Robust Lane Detection - Final Report

Charles Downs[1], **Arbër Demi**[1], **Matej Havelka**[1] and **Georgi Trevnenski**[1]

[1]TU Delft

{C.A.Downs, A.Demi, M.Havelka, G.D.Trevnenski-1}@student.tudelft.nl

Check out our code on our GitHub.

1 Introduction

Lane detection has, for a long time, been an important topic in the field of computer vision, and has emerged as an indispensable tool in modern transportation systems (handling tasks such as lane assistance, auto-pilot, and lane-keeping/-assistance). These advances largely hinge on the ability of computer vision systems to accurately and reliably interpret road markings and traffic scenarios, making lane detection a quintessential component in the modernization of transport systems. Progress in these areas of computer vision does not only improve road safety but also traffic fluidity. This progress, however, has proven to be challenging on a variety of levels. While the task of lane recognition and detection is trivial for humans, computer vision systems struggle with aspects such as environmental changes, occlusions, varying infrastructure, or faded lanes. Traditional methods, such as image processing-based pipelines—which we will further discuss and evaluate in section 2—have proven to be efficient under certain circumstances, but they largely struggle with the previously mentioned challenges. Lane detection systems were improved with the advent of the convolutional neural network, but still struggle under certain circumstances, such as occlusions.

The development of self-attention, pioneered by the Transformer model [Vaswani *et al.*, 2023], has burgeoned recently as a relatively new deep learning architecture for sequence-based tasks. Clearly, there is a sequential nature to lane detection tasks, if one assumes a real-time classification task has as input a stream of frames. (Self)-attention mechanisms have proven to be amongst some of the most powerful architectures with sequence-to-sequence tasks, such as natural language processing, translation tasks, and video processing. By treating a stack of images as a sequence, one can tap into the spatiotemporal relationships that often provide valuable insights for lane detection, insights that traditional methods might overlook. As a result, there has been a (small, yet coordinated) effort to produce both the first-ever frame-based traffic scene datasets, as well as the first set of deep learning architectures that exploit the inherent spatiotemporal nature of this vision task. Of particular interest is the `CUlane` dataset [Xingang Pan and Tang, 2018], a collection of over 55 hours of traffic scenes, with a total of 133,235 frames. For each frame, the authors provide an annotated ground truth. This is in contrast to other datasets, such as `TUSimple` [Yoo *et al.*, 2020], which also provide a collection of traffic scene footage, however, the authors only provide one annotated ground truth per clip, making it less performant for a truly recurrent architecture.

The most recent and relevant work that attempts lane detection with a Transformer-based architecture, published in 2023, is the `O2SFormer` [Zhou and Zhou, 2023]. The authors use Transformers for a hybrid approach that performs both one-to-one assignments (one detection is assigned to a single ground truth) as well as one-to-several (a single detection can be matched to multiple ground truths). By adopting a one-to-several strategy, the model can more flexibly and accurately associate detections with their corresponding ground truths. The authors argue this has the dual advantage of resolving ambiguity during the training phase and ensuring that each detection is accurately labeled, thereby potentially improving the model during real-world deployment. Another significant aspect of the `O2SFormer` model is the introduction of dynamic adjustments, such as the layer-wise soft label and the dynamic anchor-based positional query. These innovations address the inherent limitations of DETR's traditional approach, especially in terms of optimizing one-to-one assignment and providing a robust positional context for detections, respectively.

However, while this model makes a significant improvement over previous state-of-the-art models (77.83% F1 score on `CULane`, outperforming existing Transformer-based models), it does have limitations. Specifically, the model does not make full use of the sequential nature inherent in video frames or consecutive images (namely, a several-to-one approach). Exploiting the spatiotemporal relationship between sequential frames can potentially provide additional context and information that can enhance detection accuracy, especially in dynamic environments, where road scenarios change rapidly. We, therefore, posit that these Transformer-based approaches can be further improved upon if they adopt a several-to-one approach, where the spatiotemporal nature of the frames is truly exploited. Throughout this paper, we experiment with how Transformers can be used in a several-to-one fashion using the `CULane` continuous-frame dataset. We investigate multiple stages within the machine learning pipeline. This encompasses enhancements in data preprocessing (optimizing the data format for better model interpretability), a diverse selection of backbones for sophisticated feature extraction, and the adaptability of Transformers in scenarios involving continuous data streams.

2 Literature Review

As with many computer vision tasks, lane detection originates from traditional image and signal processing pipelines [Tang *et al.*, 2021]. Traditional methods would employ a variety of normalization/pre-processing to deal with noise and other artifacts (such as the usage of mean, median [Wang *et al.*, 2010], Gaussian [Hsiao *et al.*, 2009] or FIR filters [Wang *et al.*, 2011]) followed by feature-modeling methods such as edge detection filters (Sobel and Canny [Tang *et al.*, 2021]) as well as more advanced transform-based feature extractors, such as the Hough transform [Tang *et al.*, 2021]. Since the advent of the convolutional neural network, the eponymous AlexNet, and the explosion in the amount of research in deep learning, these novel data-driven approaches have largely outperformed traditional methods, as well as eliminated the need for arduous pre-processing steps such as filtering, instead performing end-to-end learning, where all underlying noise distributions are being learned and dealt with by these deep learning-based methods. More specifically, deep learning-based lane detection models can be classified into the following categories: segmentation-based, classification-based, and regression box-based. In most cases, the latter is the most commonly employed method, where the objective is to parametrize a shape or region by some parameters (e.g., a line can be parametrized by an angle and a magnitude, as a vector). This approach is the least computationally expensive since it does not require performing per-pixel labels.

Before the progress in sequence-based models, the cornerstone operator was the convolution, used inside CNNs in order to progressively learn richer features, allowing for tasks such as classification. However, a lot of the early breakthroughs in lane detection were achieved using CNNs. One of the early and most widely-cited papers that employ a CNN is Line-CNN [Li *et al.*, 2019]. Based on the Faster R-CNN architecture, Line-CNN replaces the *Region Proposal Unit* (RPU), with a *Line Proposal Unit* (LPU), effectively producing a line rather than a region. Line-CNN predicts k line shapes at each location for the boundary (bottom, left, right), where each line has 2 scores for the confidence of being a real traffic lane. The LPU itself is implemented as a Fully Convolutional Network (FCN), where the last convolutional feature map is used in order to extract k line proposals. A 1024-dimensional feature vector is extracted from each boundary position in the feature map. This feature map is then fed into two sibling classification and regression networks, which in turn output a set of k lines, each with output scores (the scores mentioned above) and then a set of $k \times (S + 1)$ coordinates for each line $k_i \in k$. This FCN effectively acts as a feature extractor backbone, the authors here use a ResNet, with 122 layers[1] as their backbone of choice for the feature extraction. In terms of model architecture, Line-CNN is archetypal: it combines a feature extractor, followed by the main model which performs the detection task. In terms of performance, the authors benchmark their model on the MIKKI dataset, and the `TuSimple` dataset. The authors evaluate their method against 3 other methods, of which only 2 employ deep learning. As this paper was one of the first works to employ deep learning for lane detection, it naturally achieved state-of-the-art results, having better accuracy, precision, and recall than the other methods, the details of which we spare from this report.

With this in mind, we can now introduce models that employ a recurrent/sequential structure in their architecture. With the introduction of recurrent sequence-based architectures, we can truly exploit the inherent nature of traffic scenes. Of particular interest is the work from [Zou *et al.*, 2019], with their introduction of ConvLSTM mechanisms, which can capture temporal information—making it fitting for a lane detection task. The overall architecture of this model follows an encoder-decoder structure, with ConvLSTM in the bottleneck. The encoder is used as a feature extractor, similar to in Line-CNN, which then feeds the feature maps into the RNN blocks. The encoder-decoder network then models the lane detection as a semantic segmentation task (as is common with encoder-decoder type networks). The decoder then employs deconvolution and upsampling to highlight the targets and spatially reconstruct them. Both the encoder and decoder are Fully Convolutional (FC). In terms of training, the authors train the network using the pre-trained weights from SegNet and U-Net trained on ImageNet (a relevant aspect of this paper, as we will discuss pre-trained networks for lane detection in the methodology section). The authors employ a cross-entropy loss to treat the task as a discriminative segmentation task. The authors use the `TuSimple` dataset, as well as their own dataset, one noteworthy aspect (particularly in regard to our own model) is that both these datasets only contain 1 ground truth image per video, meaning that the overall accuracy and loss will not always be entirely accurate, this is in contrast to the `CuLane` dataset, which has a labeled ground truth for each frame in a clip. The authors then evaluate their results using precision, recall, and F1-measure. The authors benchmark their model against a variety of encoder-decoder modules, such as SegNet and UNet, as well as publish the results of their model compared to other benchmarks on the **TuSimple** dataset, where their model is second in terms of accuracy (as of March 14th, 2018). Following a similar architecture as above, the authors of [Dong *et al.*, 2023] propose a comparable encoder-decoder model, using what they call a "spatiotemporal RNN", or ST-RNN in order to learn spatiotemporal (ST) dependencies. The authors argue that most existing methods do not take full advantage of the essential properties of the lane being long continuous solid dashed lines, nor do they make use of the ST information correlation and dependencies in the continuous driving frames. The authors treat the detection task as a segmentation task, in contrast to [Zhou and Zhou, 2023], however, the authors use a novel hybrid sequence-to-one deep learning architecture. Additionally, the authors propose an encoder CNN with SCNN [Pan *et al.*, 2018] layers, and then a decoder

[1]The authors argue that with 122 layers, the receptive field of the last convolutional feature map is large enough to cover the whole image, which matches traffic lines

CNN with FC layers. Here, SCNN refers to what is known as a "spatial CNN"—an approach to capture spatial relationships using convolution. To better exploit the spatiotemporal relationships, the authors introduce the aforementioned SCNN layer in the encoder to extract and make use of these relations in a given image frame. The features are then fed into the ST-RNN module, the authors employ both ConvLSTM and ConvGRU to model the time-sequence aspect, similarly to [Zou *et al.*, 2019]. The novelty here is the usage of the SCNN module, which performs a more robust feature extraction by propagating spatial information via slide-by-slide message passing (`UP, DOWN, LEFT, RIGHT`). The authors benchmark their method against a variety of other models, including SCNN, SegNet, UNet, and LaneNet. The authors try a variety of backbones as well as RNN architectures, resulting in 13 different models, ranging from fairly shallow and lightweight to deeper SegNets. Overall, the authors achieve the best performance compared to the other models (using common scores such as F1, precision, and recall).

Having discussed some of the sequence-based models, we can now introduce the concept of attention mechanisms. As discussed in section 1, attention mechanisms have become a very popular choice for sequence or time-series data, with their strong ability to identify relationships in data, making it a good fit for lane detection. This was explored by [Tabelini *et al.*, 2021] in a relatively recent paper. The overall architecture of this approach also employs a CNN for feature extraction, in this case, the authors use a ResNet. Here the authors operate on what they call *anchor points*, which are obtained by pre-processing the lane annotations (the collection of points $\{(x_i, y_i)\}_{i=0}^{N}$) such that they obtain anchors defined by i) an origin $O = (x_{orig}, y_{orig})$, with y_{orig} located on the border of the y-axis, and then ii) a direction θ. In this paper, this comprises a total of 2782 anchors. The feature extraction operates directly on this data, where the feature map corresponds to extracted features from each anchor. These features are extracted through a process the authors call *anchor-based feature pooling*, where a feature vector for a specific anchor is pooled from a feature map. Here, an anchor defines the points of the feature map that will be used for the respective proposals. We note that since these anchors are modeled as lines, the interest points for a given vector are those that intercept the anchors *virtual* line. This is similar to Line-CNN, however, the authors do not fix the y coordinate to be clipped to the bottom border, yielding more profound features. Due to the intricacy of the next part, we propose to introduce some notation: the feature vector extracted from the feature map $\mathbf{F}$ will be named $\mathbf{a}_i$. Due to the nature of the feature pooling (and the inherent nature of CNNs) the information carried by the feature vector is mostly local. To overcome this, the authors introduce attention mechanisms to produce global features, $\mathbf{a}_{\cdot}^{glob}$, which are aggregated with the local features. This attention mechanism is composed of a fully-connected layer, L_{att}, which processes the local features $\mathbf{a}_i$, and outputs a weight (via a softmax operator) for how much attention each local feature should get. This is denoted as $w_{i,j}$ to represent the attention weight from position i to position j. $\mathbf{a}_{\cdot}^{glob}$ is then aggregated with the local features by a weighted summation, as is the case with classical attention mechanisms. A lane proposal is then predicted for each anchor, which is then fed into two FC layers; one for classification (probability of proposal i being a lane), and another for regression, which parametrizes the line. The authors then employ non-maximum impression (NNMS) in order to reduce the number of false positives. In terms of experiments, the authors train and test their model on 2 notable datasets: `CuLane` and `TuSimple`, both of which are relevant to this research project. **TuSimple**: In terms of accuracy, the author's method is on par with other state-of-the-art methods (including SCNN, Line-CNN, and PolyLaneNet). The authors, however, note that these datasets mostly consist of simple cases, where the metrics are rather permissive. The authors, however, note the speed difference with their own method is significantly higher than other methods. **CuLane**: Lastly, when benchmarking against `CuLane`, the authors note that their model achieves the highest F1 amongst all compared methods while keeping a relatively high FPS. The model also performs well considering that this dataset contains more challenging and complex scenes.

Finally, we propose to conclude this review with some of the latest papers that focus on lane detection using Transformers. More specifically, [Zhou and Zhou, 2023] proposed a One-to-Several architecture in 2023. As their premise, the authors note that although some attempts have been made to employ Transformers (more specifically DETR, or DEtection TRansformer [Carion *et al.*, 2020]) for lane detection, two main issues that limit their performance are i) the positional query in DETR lacks a clear focus area, making it difficult to detect lanes, and, ii) the one-to-one label assignment causes low training efficiency due to label semantic conflicts. To overcome each of these concerns, the authors propose the following: i) the authors design a dynamic anchor-based positional query, where lane anchors are encoded as a positional query. This allows the exploration of explicit positional prior to the positional query. Next, they introduce One-to-Several label assignments to solve label semantic conflicts, this works by using one-to-many assignments for the first $N-1$ layers of the decoder, and then one-to-many for the last layer. Lane anchors are represented in a similar fashion to [Li *et al.*, 2019], where x-coordinates have a direct mapping to the y-coordinates, based on a uniform sampling along the edge of the image, and the network output is identical too (length of lane anchors, start and end point, angle, and probability of existing). The setup of the model is archetypal, as it follows the widely-cited DETR model, using a pre-trained ResNet as a backbone. The authors evaluate the model on the `CULane` dataset, using the F1 score as a precision metric. Their model is then evaluated against many state-of-the-art models, including SCNN and LanteATT, discussed above. The authors benchmark their model with ResNet18, ResNet34, and ResNet50, with the expectation that the deeper ResNets achieve higher scores. The authors benchmark their model in a variety of challenging scenarios that are available in the `CULane` dataset[2], where even with ResNet18, the

[2]Normal, Crowded, Dazzle, Shadow, No Line, Arrow, Curve, Night

model outperforms almost all other models in all scenarios, with the ResNet50 backbone achieving the absolute highest F1 score across all scenarios, and against the 8 other models (LSTR, SCNN, UFLD, PINet, CurveLane-L, LaneATT, LaneFormer with ResNet18 and ResNet34 backbone). The authors also examine the convergence curves for their model versus vanilla DETR, noting that their model converges much faster than DETR, illustrating the clear improvement over DETR.

3 Methodology

Having discussed pre-existing research, as well as their methods and results, we would like to discuss the approach and methodology employed in this project. First and foremost, for a better comprehension of how the project evolved based on discoveries and results, we heavily encourage the reader to refer to the methodology section from the mid-term report. For ease of access, it has been added to the appendix, in section A.

The main architectural changes involved switching from an anchor-based approach (analogous to those discussed in section 2) to a segmentation-based approach and replacing the ResNet feature extractor with a model that—following our investigation—seems more fitting. The Transformer-based approach (namely, exploiting self-attention) remains present in our approach. We propose to outline each of these 2 changes in the subsections below, so as to give sufficient detail into our feature extractor, followed by our main model and architecture. These 2 aspects naturally follow each other, but before we start on the changes made, we will dive into the dataset used for this work.

3.1 Dataset

The lane detection dataset that we used for our experimentation is `CULane` [Xingang Pan and Tang, 2018]. As mentioned before, the dataset is comprised of 133235 frames, adding up to 55 hours of video. The frames are of 1640×590 resolution. The dataset is split into 88880 frames for the training set, 9675 for the validation and 34680 for the test set. The test set has its own separation into different categories, each with a separate challenging scenario and making up a different percentage of the test set:

Normal, Crowded, Night, No line, Shadow, Arrow, Dazzle light, Curve, Crossroad.

Examples of each scenario can be seen in the appendix 8. The lanes for each frame are annotated with cubic splines, which are just piecewise cubic functions that interpolate a set of datapoints.

A worthwhile mention is also the fact the annotators chose to annotate lanes in some frames where lanes are not visible, but there is enough context from the video to know that they are there. At the same time, some lanes that are visible are not annotated, such as lanes behind barriers, where entry in the visible area with a vehicle would not be expected.

The main focus of the dataset is four main lane markings which are most used in real-world scenarios, therefore other lane markings are not annotated.

However, this main focus does not seem to always be consistent. There are cases where annotations for lanes are made but there is not really any visual context other than general driving experience that shows that lanes should exist in that particular location. This can be seen in the appendix 10. These cases can be damaging to the training process depending on the pipeline used.

At the same time, there are also examples of lanes being fully visible, however they are annotated only after the car starts moving from a stop, and not before. This makes a lot of frames with lanes present appear as if there are no lanes in the ground truth. Two such cases are presented in appendix 9.

Some practical information about the dataset that is not made immediately explicit by the authors is that the videos provided are not all at the same sampling rate, some being sampled every 30 frames, and some every 90 frames, which might have effects on performance depending on the pipeline being used.

3.2 Feature Extraction

Most research on lane detection, including the studies cited in our literature review, utilizes a pre-trained network in order to perform low-level feature extraction—commonly a ResNet or U-Net. Typically, these feature extractors (herein referred to as backbones) are pre-trained on large datasets such as ImageNet. The ImageNet dataset consists of a large number of images and has been instrumental in advancing computer vision. The wide variety of images featured in this dataset makes it an attractive feature extractor for a variety of downstream tasks, plainly due to the fact that it has the ability to capture the most salient features in many downstream vision tasks. In the case of lane detection, however, it would not be unreasonable to doubt the saliency and descriptiveness of these features. Lanes are typically described as long white lines, any reasonable feature extractor for downstream tasks should be able to capture these markings.

In an effort to investigate the saliency of these features, we set up a variety of experiments in order to help us evaluate feature extractors. Given that the ResNet architectures are some of the most widely-used backbones for lane detection, we trained a ResNet34 in an auto-encoder setup. We import a ResNet34 with ImageNet default weights, and omit the last fully connected layer, as we are not interested in a classification task. The modified ResNet34 acts as the encoder, outputting a feature vector of size `[1, 512, 1, 1]`, corresponding to the `[N, C, H, W]` convention. This effectively gives us a 512-sized vector as per the ResNet34 architecture specification, representing the latent space of the input image. The encoder's task ends with the generation of this latent representation. The subsequent challenge is for the decoder to reconstruct the original image from this compressed form. The success of the decoder in recreating the lanes from the latent representation would demonstrate the sufficiency of the extracted features for the image reconstruction task. This feature vector is then fed into a decoder layer, effectively constituted of a succession of 2D transposed convolutions, and ReLU activations in order to re-create the image. This model was then trained on a subset of the `CULane` dataset over the course of 12 hours, for a total of 100 epochs. These results are illustrated in the appendix, figure 11. These images are both results obtained during the later epochs in the training stage. The lower picture illustrates the best result obtained, where a car can be (somewhat) identified (see evaluation and appendix). In the other case, the most salient features captured are the blue sky and the grey road. Little to no lane can be distinguished, and it is noteworthy that these images are hand-picked from the training set, most likely indicating the validation or real-world reconstructions would not necessarily be better. This can be seen as an indication that ResNet (in our case, ResNet34) is perhaps not the most adequate feature extractor when used pre-trained on ImageNet. Considering the results obtained here are after specific training on a lane detection dataset, it is safe to assume that these types of backbones are sub-optimal for this task. With this in mind, we opted to investigate alternative models in order to create a better feature extractor. We followed the same auto-encoder approach as mentioned above in order to validate the efficacy of the feature vector, where a decoder would be tasked with re-creating the images from the feature vector. Due to the inherent sequential nature of these datasets, self-attention mechanisms may be a better fit, as one would expect if a lane is there, it would be present across multiple frames, despite the environment changing. With this in mind, we decided to train a Vision Transformer (ViT) encoder, once again with the MLP head removed. This approach proved sub-optimal, as the sequential nature of the frames being fed to the ViT would cause the model to simply learn to output a clone of the ground truth it received after multiple frames. A more robust approach would use a masked auto-encoder (MAE) [He *et al.*, 2022], such that the feature extractor is forced to learn a representative latent feature space. The approach here is to load the MAE's pre-trained weights from ImageNet into an autoencoder ViT, which is effectively an autoencoder that employs the ViT architecture for its encoder and decoder. We then train the decoder to reconstruct the image from the latent space, with frozen weights on the encoder–as un-frozen weights were not possible given time and hardware constraints of this project. We use a masking ratio of 0, however, the weights of the imported encoder were trained with a 75% masking ratio, as per the original MAE paper, meaning the added robustness from the masking of the reconstruction is still present in this downstream task, yet we do not withhold any information from our model itself, as missing patches can easily break the continuousness of lanes, thereby perturbing learnable features. A sigmoidal activation is then placed on the output of the decoder in order to produce pixel values in the $[0, 1]$ range. The results are shown in figure 13 and 15 in the appendix, with pairs representing ground truth and output, respectively. It is worthwhile mentioning here that the learned features here are not from `CULane`, in the same way as we did with ResNet34, but rather from the pre-training performed on ImageNet. Better results could have been achieved if the encoder had been trained on the `CULane` dataset with a small masking ratio (e.g. 25%), and no masking for the fine-tuning phase. However, the computational and time requirements would not have fit in the scale of this project, given the time needed to train full MAEs. We believe that this approach may be more successful for lane detection models that rely on feature extractor networks, by training a feature extractor on the `CULane` dataset, the model may learn more representative features for the task at hand, resulting in better performance on downstream tasks. In the state-of-the-art papers reviewed, none had performed pre-training on traffic scene datasets. Results are further discussed in the evaluation section.

3.3 Main Pipeline and Model

Data processing

Before we could start training our model on a segmentation task, we had to preprocess the ground truth CULane provides. The dataset's annotations consist of a list of lanes per image, which are a collection of (x, y) coordinates of points forming a lane. To turn such representation into a segmentation mask, we start with a blank image of the same size as the input and draw short straight lines between every two consecutive points in a lane for all present lanes. We set a thickness of 5 pixels on these lines which proved to be working well in our experiments. This image we turn into a binary segmentation mask.

Furthermore, we performed data augmentation following the example of the O2SFormer. We make use of random horizontal flips and random rotations with an angle of rotation varying in a range of 30 degrees. Additionally, we crop the images at random locations and resize them to 224 by 224 pixels before they enter our model's pipeline. A step-by-step visualization of the augmentation process can be seen in Figure 20 in the appendix.

Architecture

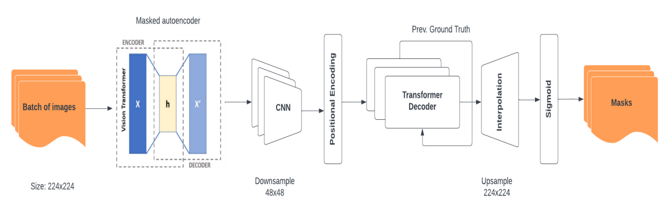

Figure 1: Pipeline of our model (Partially adapted from [Hinton and Salakhutdinov, 2006]). For more detailed explanation of the pre-trained MAE used please refer to [He *et al.*, 2022]

The architecture of our lane detection algorithm consists of two major components - a Masked Autoencoder (MAE) ([He *et al.*, 2022]) and a decoder layer. As Figure 1 displays, between these two components, we utilize a standard convolutional neural network to downsample the images so they can fit on the GPU we used to train. The CNN consists of three convolutional layers with average pooling layers in between. The downsampled images are then positionally encoded before they are fed to the transformer decoder. The positional encoding used is a traditional one with sine and cosine functions, suggested by the original Transformer model ([Vaswani *et al.*, 2023]).

The MAE is used in our architecture as a feature extractor and for this purpose, we load the entire model, pre-trained on the ImageNet dataset. This autoencoder works by dividing the input image into patches (squares of a specific size) and a percentage of those are masked - set to a pixel value of 1. The unmasked patches are run through an encoder and the encoded patches are assembled together with mask tokens in a feature map with the size of the original image. A decoder then tries to reconstruct the original image given the encoded features. Important to note here is that masking is used only in the pertaining of the MAE and we do not mask the images in our pipeline. The way we incorporate the already trained MAE in our architecture is by freezing its encoder's weights and training only its decoder in our backward pass. Therefore, the decoder is expected to learn to output a feature map optimized for the segmentation task we have.

The decoder we use after the MAE is a standard PyTorch implementation without any LayerNorms of a transformer decoder with one decoder layer. The encoded feature maps corresponding to the images in the batch are fed into it one by one. The ground truth of the previous image is passed as a target sequence to the decoder during training, except for the first image where we input a tensor of zeroes. This is the conventional way of training transformers for NLP tasks and even though not recommended in segmentation tasks. In the end we decided to follow this, as to help the network to learn from previous frames. We also tested an approach where we stop feeding it the ground truth once the validation loss stops falling, which gives promising results. During evaluation, of course, the output of the decoder from the previous iteration in the batch is used instead of the ground truth.

Finally, the output of the transformer decoder is upsampled to the original size of the images through bilinear interpolation and a sigmoid function is applied to it. The end result per image is a segmentation mask with a probability for every pixel whether it belongs to a lane or not.

3.4 Loss

Throughout the time we have worked on our pipeline, we have experimented with many different losses in attempts to rectify the issues we were faced with. We will mention them here, however, most will be brief as they were used at a time when our model was not fully functional for reasons unrelated to the loss.

IoU Loss

The main loss we used during training for our final version of the pipeline is a self-made IoU loss. It is quite simple, however, it has proven to be effective. The loss is formed by components in the numerator and denominator, up and $down$, where

$$up = \sum_{dim=1,2} prediction * target$$

with prediction being the result of the model and target being the ground truth. Whereas

$$down = \sum_{dim=1,2} 0.5 * prediction + 0.5 * target$$

This makes the IoU differentiable, while still keeping the overall expected shape of the function. The main reason for picking this loss is that it is ensured that the global minimum is only at the point where the prediction perfectly matches the ground truth.
The end result is

$$IOU_{loss} = 1 - \frac{1}{N} * \sum \frac{up + \epsilon}{down + \epsilon}$$

where $\epsilon = 0.000001$ to address some frames having no lanes, which would lead to division by zero.

Cross Entropy Loss

Another loss that we worked a lot with was the PyTorch implementation of the CrossEntropyLoss, which can be found in appendix D.1. We tried a weighted approach as we had quite an unbalanced training set, with many cases being annotated as lanes, and few without. Although it was promising, most of the experimentation was during the faulty stage of our pipeline, so we cannot make conclusions on its performance for our case.

We also tried the Binary CE loss from PyTorch which also showed promising results, however, it also was in the early stages of our pipeline.

4 Evaluation

In order to evaluate results, we propose the same structure as the methodology; where the section is split into the feature extractor evaluation, and then the main model backbone itself. One important aspect to note is that, given time and hardware constraints, the authors were unable to train the model until convergence. All training was done on the author's own personal hardware. Given the size of the dataset, and the time required for convergence, a realistic comparison and a fully-converged model are not realistic. We therefore propose to evaluate the results obtained *so far* in an *a priori* manner. Training is particularly cumbersome due to the MAE.

4.1 Feature Extractor

An appropriate evaluation of the MAE ViT feature extractor would be to benchmark it against some of the other state-of-the-art methods discussed in the literature review. Unfortunately, due to time and hardware constraints, this was not possible. All training was performed on the author's own, personal hardware. On the whole, the auto-encoder setup used in order to train the MAE ViT seemed to show promising results on image re-creation, with lanes being identifiable in almost all cases. The only comparison we can do is to visually inspect how ResNet34 performed in a similar setup, obviously, we have not tested the feature extractor on downstream tasks. Visual comparisons of reconstructed images can be seen in figures 11 and 13, where the ResNet34 output seems to only be able to re-create road and sky, whereas the proposed model seems to re-create the entire image.

4.2 Main model

To conclude, we will discuss the results obtained on the main model and architecture. As discussed above, the feature extractor backbone is using the ImageNet pre-trained weights. Once again, due to time and hardware constraints, we were not able to train the model long enough to get it to converge, as can be seen in the loss graphs below.

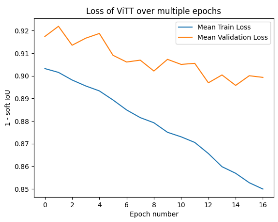

Figure 2: Loss over 16 epochs, training enabled (ground truth being fed back into decoder, see 1.

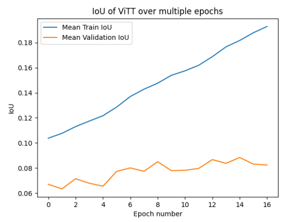

Figure 3: IoU over 16 epochs, training enabled (ground truth being fed back into decoder, see 1.

As illustrated with the IoU and loss on the test set, the model is clearly not converging. We intertwined the running of training and validation in order to evaluate the model at each epoch. The training time for 16 epochs was roughly 30 hours, a significant amount of time considering that we have been advised that models only start learning after 10 epochs[3]. On these types of tasks, it is not uncommon to train for over 100 epochs, which is simply not viable given the aforementioned time and hardware constraints. Below (Figures 4,5,6,7) are some illustrations of the output with training set to **off** (the ground truth is no longer fed to the forward pass, instead model uses its own predictions) after 4 epochs. Green is ground truth, red is where IoU matches, and blue is output from the model.

Clearly, the model is learning to output lanes, as demonstrated by the array of frames outputted, straight lanes seem to be somewhat easy to capture, but the model still seems to output lanes where there are none (Figure 6) and struggles with curved lanes (Figure 7), though the segmentation mask does seem to follow the general direction. This is further illustrated by the mean IoU which is continually decreasing across frames and has not yet converged. These results do, however, suffice to demonstrate this approach and method as a proof of concept for this architecture. We have adopted a mixed ViT/Transformer Decoder approach, combining a novel type of feature extractor, as well as the Transformer decoder using previous frames to truly exploit the sequential nature of the problem.

Finally, an important limitation that we faced due to the hardware constraints is the high batch size required (32), on the relatively large images used. As discussed in the methodology section, our architecture had to employ a convolutional neural network (CNN) in order to downsample the images such that they could fit on our hardware (see appendix for a full list of hardware specifications and hyperparameters). This has the effect that the up and down-sampling from 48×48 to 224×224 means that a single pixel goes into a roughly 5×5 grid, where lanes have a thickness of 5. This results in an inherent limitation in how good of an approximation the network can get, but unfortunately was an unavoidable step, considering 1) hardware limitations, 2) the ViT is pre-trained on images of size 224×224.

In Table 1 you can find all the average metrics on the different models we have trained until now. These are ran on

[3] As noted by this project supervisor

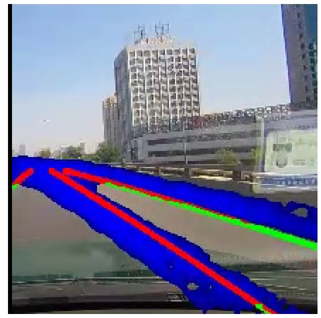

Figure 4: Loss over 16 epochs, without ground truth fed into decoder

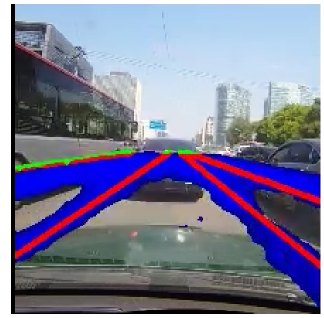

Figure 5: IoU over 16 epochs, without ground truth fed into decoder

Figure 6: Loss over 16 epochs, without ground truth fed into decoder

Figure 7: Loss over 16 epochs, without ground truth fed into decoder

Threshold	IoU	Accuracy	Precision	Recall	F1
0.5	10.4%	84.6%	10.9%	74.1%	9.3%
0.85	11.9%	92.8%	14.7%	45.9%	11.2%

Table 1: Results of the Main model with different thresholds for the statistic computation. With threshold 0.5 we predict a lane if the segmentation mask outputs a value above 0.5 for a pixel.

the CULane test set, with the same settings as the model, and feeding of ground truth turned off. We run it on different thresholds, used to determine whether a lane is predicted or not, based on the results obtained in Figure 18.

5 Future work

5.1 Content queries

Inspired by the concept of content queries described in the O2SFormer paper, we worked in the direction of adding more positional prior information to the model. Similarly to the approach of O2SFormer, we make use of the DETR model ([Carion *et al.*, 2020]), which was initially developed to predict sets of bounding boxes. It utilizes a ResNet-50 backbone trained on ImageNet, a transformer architecture and a shared feed forward network that predicts the class of the detected object or the lack of such an object.

In order to adapt the DETR model to our use case, we drop the feed forward network at the end of its pipeline and use the feature vectors outputted by the decoder. As we use the loaded pre-trained model we also had to configure the number of object queries which are the input to the decoder alongside the encoder output. Each of these object queries specializes in finding information about objects in different parts of the image while going through the decoder layers. Thus, we take the feature vectors from the decoder output and incorporate them with the output of the MAE in our architecture. This setup is different from the one in O2SFormer where the DETR decoder output is fed into the decoder of the O2SFormer architecture as a target sequence. As mentioned before, we use the ground truth of the previous frame as the target sequence and the results from this cannot be outdone by the content queries. Unfortunately, we could not get good results from the addition of the DETR decoder output. We speculate that more fine-tuning and longer training are needed to make the model converge with this setup.

5.2 Training optimization

Another possible improvement that can be done in the future is an optimization of the training process. Currently, our model starts exhibiting signs of overfitting, i.e. significant difference between the training and validation losses, after a certain point in the training. This problem is overcome after some time and we see the validation loss falling to values close to the training loss. However, an even more optimal training could be performed if we stop feeding the ground truth from the previous frame to the decoder once the overfitting starts occurring and instead use the output of the decoder from the previous iteration. As shown in section 4, once we do not feed the ground truth, the validation loss decreases and continues decreasing in a more stable manner.

5.3 Network simplification

Lastly, one can simplify our architecture which might lead to quicker and more stable results. Because we did not have the time to run our model on a more powerful machine, we had to include down and up sampling in our architecture, as we could not afford so many parameters. One could possibly remove the down and up sampling which should make it easier for the network to become more precise. Additionally, one could further simplify the MAE decoder to output a smaller dimension, but this would require further work into how to unshuffle the shuffled latent space into a smaller output dimension.

5.4 Feature Extractor

As mentioned in section 4, we are unable to provide a full evaluation of the feature extractor in downstream lane detection tasks, primarily due to time and hardware constraints. When training the feature extractor on 30% of the data, and with a batch size of 8, preliminary results seem to indicate a better (more rapidly decreasing) loss than the MAE ViT pre-trained on ImageNet. Visual inspections when using the model on the downstream task did not yield satisfactory results. This would ideally be remedied by training the feature extractor on the full dataset, and on a dedicated GPU cluster. If we are to believe that reconstruction is a good indicator of a feature extractor, then this MAE ViT may significantly outperform other state-of-the-art models using ResNets as backbones.

6 Ethical and Technical Implications

Lastly, we would like to draw attention to the ethical and technical implications that should be considered with such a project. Due to the vast amounts of data required to train contemporary deep learning models, the quality, as well as quantity, of the data needs to be considered. It has been proven that deep learning models suffer from inherent bias present in the training data [Roselli *et al.*, 2019] [Mehrabi *et al.*, 2022], with bias leading to unintended outcomes, such as models that may have a racist or a sexist bias. As outlined in section 1, lane detection is inherently difficult due to the lack of consistency in lane markings, lane marking quality, per-country differences in road structure, and environmental conditions. Given this complexity, it is imperative to ensure the dataset used for training encompasses a broad range of scenarios, conditions, and geographical locations (worldwide, ideally). For instance, much of the research in deep learning emanates from developed, wealthier countries, meaning that the model is trained on predominantly well-marked, well-maintained roads. This is exactly the case with the dataset used throughout this project; the **CULane** dataset consists exclusively of clips from the Beijing area, resulting in a two-fold ethical consideration. First, this means that any model trained on this data could be unsafe to operate in regions that have geographical differences and infrastructure differences without extensive testing on a local dataset. Beyond this, this has the effect of making these models only usable in wealthier, developed countries. Under-developed countries, which suffer from worse infrastructure cannot benefit from the development of these lane detection models if they are exclusively trained on data from predominantly wealthy countries, making access to this technology unattainable to underdeveloped countries.

Privacy aspects can also be a consideration, the functionality of the model should serve only to predict lane position given a fixed number of frames K. Image frames should not be stored or used beyond the lifecycle needed to predict the lane markings.

References

[Carion *et al.*, 2020] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[Dong *et al.*, 2023] Yongqi Dong, Sandeep Patil, Bart van Arem, and Haneen Farah. A hybrid spatial–temporal deep learning architecture for lane detection. *Computer-Aided Civil and Infrastructure Engineering*, 38(1):67–86, 2023.

[He *et al.*, 2022] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[Hinton and Salakhutdinov, 2006] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[Hsiao *et al.*, 2009] Pei-Yung Hsiao, Chun-Wei Yeh, Shih-Shinh Huang, and Li-Chen Fu. A portable vision-based real-time lane departure warning system: Day and night. *IEEE Transactions on Vehicular Technology*, 58(4):2089–2094, 2009.

[Li *et al.*, 2019] Xiang Li, Jun Li, Xiaolin Hu, and Jian Yang. Line-cnn: End-to-end traffic line detection with line proposal unit. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):248–258, 2019.

[Mehrabi *et al.*, 2022] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning, 2022.

[Meng *et al.*, 2021] Depu Meng, Xiaokang Chen, Zejia Fan, Gang Zeng, Houqiang Li, Yuhui Yuan, Lei Sun, and Jingdong Wang. Conditional detr for fast training convergence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3651–3660, October 2021.

[Pan *et al.*, 2018] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Roselli *et al.*, 2019] Drew Roselli, Jeanna Matthews, and Nisha Talagala. Managing bias in ai. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 539–544, New York, NY, USA, 2019. Association for Computing Machinery.

[Tabelini *et al.*, 2021] Lucas Tabelini, Rodrigo Berriel, Thiago M Paixao, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Keep your eyes on the lane: Real-time attention-guided lane detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 294–302, 2021.

[Tang *et al.*, 2021] Jigang Tang, Songbin Li, and Peng Liu. A review of lane detection methods based on deep learning. *Pattern Recognition*, 111:107623, 2021.

[Vaswani *et al.*, 2023] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[Wang *et al.*, 2010] Jyun-Guo Wang, Cheng-Jian Lin, and Shyi-Ming Chen. Applying fuzzy method to vision-based lane detection and departure warning system. *Expert Systems with Applications*, 37(1):113–126, 2010.

[Wang *et al.*, 2011] Xiaoyun Wang, Yongzhong Wang, and Chenglin Wen. Robust lane detection based on gradient-pairs constraint. In *Proceedings of the 30th Chinese Control Conference*, pages 3181–3185. IEEE, 2011.

[Xingang Pan and Tang, 2018] Ping Luo Xiaogang Wang Xingang Pan, Jianping Shi and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[Yoo *et al.*, 2020] Seungwoo Yoo, Heeseok Lee, Heesoo Myeong, Sungrack Yun, Hyoungwoo Park, Janghoon Cho, and Duck Hoon Kim. End-to-end lane marker detection via row-wise classification, 2020.

[Zhou and Zhou, 2023] Kunyang Zhou and Rui Zhou. End-to-end lane detection with one-to-several transformer, 2023.

[Zou *et al.*, 2019] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE transactions on vehicular technology*, 69(1):41–54, 2019.

A Methodology section from the midterm

The approach we have taken is based on the O2SFormer architecture. As mentioned previously, this is one of the most recent papers on the topic of lane detection using transformers and it showcases state-of-the-art performance. Thus, it provides a good starting point to experiment with using sequential images in a transformer setting, which is ultimately the goal of the project. At this stage, the project is still in the implementation phase, therefore, this section of the report portrays our progress to this moment and our plans for the coming weeks.

First, we stick to the pre-processing steps that O2SFormer does, namely we resize the image into 320x800 image and normalize its values. We currently do not do randomized horizontal flipping, but we will want to add it in the coming weeks. The data is preprocessed so during training it loads the inputs and targets in the correct format, and these formats are also cached for quicker access.

The workflow of the system begins with a pre-trained convolutional neural network (ResNet18) - a so-called backbone. This CNN is trained on more than a million images from the ImageNet database and is commonly used in the literature as a feature extractor. ResNet is the CNN used by the O2SFormer and we chose ResNet18 specifically as the most lightweight ResNet architecture. The data from the CULane dataset is fed into the backbone network in batches, the size of which we are still experimenting with as we have hardware limitations. The output of the backbone per batch is a collection of vectors, each with the size of the input image. These vectors, which we refer to as segments, contain extracted features from the images of our input dataset.

The next step is to positionally encode the segments coming from the backbone. Positional encoding is used to provide information about the positions of elements in an input sequence and is a key component of any transformer-based model. Thus, we use positional encoding which computes sine and cosine functions in order to preserve the spatial information of the sequential images we have. The encoding used by us is inspired once again by the O2SFormer.

Once we have the batches of positionally encoded segments, we pass those to the transformer. It naturally consists of an encoder and a decoder. At this stage, we make use of the PyTorch implementation of these components. The O2SFormer paper does not suggest any changes to the encoder part, however, it adds two additional components to the decoder - content query and dynamic anchor-based positional query. These components are influenced by the Conditional DETR ([Meng *et al.*, 2021]) paper which proposes a division of the object query in DETR into content query and positional query. Object query in DETR is an attention mechanism that finds the important features in the encoder output and eventually produces the output of the decoder. This object query in DETR and the content and positional queries in the detection transformers replace the output of the previous step as an input to the decoder.

To train we have defined our own loss function which computes the MSE loss between all the points where a lane is supposed to be and all the points where it is not supposed to be. The final loss then consists of equally-weighted average of both of these sub-losses. We have done this for 2 reasons, first the loss in [Zhou and Zhou, 2023] is not well defined, as the output of the model does not correspond to the target, so some post-processing had to be done, which influences how the gradients for backwards pass are computed. Secondly, we wanted to ensure that the model learns to output something, as there are many more cells with no lane than ones with a lane.

Thus far, we have not added a content query and a dynamic anchor-based positional query as proposed by the O2SFormer. Instead, we feed the output of the previous prediction step back into the decoder as normally done in transformers for NLP tasks. Additionally, we still experiment with different output formats. Our initial idea was to represent the output as a grid over the image and each cell in this grid to have 3 parameters assigned - angle length and probability. The angle is between the predicted lane and the x-axis of the grid, the length and probability again refer to the lane and how likely it is to have one in that cell of the grid.

We have also tried a simpler architecture, where the model directly outputs a mask which is treated as a binary mask indicating where the lanes can be found. While in general, predicting the masks directly results in longer training times and the models are easier to overfit, it gives us the opportunity to use the Intersection over Union as our loss function. Therefore it should in theory be actively pushed towards making a better prediction per frame.

B Evaluation from midterm report

To evaluate the current state of our project, we use the same approach as other researchers working with CULane. The authors of the CULane dataset have created separate categories of videos, which correspond to a certain difficulty when trying to predict lanes. These categories consist of videos without lines, with cars crossing over lines, or videos done at night time. To measure how well does the model predict these lines we use the intersection over union score between binary mask generated from the output of the model and binary mask generated from the ground truth. Be aware that these results were trained for XYZ epochs on only 30% of the training data. These results are currently quite bad, as the models tend to learn to output the entire road rather than the specific lanes. The model that directly predicts a masked image suffers from the same problem.

Part of the evaluation is the intensity of computation. Currently we manage to treat 30 frames as one video (which is one batch), which gives an overall window of about 15 seconds, as there are 2 frames per second in the dataset. We use 30 as there are 180 or 90 frames in each video, and using 30 avoids having frames from different videos in one batch. At this stage for both approaches going through 30% of the dataset takes around 1 hour. One epoch over the entire dataset tends to take slightly less than 4 hours. While we can get some preliminary results, we will not be able to compare our models to other models, as they tend to train for around 20 epochs, which in our case would take 80 hours.

C CULane supplementary material

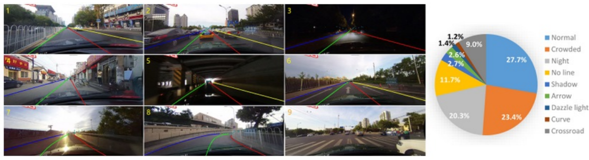

Figure 8: Examples for each challenging category in the test set.

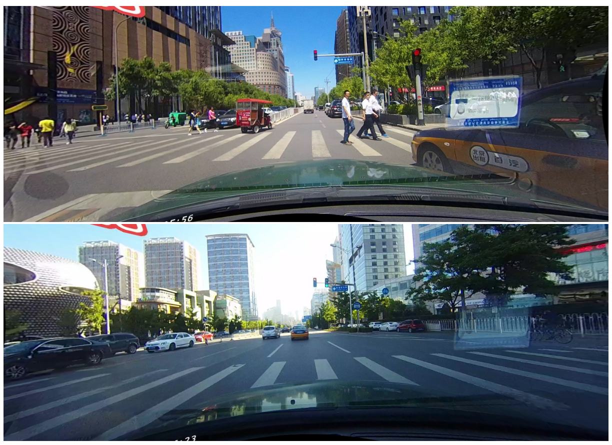

Figure 9: Examples of unannotated lanes in sight.

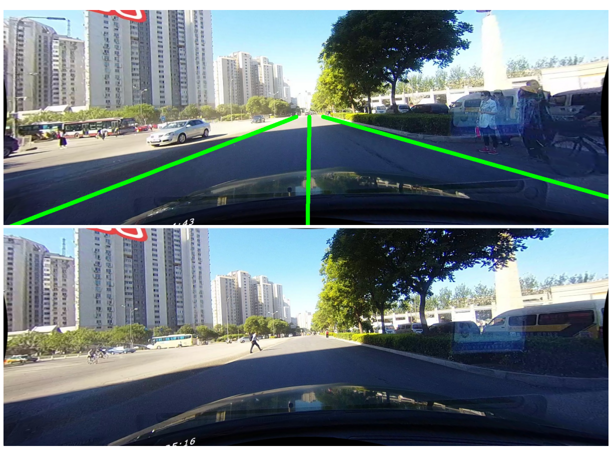

Figure 10: Lane annotations on the first image can be seen to not have much visual indication on the second image.

D Loss formulas and references

D.1 CE Loss

For more information on the specifics for this loss, and the Binary CE loss please visit the pytorch website for CE loss and the one for BCE loss.

E Model and hardware parameters

For the final version of the model we worked on, we have the following parameters:

```
d-model = 2304, nhead=1, out-dim=(224,224),
dropout = 0.1, num-decoder-layers = 1,
batch-size = 32, learning-rate= 1e-5,
weight-decay = 1e-7.
```

Hardware used for training:

CPU: AMD Ryzen 9 5900X 12-Core, 11th Gen Intel(R) Core(TM) i9-11900KF @ 3.50 GHz, AMD Ryzen 7 5600X

GPU: NVIDIA GeForce RTX 3070 Ti, NVIDIA GeForce GTX 1080 Ti

F Reconstruction Results from ResNet34 Auto-Encoder

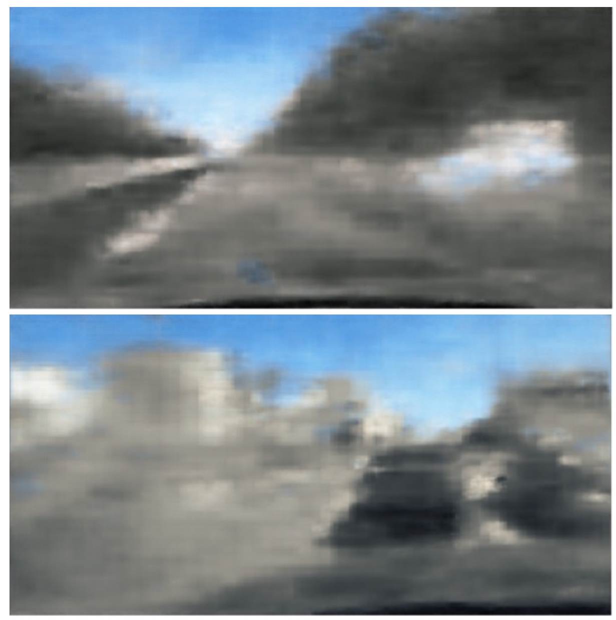

Figure 11: 2 images from the CULane train dataset, reconstructed by a decoder

G Reconstruction from a MAE ViT

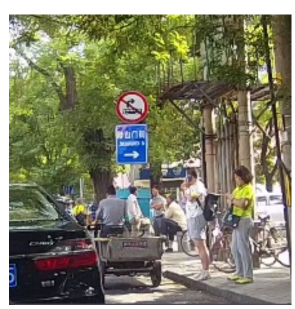

Figure 12: Input image to the MAE ViT feature extractor

Figure 13: Output image to the MAE ViT feature extractor

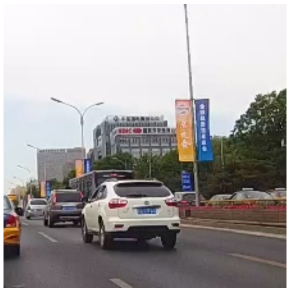

Figure 14: Input image to the MAE ViT feature extractor

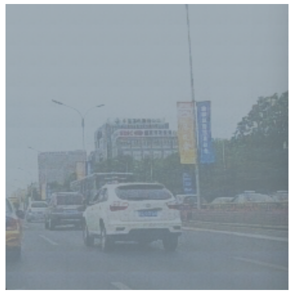

Figure 15: Output image to the MAE ViT feature extractor, lanes are clearly visible.

H More results

Figure 17 and Figure 16 present the loss and IoU when training without feeding the ground truth to the decoder. We see that again both the training and validation losses are decreasing. However, we have run this setup for only 6 epochs, starting from the checkpoint after the 16th epoch of the previously discussed training. Thus, the results are somewhat unstable and the validation loss is better than the training one which suggests that this model probably finds it easier to generilize, but the training set is filled with more instances of outliers. Additionally, Figure 19 displays the mean IoU for specific frames. One can observe that after approximately 25 epochs (first with ground truth being fed as input, then without) the network tends to exploit some information about the previous frame to get a better prediction. The overall variation is small between epochs so we can say that the IoU stays relatively stable. When it comes to the test set, Figure 18 illustrates the IoU score over different thresholds set for the confidence level. We notice that the best results are when the threshold is around 0.85 when the mean IoU reaches levels of about 0.12.

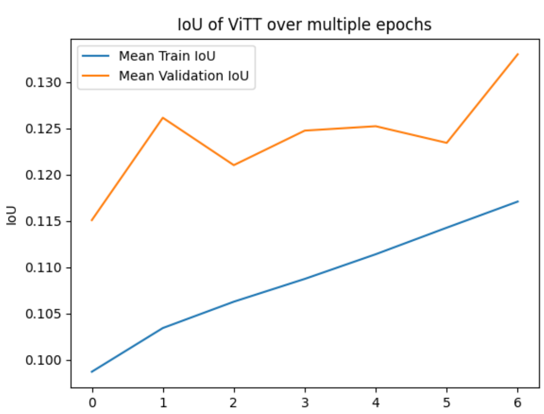

Figure 16: IoU over 6 epochs without using the ground truth in the decoder

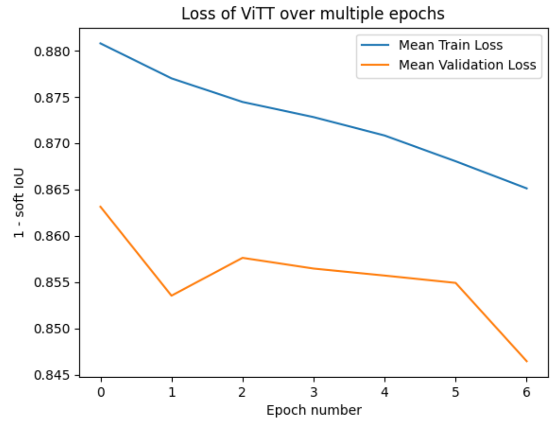

Figure 17: Loss over 6 epochs without using the ground truth in the decode

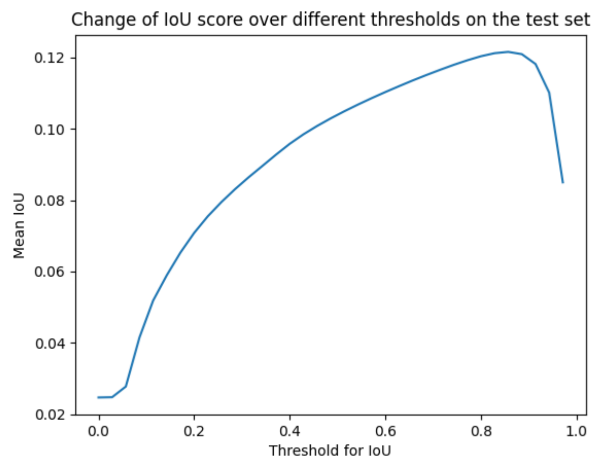

Figure 18: IoU over different thresholds for lane classification.

Figure 19: Average IoU for specific frames of the 15 frames which are being consumed by the model.

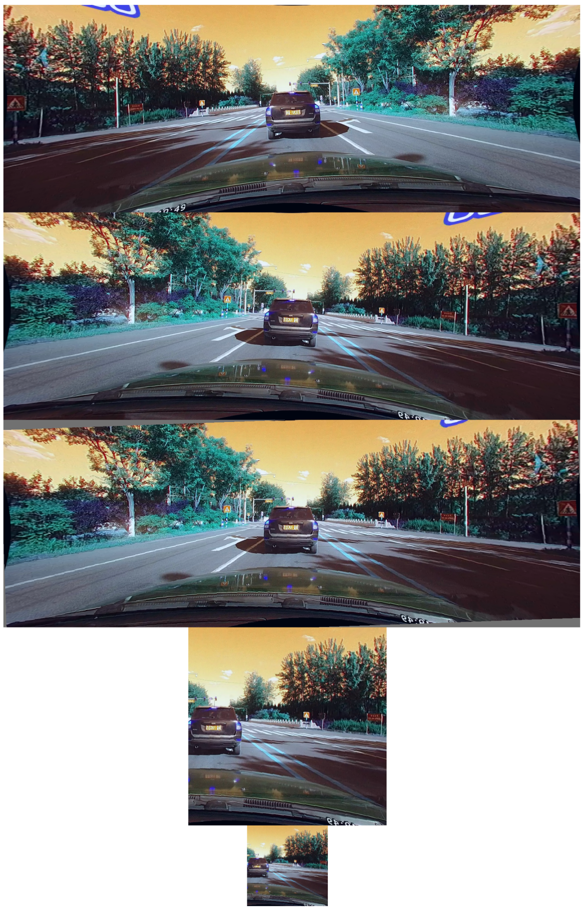

Figure 20: Visualization of the four data augmentation and preprocessing steps in the order they are executed from top to bottom. These are flips, rotations, cropping, and resizing. The colors appear distorted due to pixel value normalization.