sobel edge filter will be applied to the downsampled version, but also computed at a higher resolution, then downsampled in several methods to see how they affect the results. They include taking the maximum, average, or sample of the edge map. All models will be trained on all of these input types to verify the effect of the preprocessing methods to find the optimal input format.

Lastly, dual-channel models will be considered throughout the project. Inputting both the original input image and the edge map retains the maximal amount of information. The effects of inputting "more" information will be observed.

3.3 Experimental Process: IP pipelines

Two approaches are considered in this section. First, the manual creation of our pipeline. Second, adapting an existing pipeline to fit on our microcontroller. Both options will be explored. For the first method, the steps in Figure 1 will be followed. Additional steps will be experimented with. These include a sliding window approach, replacing Hough transforms with Centroid detection and gradient direction analysis.

3.4 Evaluation Metrics

To have a global idea of how this research compares to existing research, we will measure the accuracy of our proposed solutions (IP and ML), and existing solutions. The resource consumption of each model will be an important metric. Given that our models will use significantly fewer resources than the existing research, the expectations are to have a lower accuracy but comparable (under 10% difference).

To evaluate and compare the effectiveness of our lane detection approaches, we will use the following evaluation metrics:

- **Accuracy of Lane Detection:** For segmentation models, we will compute the Intersection over Union (IoU) between predicted and ground-truth lane masks. For models predicting lane curvature or steering angle, mean absolute error (MAE) will be used.
- **Frame Rate (FPS):** The models must achieve a minimum of 5 FPS, with 10–20 FPS considered ideal for real-time inference on embedded hardware.
- **Model Size:** The models are measured for the necessary RAM at inference time.

3.5 Measuring and Selecting Adequate Models

Given the goal to run a model on the Pico with an acceptable accuracy to performance trade-off, a compromise will need to be reached. The first step in this direction is to create models and benchmark them systematically against existing models.

Due to the varying amount of information in the images, and the varying nature of the labels, the metrics considered will be False Negative (FN) rate, False Positive (FP) rate, and the lane precision metric. The lane precision metric will be calculated as the difference between the polynomial mean square error of the predictions and the polynomial mean square error of the labels.

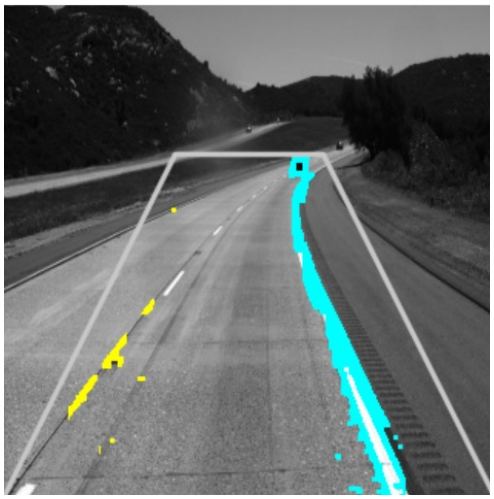


Figure 3: In-house Image Processing Pipeline. Yellow represents the left lane, blue represents the right lane. The gray trapezoid is the ROI, the bounds of the computation.

Each 40-pixel model is tested in four different configurations: "NO SOBEL", "SOBEL BLURRED", "SOBEL SAMPLED", and "SOBEL MAX". The same is done for the 80-pixel models, but only with "NO SOBEL" and "SOBEL".

4 Testing Models

In this section, the created ML models and IP pipelines will be evaluated. As the IP pipelines were unsuccessful and a minor part of the research, they are evaluated first in Section 4.1. The following Sections outline the results of the various trained ML models.

4.1 Image Processing Approach

Although implementations of image processing pipelines to detect lanes exist, none were compatible with the microcontrollers or could be downsized to. An image processing pipeline was created following the steps in Figure 1. Results of the simple pipeline can be seen in Figure 3. Modifications to the pipeline were added to recognize segmented lanes and ignore traffic. However, none succeeded in the project's timeframe. No working hardware-constrained IP pipeline was acquired or created during the duration of the project, but we hold a strong conviction that a working system would not be adaptable in complex environments.

4.2 ML models, TUSimple dataset

Each model was trained on using 80% of the TUSimple dataset [17] as the training data, 10% as the validation set, and 10% for testing the accuracy.

CNN and BNN models

Several CNN architectures were considered during the term of the research, however, the resulting models had a heavy tendency to overfit on straight lines and the horizon, or to always take the "safe guess" of the negative majority class. CNN and the subclass BNN were phased out for U-Net, whose skip connections enabled the models to put a further emphasis on edges. The gains in accuracy and output coherence outweighed the losses of necessitating layer caches.

UFLD: Performance of Preprocessing Methods

Preprocessing 80	FP (%)	FN (%)	Accuracy (%)	Pixel Accuracy
NO_SOBEL	15.1	12.4	83.1	2.737
SOBEL	15.8	9.6	83.9	2.680
DUAL	15.4	12.7	82.6	2.789
Preprocessing 40	**(%)**	**FN (%)**	**Accuracy (%)**	**Pixel Accuracy**
NO_SOBEL	16.1	15.1	80.9	3.741
SOBEL_BLURRED	15.5	17.3	80.4	4.671
SOBEL_SAMPLED	13.4	12.1	84.4	3.461
SOBEL_MAX	14.7	14.7	82.0	3.743
DUAL	14.4	11.0	84.2	3.141

Table 2: Inference result of different preprocessing methods for UFLD, average of 3 models.

Due to the observed variance in model results, attributed to the randomness of the initial weights, the results for each preprocessing type are the average of 3 model trainings. Table 2 shows that larger models with 80x80 inputs perform better, mainly in the pixel accuracy measure.

U-Net: Performance of Preprocessing Methods

Preprocessing 40	FP (%)	FN (%)	Accuracy (%)	Pixel Accuracy
NO_SOBEL	3.8	28.4	95.2	2.6
SOBEL_BLURRED	3.8	29.4	95.2	2.1
SOBEL_SAMPLED	3.8	28.0	95.3	2.3
SOBEL_MAX	3.9	30.9	95.1	2.6

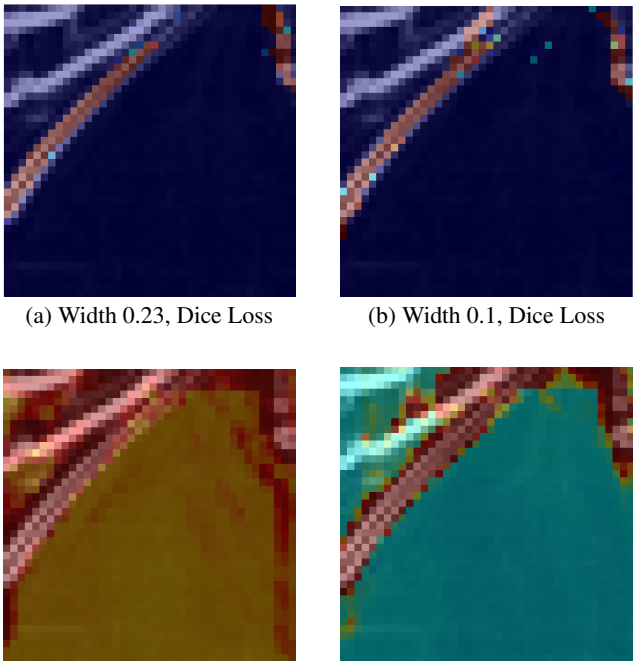
Table 3: Inference result of different preprocessing methods for U-Net trained with Width = .23 and trained with Dice Loss

The results in Table 3 show more promising results than The results for the UFLD, as shown previously in Table 2. Again, the preprocessing methods don't show a qualitative improvement. The high False Negative rate can be attributed to both the model missing some details of lanes, but also to the uniformly thick labeling, which may overestimate the lane width in certain instances. Achieving 95% accuracy despite significantly reducing performance from traditional models is laudable, although it must be noted that only a small comparable section of the input image is considered in this model.

In this Section, only 40x40 models were considered, as only they were small enough for the pico 0 as discussed in Section 4.4.

Introduction of Mini U-Net

Given the promising results of U-Nets at a 40x40 scale, a new architecture of Mini U-Net was introduced with two goals: accelerating predictions and increasing possible input size. Condensing the model reduced the accuracy and the model's ability to comprehend complex situations. In many cases where the lane was disjuncted, the model preferred not to guess. To counteract this, the focal loss function was introduced to encourage the model to guess and understand the underlying structures of the lane. As shown in Figure 4, the focal loss allows the model to overestimate the shapes, creating inaccurate but structurally relevant outputs that capture the essence of the desired lane shape. The performance results of the decrease in size are discussed in Section 4.4.



(a) Width 0.23, Dice Loss

(b) Width 0.1, Dice Loss

(c) Width 0.1, Focal Loss α 0.25

(d) Width 0.1, Focal Loss α 0.1

Figure 4: Visual result of U-Net inference with different training parameters, width, and loss function. The scale is from blue to red, representing the probability of lane presence.

4.3 ML Models on More Complex Datasets: CULane

Results of models trained in the CULane [14] dataset return meaningless results. Without distilling the dataset, input images include blinding reflections, parking tickets covering part of the input, and the hood of the car. Furthermore, the dataset is difficult to recognize by a human once blurred and cropped to a 320x320 size. Although accuracies of 99% were achieved, it was attributed to the lane class representing less than 1% with the model always predicting no lane. The results for this model were discarded.

4.4 On Board Performance of Selected Models

Model Type	Inference time (ms)	Tensor Arena (KB)
UFLD 40	81	21.3
Mini U-Net 40	650	67.5
U-Net 40	721	79.8
UFLD 80	220	40.0
Mini U-Net 80	-	overflow
U-Net 80	-	overflow

Table 4: Inference result of all models on Raspberry Pi Pico 0

Table 4 presents the inference results obtained on the pico 0. The primary metrics of interest are inference time and memory usage, specifically the size of the allocated tensor arena. A clear trade-off is observed between the U-Net and UFLD architectures: while U-Nets achieve higher accuracy, they are significantly more resource-intensive, whereas UFLD models offer faster inference with reduced memory requirements.

Due to the Pico's maximum tensor arena size of 80.0KB, the U-Net width had to be constrained to 0.23, utilizing approximately 99.8% of the available memory for a 40x40 input. Despite this limitation, U-Nets remain computationally demanding and are incapable of real-time performance, achieving inference speeds little above one frame per second on the intended hardware.

In contrast, the UFLD model fulfills its design objective of rapid inference, albeit with reduced accuracy. They allow a rate of up to 12.3 frames processed per second, which can be considered real-time.

The Mini U-Net, intended as a compromise between the two extremes, does not significantly reduce inference time nor enable the processing of 80×80 inputs, which, even quantized, failed to fit within the flash resource constraints of the target. Therefore, Mini U-Net fails to deliver the anticipated balance between accuracy and efficiency.

Adding the preprocessing methods did not change the inference time or the necessary memory for any of the models, as the architecture itself did not change. The overall necessary RAM was not increased as the buffers used by the sobel edge filter were smaller than the ones used by the models and were reused by the models once the image was preprocessed. The preprocessing took an insignificant amount of time, adding 0.9ms for a 40x40 input and 3.7ms for an 80x80 input. The different preprocessing methods can therefore be considered equally on the Pico 0, although all the results for inference time given in Table 4 are the ones without edge filtering.

The inference results on the pico 0 are encouraging. All evaluated models were able to run on the device to varying extents, each offering a distinct balance between accuracy and speed. Depending on the specific application context, both high-accuracy and high-speed models may be considered viable.

5 Responsible Research

5.1 Ethics in Data Collection

To follow the licenses of the used dataset, the preprocessing code and instructions are made available in the repository. Additionally, none of the created models innately store any data; they process the input images and display the results.

All road images are derived from the TuSimple Lane Detection Benchmark Dataset, which is licensed under the Apache License 2.0. Dataset available at: https://github.com/TuSimple/tusimple-benchmark

The content of this paper is reproducible given the code available in a public repository[1]. However, it is important to note that the start weights of the models are random and non-seeded, and therefore present variance. That is why the results are averaged over several models.

5.2 Use of Generative AI

Artificial intelligence, notably ChatGPT, has been used throughout the duration of the project for `LaTeX` figure refor-

[1] https://github.com/pandaandotter/Lane_Detection

matting, code debugging, and code refactoring. The research process, report writing, and created models did not include generative AI in their creation.

6 Discussion of Models in Broader Context

6.1 Interpretation of Results

The aim of the research was to find *What is the best solution, focusing on efficiency and accuracy, to detect lanes dynamically on a lightweight microcontroller?* As the target of this research is the pico 0 microcontroller, the goal was achieved, albeit at the expense of accuracy or speed.

Additionally, to respect the constraints of the hardware, the input datasets were excessively affected. In some models, only 40x40 of the TUSimple's 1280×720 dataset was considered.

However, the research proves that with minimal information and hardware, low-resource ML models can find meaningful information about the car's surrounding lanes.

6.2 Limitations

While the results prove that results can be achieved on limited hardware, several limitations should be acknowledged.

Firstly, the research is based mainly on the TUSimple dataset, which consists of relatively clean highway scenes with minimal visual clutter, allowing the model to learn from a reasonable context and with minimal information loss, even in a low resolution. In new, more noisy contexts that contain a larger degree of close obstacles, the models struggled to make predictions given their limited field of view.

Secondly, given that the lane class was present around 3% of the time in the TUSimple dataset, numerical measures were hard to trust during the development process. Pure numerical accuracy measures often led to suboptimal results, such as overfitting or underfitting. To encourage the models to guess more frequently and accurately, visual results and tweaks led part of the development process of the project.

Finally, while larger models with higher accuracies were trained during the course of this project, they were not explored in depth due to the strict focus on the pico 0. This microcontroller's 264KB RAM cap limited the experimentation space. Early indications suggest that expanding the RAM budget to 1MB could still substantially improve performance. Future work should consider broadening the hardware range to explore these possibilities.

6.3 Broader context

The field of autonomous lane detection has progressed rapidly, with state-of-the-art models often relying on specialized hardware and computationally intensive architectures. These systems, while highly accurate, are largely inaccessible due to their cost, energy requirements, and proprietary nature.

This research reframes the problem by targeting ultra-low-power microcontrollers like the pico 0. Rather than aiming to match industry-leading accuracy, the goal is to demonstrate that lane detection can function under severe resource constraints. By shifting the focus from performance, this project highlights an underexplored but promising space for embedded autonomy.

Most current academic work assumes a first-person (POV) or warped bird's-eye view and builds around large neural networks trained on massive datasets. However, these models dominate the field primarily because lightweight, efficient alternatives have not yet been fully realized. This study suggests that downsized, efficient models could provide a foundation for challenging Tesla's models on an equal bird's eye view perspective.

7 Conclusions and Future Work

This research demonstrates that lane detection is feasible on ultra-low-power microcontrollers, such as the Raspberry Pi Pico 0, using lightweight machine learning models. By adapting and evaluating both classification-based, UFLD, and semantic segmentation models, U-Net, we show that even under strict resource constraints, meaningful lane detection is achievable. Despite reductions in input resolution and computational capacity, the U-Net boasts an accuracy score of 95% on the TUSimple dataset while the less precise UFLD architecture offers fast inference of down to 88ms, enabling it to process 12 frames per second.

TinyML models represent a largely unexplored avenue for lane detection and would benefit from further investigation. Future work should focus on expanding the range of neural network architectures adapted for TinyML deployment. Additionally, implementing larger and more accurate models, operating on higher-resolution inputs, should be explored using more capable microcontrollers, such as those listed in Table 1.

References

[1] P. Warden and D. Situnayake, *Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2020.

[2] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine vision and applications*, vol. 25, no. 3, pp. 727–745, 2014.

[3] G. Kaur and D. Kumar, "Lane detection techniques: A review," *International Journal of Computer Applications*, vol. 112, no. 10, 2015.

[4] J. Tang, S. Li, and P. Liu, "A review of lane detection methods based on deep learning," *Pattern Recognition*, vol. 111, p. 107623, 2021.

[5] V. Devane, G. Sahane, H. Khairmode, and G. Datkhile, "Lane detection techniques using image processing," in *ITM web of conferences*, vol. 40, p. 03011, EDP Sciences, 2021.

[6] M. A. Al Noman, M. F. Rahaman, Z. Li, S. Ray, and C. Wang, "A computer vision-based lane detection approach for an autonomous vehicle using the image hough transformation and the edge features," in *Advances in Information Communication Technology and Computing: Proceedings of AICTC 2022*, pp. 53–66, Springer, 2023.

[7] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[8] W. V. D. Hodge, *The theory and applications of harmonic integrals*. CUP Archive, 1989.

[9] I. Sobel, G. Feldman, *et al.*, "A 3x3 isotropic gradient operator for image processing," *a talk at the Stanford Artificial Project in*, vol. 1968, pp. 271–272, 1968.

[10] I. Aliyu, M. A. Bomoi, and M. Maishanu, "A comparative study of eigenface and fisherface algorithms based on opencv and sci-kit libraries implementations," *International Journal of Information Engineering and Electronic Business*, vol. 12, no. 3, p. 30, 2022.

[11] Z. Wang, W. Ren, and Q. Qiu, "Lanenet: Real-time lane detection networks for autonomous driving," 2018.

[12] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241, Springer, 2015.

[14] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[15] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*, pp. 276–291, Springer, 2020.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] F. Pizzati, M. Allodi, A. Barrera, and F. García, "Lane detection and classification using cascaded cnns," 2019.

[18] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. Jorge Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pp. 240–248, Springer, 2017.

A Auto Manufacturer's models: Tesla

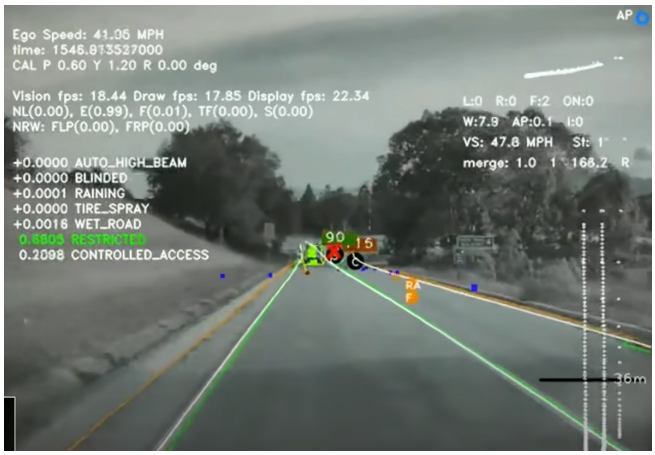


Figure 5: Tesla Old "HydraNet", limited to a point of view prediction

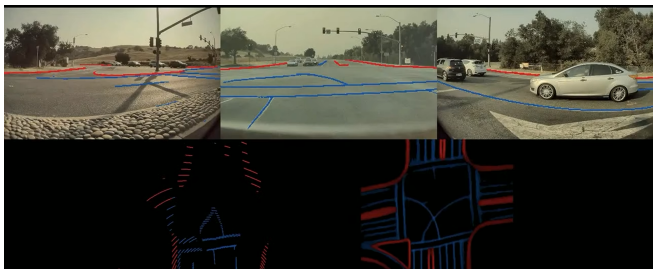


Figure 6: Tesla "HydraNet" left, new "Spatial RNN" featurespace right

This appendix is based on Tesla 2021 AI summit [2]

One subject of this summit is the transition of Tesla to a new feature space. They describe the old "Hydranet" (figure 5), a first-person view feature space that is similar to the feature space current research is exploring. The framework had parallel processes for each task, lane finding, traffic sign recognition, vehicle detection, etc... However, the combined models struggled in new environments and required supervision, retraining, and a lot of manual fine-tuning to deploy.

To make the models more adaptable, the AI team decided to start a new architecture from scratch. Tesla transitioned to an unwarped bird's eye view feature space. With the benefit of an increase in spatial accuracy was paired the adaptability in other tasks, such as vehicle detection.

Being able to map the surroundings accurately allowed Tesla to increase the use of RNNs, allowing past predictions to guide the current spatial awareness. The models spread the prediction by saving a precise map of the discovered environment and only updating new information.

They also go in-depth about creating a record of these feature spaces, creating precise maps of the surroundings by overlapping several passages by one or multiple commuters. They recognize key shapes that are common in several point of views to make a unified prediction.

To conclude, Tesla is an interesting case of auto manufacturers pushing the boundaries of the industry. Given Tesla employs 8 cameras and 16GB of RAM for the combination of lane finding and object recognition of depth and velocity, it is far out of the scope of this project.

[2] https://www.youtube.com/watch?v=j0z4FweCy4M