Variational Autoencoder for End-to-End Control of Autonomous Driving with Novelty Detection and Training De-biasing

Alexander Amini[1], Wilko Schwarting[1], Guy Rosman[2], Brandon Araki[1], Sertac Karaman[3], Daniela Rus[1]

***Abstract*— This paper introduces a new method for end-to-end training of deep neural networks (DNNs) and evaluates it in the context of autonomous driving. DNN training has been shown to result in high accuracy for perception to action learning given sufficient training data. However, the trained models may fail without warning in situations with insufficient or biased training data. In this paper, we propose and evaluate a novel architecture for self-supervised learning of latent variables to detect the insufficiently trained situations. Our method also addresses training data imbalance, by learning a set of underlying latent variables that characterize the training data and evaluate potential biases. We show how these latent distributions can be leveraged to adapt and accelerate the training pipeline by training on only a fraction of the total dataset. We evaluate our approach on a challenging dataset for driving. The data is collected from a full-scale autonomous vehicle. Our method provides qualitative explanation for the latent variables learned in the model. Finally, we show how our model can be additionally trained as an end-to-end controller, directly outputting a steering control command for an autonomous vehicle.**

I. Introduction

Robots operating in human-centered environments have to perform reliably in unanticipated situations. While deep neural networks (DNNs) offer great promise in enabling robots to learn from humans and their environments (as opposed to hand-coding rules), substantial challenges remain [1]. For example, previous work in autonomous driving has demonstrated the ability to train end-to-end a DNN capable of generating vehicle steering commands directly from car-mounted video camera data with high accuracy so long as sufficient training data is provided [2]. But true autonomous systems should also gracefully handle scenarios with insufficient training data. Existing DNNs will likely produce incorrect decisions without a reliable measure of confidence when placed in environments for which they were insufficiently trained.

A society where robots are safely and reliably integrated into daily life demands agents that are aware of scenarios for which they are insufficiently trained. Furthermore, subsystems of these agents must effectively convey the confidence associated their decisions. Finally, robust performance of these systems necessitates an unbiased, balanced training dataset. To date, many such systems have been trained with datasets that are either biased or contain class imbalances, due to the lack of labeled data. This negatively impacts both the speed and accuracy of training.

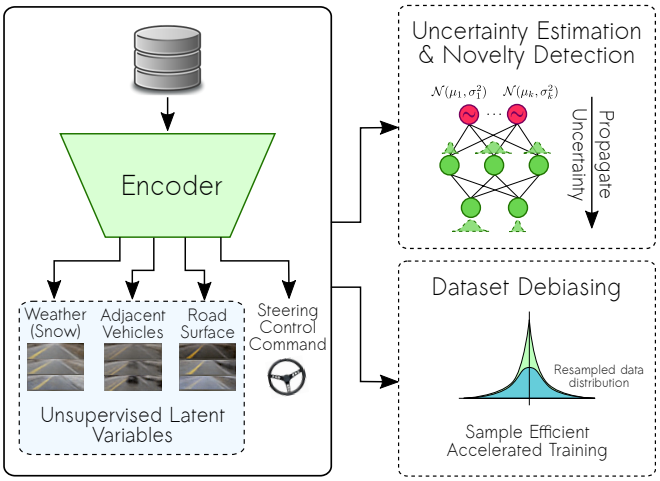

Fig. 1: **Semi-supervised end-to-end control.** An encoder neural network is trained to learn a *supervised* control command as well as various other *unsupervised* outputs that qualitatively describe the image. This enables two key contributions of novelty detection and dataset debiasing.

In this paper, we address these limitations by developing an end-to-end control method capable of novelty detection and automated debiasing through self-supervised learning of latent variable representations. In addition to learning a final control output directly from raw perception data, we also learn a number of underlying latent variables that qualitatively capture the underlying features of the data cf. Fig. 1. These latent variables, as well as their associated uncertainties, are learned through self-supervision of a network trained to reconstruct its own input. By estimating the distribution of latent factors, we can estimate when the network is likely to fail (thus increasing the robustness of the controller,) and adapt the training pipeline to cater to the distribution of these underlying factors, thereby improving training accuracy. Our approach makes two key contributions:

1) Detection of novel events which the network has been insufficiently trained for and not trusted to produce reliable outputs; and
2) Automated debiasing of a neural network training pipeline, leading to faster training convergence and increased accuracy.

Our solution uses a Variational Autoencoder (VAE) network architecture comprised of two parts, an encoder and a decoder. The encoder is responsible for learning a mapping from raw sensor data directly to a low dimensional latent

Toyota Research Institute (TRI) provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Volta V100 GPU used for this research.

[1] Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology {amini,wilkos,araki,rus}@mit.edu

[2] Toyota Research Institute (TRI) {guy.rosman}@tri.global

[3] Laboratory for Information and Decision Systems, Massachusetts Institute of Technology {sertac}@mit.edu

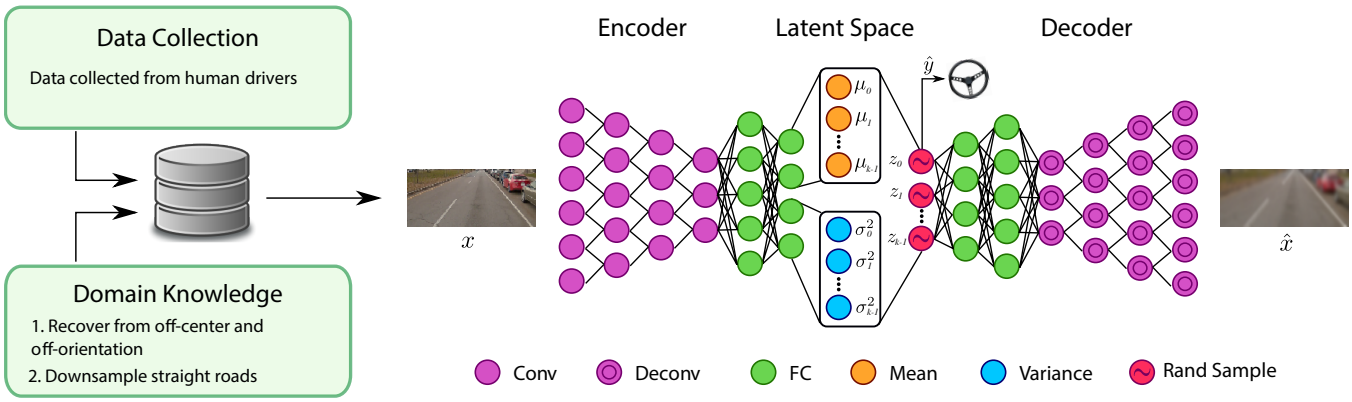

Fig. 2: **Novel VAE architecture for end-to-end control.** Image features are extracted through convolutional layers to encode the input image into the variational latent space with one of the latent variables explicitly supervised to learn steering control. The resulting latent variables are self-supervised by feeding the entire encoding into a decoder network that learns to reconstruct the original input image. Uncertainty is modeled through the variance of each latent variable (σ_k^2).

space encoding that maximally explains as much of the data as possible. The decoder is responsible for learning the inverse mapping that takes as input a single sample from the aforementioned latent space and reconstructs the original input. As opposed to a standard VAE model, which self-supervises the entire latent space, we also explicitly supervise a single dimension of the latent space to represent the robotic control output.

We use end-to-end autonomous driving as the robotic control use case. Here a steering control command is predicted from only a single input image. As a safety-critical task, autonomous driving is particularly well-suited for our approach. Control systems for autonomous vehicles, when deployed in the real world, face enormous amounts of uncertainty and possibly even environments that they have never encountered before. Additionally, autonomous driving is a safety critical application of robotics; such control systems must possess reliable ways of assessing their own confidence.

We evaluate our VAE-based approach on a challenging real driving dataset of time synchronized image, control data samples collected with a full scale autonomous vehicle and demonstrate both novelty detection and automated debiasing of the dataset. Our algorithm includes a VAE-based indicator to detect novel images that were not contained in the training distribution. We demonstrate our algorithm's ability to detect over 97% of novel image frames that were trained during a different time of day and detect 100% of camera sensor malfunctions in our dataset.

We address training set data imbalance by introducing unsupervised latent variables into the VAE model. These latent variables qualitatively represent many of the high level semantic features of the scene. Moreover, we show that we can estimate the distributions of the latent variables over the training data, enabling automated debiasing of newly collected datasets against the learned latent variables. This results in accelerated and sample efficient training.

The remainder of the paper is structured as follows: we summarize the related work in Sec. II, formulate the model in Sec. III, describe our experimental setup and dataset in Sec. IV, and provide an overview of our results in Sec. V.

II. Related Works

Traditional methods for autonomous driving first decompose the problem into smaller components, with an individual algorithm applied to each component. These submodules can range from mapping and localization [3], [4], to perception [5]–[7], planning [8], [9], and control [10], [11]. On the other hand, end-to-end systems attempt to collapse the entire problem (from raw sensory data to control outputs) into a single learned model. The ALVINN system [12] originally demonstrated this idea by training a multilayer perceptron to learn the direction a vehicle travel from pixel image inputs. Advancements in deep learning have caused a revival of end-to-end methods for self-driving cars [2], [13]–[15]. These systems have shown enormous promise by outputting a single steering command and controlling a full-scale autonomous vehicle through residential and highway streets [2]. The system in [13] learns a discrete probability distribution over possible control commands while [15] applies the output of their convolutional feature extractor to a long short-term memory (LSTM) recurrent neural network for generating a driving policy. However, all of these these models are still largely trained as black-boxes and lack a measure of associated confidence in their output and method for interpreting the learned features.

Understanding and estimating confidence of the output of machine learning models has been investigated in different ways: One can formulate training of DNNs as a maximum likelihood problem using a softmax activation layer on the output to estimate probabilities of discrete classes [16] as well as discrete probability distributions [17]. Introspective capacity has been used to evaluate model performance for a variety of commonly used classification and detection tasks [18] by estimating an algorithm's uncertainty by the distance from train to test distribution in feature space. Bayesian deep learning has even been applied to end-to-end autonomous vehicle control pipelines [19] and offers an additional way to estimate confidence through Monte Carlo dropout sampling of weights in recurrent [20] and convolutional neural networks [21]. However, Monte Carlo dropout sampling is not always feasible in real-time on many hardware platforms

due to its repetitive inference calls. Additionally, there is no explanation or interpretation of the resulting control actions available, as well as no confidence metric of whether the model was trained on similar data as the current input data.

Variational Autoencoders (VAEs) [22], [23] are gaining importance as an unsupervised method to learn hidden representations. Such latent representations have been shown to qualitatively provide semantic structure underlying raw image data [24]. Previous works have leveraged VAEs for novelty detection [25], but did not directly use the predicted distributions of the sensor for a model fit criterion. Conversely, our work presents an indicator to detect novel images that were not contained in the training distribution by weighting the reconstructed image by the latent uncertainty propagated through the network. High loss indicates that the model has not been trained on that type of image and thus reflects lower confidence in the network's ability to generalize to that scenario. Operating over input distributions diverging from the training distribution is potentially dangerous, since the model cannot sufficiently reason about the input data.

Additionally, learning in the presence of class imbalance is one of the key challenges underlying all machine learning systems. When the classes are explicitly defined and labeled it is possible to resample the dataset [26], [27] or even reweight the loss function [28], [29] to avoid training bias. However, these techniques are not immediately applicable if the underlying class bias is not explicitly labeled (as is the case in many real world training problems). While [30] demonstrates how K-means can be used to find clusters within the data before training to provide a loss reweighting scheme, this method does not adapt to the model during training. Additionally, running K-means on high dimensional image data can be extremely computational and destroys all spatial information between nearby pixels. This paper provides an algorithm for adaptive dataset debiasing during training by learning the latent space distribution and sampling from the most representative regions of this space.

III. Model

In this section, we start by formulating the end-to-end control problem and then describe our model architecture for estimating steering control of an autonomous vehicle (in an end-to-end manner). We explain how our model allows us to self-supervise on a number of other latent variables which qualitatively describe the driving dataset. All models in this paper were trained on a NVIDIA Volta V100 GPU.

A. End-to-End Model Formulation

We start from the end-to-end model framework. In this framework we observe n training images, $X = \{x_1, \ldots, x_n\}$, which are collections of raw pixels from a front-facing video camera. We aim to build a model that can directly map our input images, X, to output steering commands based on the curvature of the road $Y = \{y_1, \ldots, y_n\}$. To train a single image model we optimized the mean squared error (MSE) or L_2 loss function between the human actuated control, y_i, and the predicted control, $\hat{y}_i$:

$$\mathcal{L}_y(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{1}$$

Note that only a single image is used as input at every time instant. This follows from original observations where models that were trained end-to-end with a temporal information (CNN+LSTM) are unable to decouple the underlying spatial information from the temporal control aspect. While these models perform well on test datasets, they face control feedback issues when placed on a physical vehicle and consistently drift off the road.

B. VAE Architecture

We extend the end-to-end control model from Sec. III-A into a variational autoencoding (VAE) model. Unlike the classical VAE model [22], [23], one particular latent variable is explicitly supervised to predict steering control, and combined with the remaining latent variables to reconstruct the input image. Our model is shown in Fig. 2. The model accepts as input a $66 \times 200 \times 3$ RGB image in mini-batches of size $n = 50$. We use a convolutional network encoder, comprised of convolutional (Conv) and fully connected (FC) layers, to compute $Q(z|X)$, the distribution of the latent variables given a data point. The encoder has a similar architecture as a standard end-to-end regression model and is detailed in Table I, with 5 convolutional layers and two fully connected layers with dropout. The latent space section of Fig. 2 shows the encoder outputting $2k$ activations corresponding to $\boldsymbol{\mu} \in \mathbb{R}^k, \boldsymbol{\Sigma} = Diag[\sigma] \succ 0$ used to define the distribution of z. Next, there is a decoder network that mirrors the encoder (two FC layers with dropout and then 5 de-convolutional layers) to reconstruct the image back from the latent space sample z.

In order to differentiate the outputs through the sampling phase, VAEs use a reparameterization "trick", sampling $\epsilon \sim \mathcal{N}(0, I)$ and compute $z = \boldsymbol{\mu}(X) + \boldsymbol{\Sigma}^{1/2}(X) \times \epsilon$. This allows us to train the encoder and decoder, end-to-end, using backpropagation.

In our VAE model we additionally supervise one of the latent variables to take on the value of the curvature of the vehicle's path. We represent this modified variable as $z_0 = z_{\hat{y}} = \hat{y}$. It is visible in Fig. 2 as the steering wheel graphic. The loss function of our VAE has three parts – a reconstruction loss, a latent loss, and a supervised latent loss. The reconstruction loss is the L_1 norm between the input image and the output image, and serves the purpose of

TABLE I: **Architecture of the encoder neural network.** "Conv" and "FC" refer to convolutional and fully connected layers, while k denotes the number of latent variables.

Layer	Output	Activation	Kernel	Stride
1. Input	66x200x3	Identity	-	-
2. Conv1	31x98x24	ReLU	5x5	2x2
3. Conv2	14x47x36	ReLU	5x5	2x2
4. Conv3	5x22x48	ReLU	5x5	1x1
5. Conv4	3x20x64	ReLU	3x3	1x1
6. Conv5	1x18x64	ReLU+Flatten	3x3	1x1
7. FC1	1000	ReLU	-	-
8. FC2	100	ReLU	-	-
9. FC3	$2\,k$	Identity	-	-
10. Latent Sample	k	Identity	-	-

training the decoder. We define this loss function as follows:

$$\mathcal{L}_x(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - \hat{x}_i| \quad (2)$$

The latent loss is the Kullback-Liebler divergence between the latent variables and a unit Gaussian, providing regularization for the latent space [22], [23]. For Gaussian distributions, the KL divergence has the closed form

$$\mathcal{L}_{KL}(\mu, \sigma) = -\frac{1}{2} \sum_{j=0}^{k-1} \left(1 - \mu_j^2 - \sigma_j^2 + \log\left(\sigma_j + \rho\right)^2\right) \quad (3)$$

where $\rho = 10^{-8}$ is added for numerical stability.

Lastly, the supervised latent loss is a new addition to the loss function, and it is defined as the MSE between the predicted and actual curvature of the vehicle's path. It is the same as Eq. 1 from Sec. III-A.

These three losses are summed to compute the total loss:

$$\mathcal{L}_{TOTAL}(\cdot) = c_1 \mathcal{L}_y(y, \hat{y}) + c_2 \mathcal{L}_x(x, \hat{x}) + c_3 \mathcal{L}_{KL}(\mu, \sigma) \quad (4)$$

where c_1, c_2, and c_3 are used weight the importance of each loss function. We found that $c_1 = 0.033$, $c_2 = 0.1$, $c_3 = 0.001$ yielded a nice trade off in importance between steering control, reconstruction, and KL loss, wherein no one individual loss component overpowers the others during training.

C. Novelty Detection

A crucial aspect underlying many deep learning systems is their ability to reliably compute uncertainty measurements and thus determine when they have low confidence in the given environment. Many standard end-to-end networks are seen as black-boxes, producing only a single control output number at the final layer. In this section, we explore a novel method for using the architecture to detect driving environments with insufficient training that cannot be trusted to produce a reliable decision.

We note the VAE architecture provides uncertainty estimates for every variable in the latent space via their parameters $\{\mu_i, \sigma_i\}_{i=0}^{k-1}$. However, what we really desire are uncertainty estimates in the input image space since these observations are available at test time. We therefore propagate the uncertainties through the remainder of the network by sampling in the latent layer space and computing empirical uncertainty estimates in any of the future layers (including the final reconstructed image space).

This can be explained by treating the VAE encoder network as a posterior model inference of the parameters θ, z as samples from the posterior distribution inferred from θ. Propagation of z into $\hat{x}$ results in a posterior predictive sample. A common model fit measure for θ would be

$$\log P(x|\theta); \quad \theta = \{\sigma_i, \mu_i\}_{i=0}^{k-1} \quad (5)$$

Using a common pixel-wise independence approximation allows us to get a model rejection criteria based on the images, using a weighted L_2 error. As commonly done in image processing, we use instead the L_1 norm due to its robustness. Averaging over image pixels yields the error term:

$$D(x, \hat{x}) = \left\langle \frac{\left|x^{(p)} - \mathbb{E}\left[\hat{x}^{(p)}|\theta\right]\right|}{\sqrt{\mathrm{Var}\left[\hat{x}^{(p)}|\theta\right]}} \right\rangle \quad (6)$$

where $\mathbb{E}\left[\hat{x}^{(p)}|\theta\right], \mathrm{Var}\left[\hat{x}^{(p)}|\theta\right]$ are computed by sampling values of z and computing statistics for the resulting decoded images $\hat{x}$. Additionally, $x^{(p)}$ denotes the value of pixel p in image x. The approach for pixelwise uncertainty estimates is similar to the unscented transform [31], and is captured in Algorithm 1. Using these statistics indicates whether an observed image is well captured by our trained model. In practice, we implemented a real-time version of this algorithm using 20 samples on each time iteration.

Algorithm 1 Compute pixel-wise expectation and variance

Require: Input image x, Encoder NN, & Decoder NN

$\{\sigma_i, \mu_i\}_{i=1}^{k} \leftarrow Encoder(x)$
$\theta \leftarrow \{\sigma_i, \mu_i\}_{i=1}^{k}$

for $j = 1$ **to** T **do**
  for $i = 1$ **to** k **do**
    Sample $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$
  end for
  $\hat{x}_j = Decoder(z)$
end for

$\mathbb{E}[\hat{x}^{(p)}|\theta] = \frac{1}{T} \sum_{j=1}^{T} \hat{x}_j^{(p)}$
$\mathrm{Var}[\hat{x}^{(p)}|\theta] = \frac{1}{T} \sum_{j=1}^{T} \left(\hat{x}_j^{(p)} - \mathbb{E}[\hat{x}^{(p)}|\theta]\right)^2$

return $\mathbb{E}[\hat{x}|\theta]$, $\mathrm{Var}[\hat{x}|\theta]$

D. Increased Training on Rare Events

A majority of driving data consists of straight road driving, while turns and curves are vastly underrepresented. End-to-end networks training often [2] handles this by resampling the training set to place more emphasis on the rarer events (i.e., turns). We generalize this notion to the latent space of our VAE model, better exploring the space of both control events and nuisance factors, for not only the steering command but all other underlying features. By feeding the original training data through the learned model we estimate the training data distribution $Q(z|X)$ in the latent space. Subsequently, it is possible to increase the proportion of rarer datapoints by dropping overrepresented regions of the latent space. We approximate $Q(z|X)$ as a histogram $\hat{Q}(z|X)$, where z is the output of the Encoder NN corresponding to the input images $x \in X$. To avoid data-hungry high-dimensional (in our case 25 dimensions) histograms, we further simplify and utilize independent histograms $\hat{Q}_i(z_i|X)$ for each latent variable z_i and approximate $\hat{Q}(z|X) \propto \prod_i \hat{Q}_i(z_i|X)$. Naturally, we would like to train on a higher number of unlikely training examples and drop many samples overrepresented in the dataset. We therefore train on a subsampled training set X_{sub} including datapoints x with probability $W(z(x)|X) \propto \prod_i 1/(\hat{Q}_i(z_i(x)|X) + \alpha)$. For small α the

subsampled training distribution is close to uniform over z, whereas large α keep the subsampled training distribution closer to the original training distribution. At every epoch all images x from the original dataset X are propagated through the (learned) model to evaluate the corresponding latent variables $z(x)$. The histograms $\hat{Q}_i(z_i(x)|X)$ are updated accordingly. A new subsampled training set X_{sub} is drawn by keeping images x from the original dataset X with likelihood $W(z(x)|X)$. Training on the subsampled data X_{sub} now forces the classifier into a choice of parameters that work better in rare cases without strong deterioration of performance for common training examples. Most importantly, the debiasing is not manually specified beforehand but based on *learned* latent variables.

IV. Dataset

The vehicle base platform employed to collect the dataset is a Toyota Prius 2015 V used in collaboration with the MIT-Toyota partnership. A forward facing Leopard Imaging LI-AR0231-GMSL camera, which has a field-of-view of 120 degrees and captures 1080p RGB images at approximately 30Hz, is the vision data source for this study. Sensors also collected the human actuated steering wheel angle (rad) as well as vehicle speed (m/sec). A Xsens MTi 100-series Inertial Measurement Unit (IMU) was used to collect acceleration, rotation, and orientation data from the rigid body frame and thus compute the curvature of the vehicle's path. Specifically, given a yaw rate γ_i (rad/sec), and the speed of the vehicle, v_i (m/sec), we compute the curvature (or inverse radius) of the path as $y_i = \frac{\gamma_i}{v_i}$. Note that we can model a simple relationship between steering wheel angle and road curvature given the vehicle slip angle by approximating the vehicle according to the Bicycle Model. While we employ curvature (y_i) to model our networks, for the remainder of this paper we will use the terms "steering control" and "curvature" interchangeably, since we can compute one given the other by estimating slip angle directly from IMU data.

All communication and data logging was done directly on an NVIDIA PX2, which is an in-car supercomputer specifically developed for autonomous driving. As part of data collection for this project we setup the PX2 to connect, communicate, and control a full-scale Toyota Prius with drive-by-wire. Additionally, we developed the software infrastructure to read the video stream and synchronize with the other data streams (inertial and steering data) on the PX2.

We drove the vehicle in the Boston metropolitan area and collected data for approximately 4 hours (which was split 3/1 into training and test sets). In the following subsection, we outline the data processing and augmentation techniques that were performed prior to training our models.

A. Processing and Augmentation

A number of preprocessing steps were taken to clean and prepare the data for training. First, we annotated each frame of collected video according to the time of day, road type, weather, and maneuver (lane-stable, turn, lane change, junk). This labeling process allowed us to segment out the pieces of our data which we wanted to train with. Note the labels were not used as part of the training process, but only for data filtering and preprocessing. Following previous work [2] we used only "lane-stable" in our dataset, which corresponds to sections of driving where the vehicle is stable in its lane.

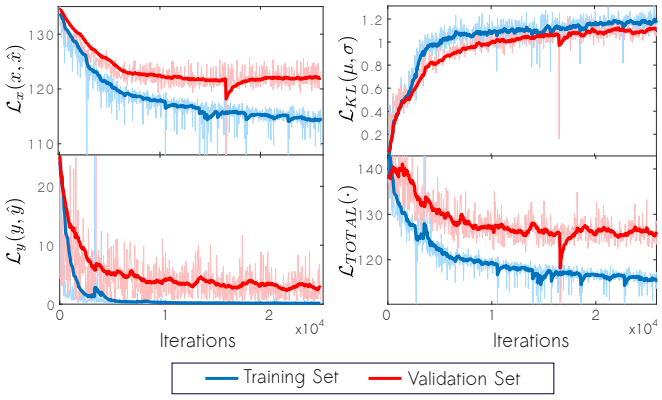

Fig. 3: **Loss evolution**. Convergence of our four loss functions (defined in Eqs. [1]-[4]) on the train (blue) and validation (red) data.

To inject domain knowledge into our network we augmented the dataset with images collected from cameras placed approximately 2 feet to the left and right of the main center camera. We correspondingly changed the supervised control value to teach the model how to recover from off-center positions. For example, an image collected from the right camera we perturb the steering control with a negative number to steer slightly left, and vice versa. We add images from all three cameras (center, left, and right) to our dataset for training.

B. Optimization

We trained our models with the Adam optimizer [32] with $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We considered the number of latent variables, k, to be a hyperparameter and trained models with 400, 100, 50, 25, and 15 latent variables. By analyzing the validation error upon convergence, we were able to identify that the model with 25 latent variables provided the best fit of our dataset while providing realistic reconstructions. Therefore, we use $k = 25$ for the rest of our analysis. Fig. 3 shows the evolution of all four losses (as defined in Eqs. 1-4) for the training and validation sets over 2.6×10^4 steps. The MSE steering loss converges to nearly 0 for training but decreases more slowly for the validation set. Meanwhile, the KL divergence loss of Eq. 3 increases rapidly before plateauing at a relatively low value for both training and validation data. This is to be expected, since the latent variable distributions are initialized as $\mathcal{N}(0,1)$ and are then perturbed to find approximations of the Gaussian that allow the distributions to best reduce the overall loss. Since we use 25 latent variables to model an image with $66 \times 200 \times 3 = 39,600$ dimensions, it is expected that the decoder cannot exactly reproduce the input image (i.e. $\mathcal{L}_x(x,\hat{x}) > 0$).

V. Results

A. Steering Control Accuracy

To evaluate the performance of our model for the task of end-to-end autonomous vehicle control of steering we start by training a standard regression network which takes as input a single image and outputs steering curvature [2]. The

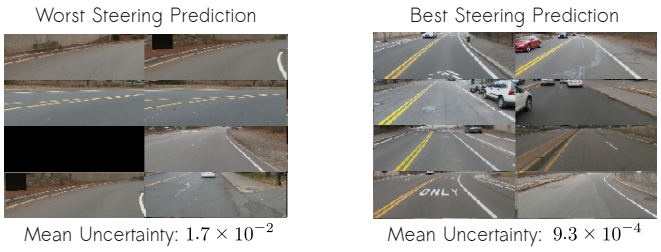

Fig. 4: **Bad apple analysis**. Test images with the best (right) and worst (left) steering error.

architecture of this model is almost identical to the first nine layers of our encoder neural network with the exception of the final layer being only one neuron (as opposed to $2k$ neurons). We found that the training loss of both the regression and our VAE network were almost exactly the same. The steering validation loss for the VAE was roughly 4.4, versus a value of 3.8 for the regression model loss. Therefore the loss is 14% higher, corresponding to a mean error in steering wheel angle of only 1.4 degrees. Therefore, we use the VAE model to predict both a steering control and uncertainty measure with roughly equal accuracy as the baseline regression model but simultaneously gain all of the additional advantages from the learned latent encodings.

Fig. 4 shows the images associated with the best and worst steering predictions. The mean uncertainty of the best predictions was 9.3×10^{-4} vs 1.7×10^{-2} for the worst predictions, indicating that our model can indeed predict when its estimated steering angle is uncertain. The images associated with the worst steering position are mostly from when the car is at a high angle of incidence towards the center of the road, representing a small portion of our training dataset. On the other hand, the images with the best steering prediction are mostly straight roads with strongly defined lanes, probably because our dataset has many examples of straight roads (despite debiasing) and because lane markers are a key feature for predicting steering angle.

Fig. 5 corroborates these results. The first chart shows that there is a strong correlation between the true steering and the estimated steering. Interestingly, the values of the estimated steering fan out at the extreme values, showing that 1) our dataset is sparse at extreme steering values and 2) that the uncertainty of the prediction should increase at extreme steering values. The second chart shows the relationship between true steering and the estimated uncertainty, and it indeed shows that uncertainty increases as the absolute value of the steering increases. Although this shows a weak point of a dataset – that it is sparse in images associated with large steering angle – it shows a strong point of the VAE model, which is that it is able to predict that there is high uncertainty for large values of the steering angle. Additionally, such uncertainty at greater turns makes intuitive sense since larger road curvatures imply less future road is visible in the image. For example, on extreme turns it is common for less than 10m to be visible in image, whereas straight road images present can present visible far more into the future closer to 100m. The fact that we can see less of the future road supports the increased uncertainty in such scenarios.

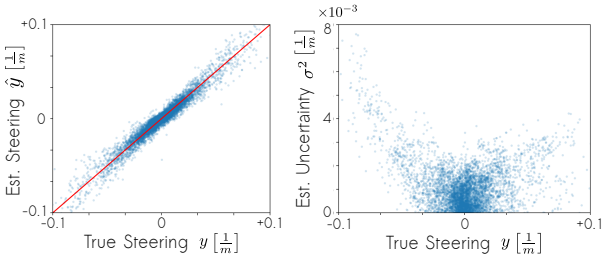

Fig. 5: **Precision and uncertainty estimation**. Plots of true vs. estimated steering (left) and true steering vs. estimated uncertainty (right). Both show that the model variance tends to increase on larger turns (i.e., greater steering magnitude).

B. Latent Variables

In this subsection, we analyze the resulting latent space that our encoder learned. We start by gauging the underlying meaning of each of the latent variables. We synthesize images using our decoder, starting with a vector in the latent space, and perturb a single latent variable as we reconstruct output images. By varying the perturbation we can understand how that specific latent variable effects the image. The results for an examplary set of latent variables is shown in Fig. 6. We observe that the network is able to generate intuitive representations for lane markings and additional environmental structure such as other surrounding vehicles and weather without ever actively being told to do so. By identifying latent variable representations we can immediately observe what the network sees and explain how the corresponding steering control is derived.

Intuitively, we know that the network will be penalized for having redundant latent variables due to the fact that the reconstruction loss penalizes reconstructed images that do not resemble the input image. This means that the encoder should learn a latent representation of the input such that as much distinct information is explained away as possible. This causes the variables learned to be the most important underlying variables in the dataset so the decoder can reconstruct the image from such a low dimensional space.

C. Detecting out of sample environments

Next, we examined ways to interpret if our network is confident in its end-to-end control prediction. Figure 7 shows sample pixel-wise uncertainties (red) obtained from this method overlaid on top of the respective input images. As one might expect, details around adjacent vehicles, the distant horizon and the presence of snow next to the road highlight the greatest uncertainty. This make sense since the model is constrained to only 25 latent variables and doesnot have capacity to hold any of these less meaningful details.

We can now plot the distribution of this distance over datasets on which the network received sufficient and insufficient training data. To be very explicit, we train a network on daytime driving and detect novel nighttime driving. Since the network has not been trained with night data, we should not trust the output and therefore want to understand, as best as possible, when nighttime data is being fed into the network. We set a simple threshold γ_{95} at the $95th$ percentile of $D(x, \hat{x})$ for every x in the entire training set. For any

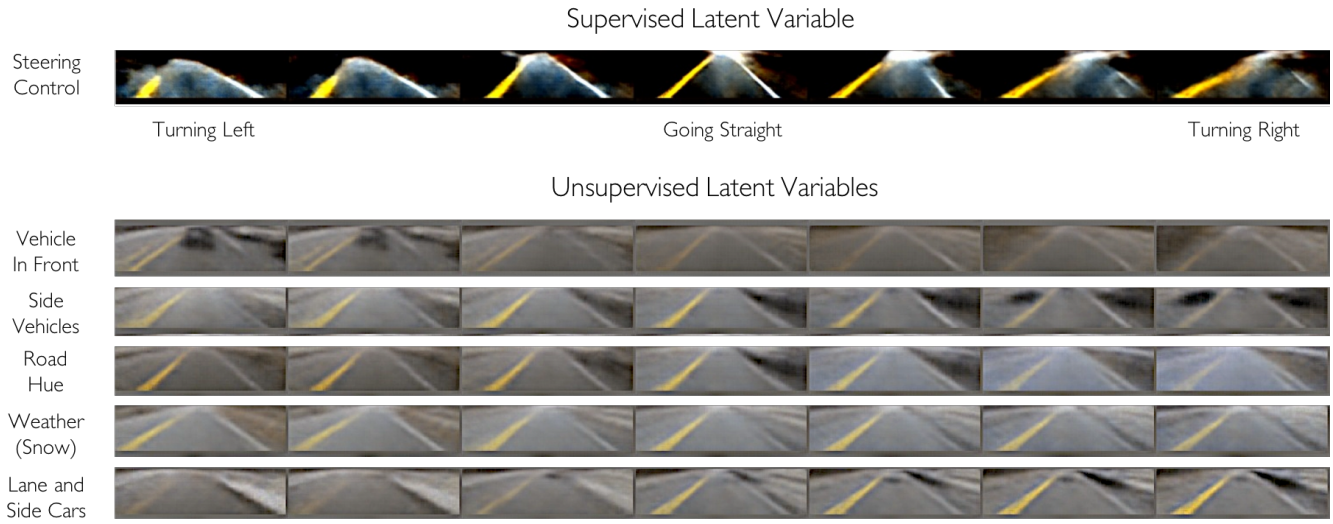

Fig. 6: **Latent variable perturbation**. A selection of six learned latent variables with associated interpretable descriptors (left labels). Images along the x-axis (from left to right) were generated by linearly perturbing the latent vector encoding along that single latent dimension. While steering command (top) was a supervised latent variable, all others (bottom five) were entirely unsupervised and learned by the model from the dataset.

Fig. 7: **Propagating uncertainty into pixel-space**. Sample images from the dataset along with pixel-wise uncertainty estimates. Uncertain pixels are highlighted red.

subsequent datasample x_{test}, we classify it as novel if

$$D(x_{test}, \hat{x}_{test}) > \gamma_{95} \tag{7}$$

Figure 8 illustrates the training set distribution (daytime driving) in blue as well as the extracted threshold γ_{95}. When image frames collected at night are fed through network we observe the orange distribution and are able to correctly classify 97% of these frames as novel. Furthermore, we experiment with frames collected during image sensor malfunctions. These malfunctions are a realistic and common failure mode for the AR0231 sensor caused from incorrect white balance estimation. Feeding such images through the network can be catastrophic and yield unpredictable control responses. Using our metric $D(x, \hat{x})$ we see that the distribution of these images (green) fall far from the training set and we can successfully detect 100% of these frames.

D. Debiasing the model

To evaluate the effect of debiasing during training we train two models, one without debiasing the dataset for inherent latent imbalances and once again now subsampling our dataset to reduce these over-represented (i.e., biased) samples. On every epoch we sample only 50% of the dataset for training while the remaining data is discarded. Figure 9 illustrates the loss evolution throughout both training schemes. Note that the loss is computed on the original data distribution (and not the subsampled distribution), since we ultimately only care about our performance on the original data. Debiasing the training pipeline allows the model to focus on events that are typically more rare (and inherently more difficult since they occur less frequently). This results in training that is more data efficient (using only 50% of the data), and also

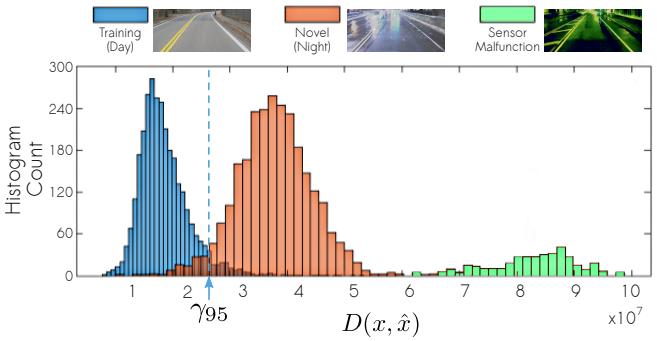

Fig. 8: **Novelty Detection**. Feeding training distribution data (day time) through the network as well as a novel data distribution (night time). A third peak forms when the camera sensor malfunctions from poor illumination.

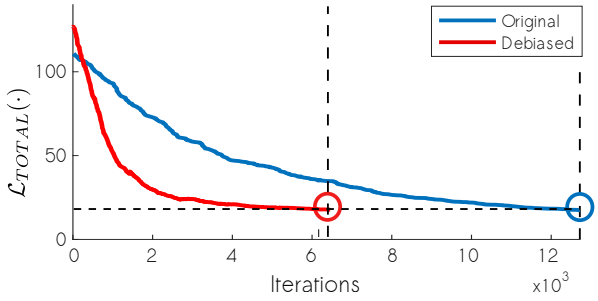

Fig. 9: **Accelerated training with debiasing**. Comparison of training loss evolution with/without automated debiasing.

faster than standard training. Figure 9 shows a minimum loss of 20 achieved after roughly half as many training iterations compared to training on the original data distribution.

VI. Conclusion

This paper presents a novel deep learning-based algorithm for end-to-end autonomous driving that also captures uncertainty through an intermediate latent representation. Specifically, we built a learning pipeline that computes a steering control command directly from raw pixel inputs of a front facing camera. Compared to previous research on end-to-end driving, our model also captures uncertainty of the final control prediction. We treat our input image data as being modeled by a set of underlying latent variables (one of which is the steering command taken by a human driver) with a VAE architecture. Additionally, we propose novel method for detecting novel inputs which have not been sufficiently trained for by propogating the VAE's latent uncertainty through the decoder. Finally, we provide an algorithm for debiasing against learned biases based on the unsupervised latent space. By adaptively subsampling half of the dataset throughout training, we remove the over-represented latent regions and empirically observe $2\times$ training speedups.

While the steering command latent variable is explicitly supervised by ground truth human data and can be used to control the vehicle, we also learn many other latent variables in an unsupervised manner. We show that these unsupervised latent variables represent concrete and interpretable features in the driving scene, such as presence of different lane markers, surrounding vehicles, and even the weather.

Our approach is scalable to massive driving datasets since it does not require any manual data-labeling of the supervised signals. While previous work in end-to-end learning presents a form of reactionary control, lane following, and object avoidance, this technique encodes a much richer set of information in a probability distribution. Furthermore, since autonomous driving is an extremely safety critical application of AI, the uncertainty measurements that we provide are absolutely crucial for end-to-end techniques to be deployed onto real-world roads.

References

[1] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 187–210, 2018.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[3] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[4] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[5] A. Amini, B. Horn, and A. Edelman, "Accelerated Convolutions for Efficient Multi-Scale Time to Contact Computation in Julia," *arXiv preprint arXiv:1612.08825*, 2016.

[6] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*. IEEE, 2008, pp. 82–88.

[7] Z. Kim, "Robust lane detection and tracking in challenging scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, 2008.

[8] U. Lee, S. Yoon, H. Shim, P. Vasseur, and C. Demonceaux, "Local path planning in a complex environment for self-driving car," in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, 2014.

[9] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," in *Proceedings of the International Symposium on Robotics Research (ISRR 2015), Sestri Levante, Italy*, 2015.

[10] W. Schwarting, J. Alonso-Mora, L. Paull, S. Karaman, and D. Rus, "Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model," *IEEE Transactions on Intelligent Transportation Systems*, no. 99, pp. 1–15, 2017.

[11] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on control systems technology*, 2007.

[12] D. A. Pomerleau, "ALVINN, an autonomous land vehicle in a neural network," Carnegie Mellon University, Computer Science Department, Tech. Rep., 1989.

[13] A. Amini, L. Paull, T. Balch, S. Karaman, and D. Rus, "Learning steering bounds for parallel autonomous systems," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[14] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *arXiv preprint arXiv:1612.01079*, 2016.

[15] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *CoRR*, vol. abs/1605.06450, 2016.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[17] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," *CoRR*, vol. abs/1612.05533, 2016.

[18] H. Grimmett, R. Triebel, R. Paul, and I. Posner, "Introspective classification for robot perception," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 743–762, 2016.

[19] A. Amini, A. Soleimany, S. Karaman, and D. Rus, "Spatial Uncertainty Sampling for End-to-End control," in *Neural Information Processing Systems (NIPS); Bayesian Deep Learning Workshop*, 2017.

[20] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1019–1027.

[21] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[23] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[24] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, "Variational autoencoder for deep learning of images, labels and captions," in *Advances in neural information processing systems*, 2016, pp. 2352–2360.

[25] C. Richter and N. Roy, "Safe visual navigation via deep learning and novelty detection," in *Proc. of the Robotics: Science and Systems Conference*, 2017.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[27] A. More, "Survey of resampling techniques for improving classification performance in unbalanced datasets," *arXiv preprint arXiv:1608.06048*, 2016.

[28] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.

[29] S. Suresh, N. Sundararajan, and P. Saratchandran, "Risk-sensitive loss functions for sparse multi-category classification problems," *Information Sciences*, vol. 178, no. 12, pp. 2621–2638, 2008.

[30] G. H. Nguyen, A. Bouzerdoum, and S. L. Phung, "A supervised learning approach for imbalanced data sets," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 2008, pp. 1–4.

[31] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. Ieee, 2000, pp. 153–158.

[32] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.