A Deep Learning Approach to Vehicle Speed Estimation

Benjamin Penchas bpenchas@stanford.edu
Tobin Bell tbell@stanford.edu
Marco Monteiro marcorm@stanford.edu

ABSTRACT

Given car dashboard video footage, we aimed to estimate the speed of a car using a deep neural network. We saw this problem as a small but important part of building an autonomous vehicle. The problem is by its nature underdetermined (since we have no absolute reference for scale/distance), so we treated it as a classification task where we bucketed speeds into 4 mph intervals. We designed our own network architecture after studying common optical flow networks and optimized the hyperparameters by experimentation. After training on cloud GPUs for several epochs, we were able to achieve 91% accuracy on unseen test frames.

I. INTRODUCTION

The ultimate goal of this research was to estimate the speed of a car in real time given a mounted dashboard video stream—a somewhat novel application of deep CNNs. Our dataset was a dashboard video taken by driving around the Bay Area. The dataset contains footage from a variety of road types and speeds. We took our data from the autonomous vehicle startup Comma AI's speed detection challenge[1] . With the video, Comma AI provides a text file with the ground truth speed of the car (m/s) for every frame. We separated this video into 20,400 image frames (an example of one such frame is given below), scaled each image to 224×224, and coupled every pair of frames. We then randomly shuffled these pairs and partitioned 80% into a training set and 20% into a test set.

[1] twitter.com/comma_ai/status/854488327797448704

Figure 1. *Example video frame from our car video dataset.*

II. STUDYING OPTICAL FLOW

We began by studying current state-of-the-art optical flow systems. In the beginning, we saw the problem of dense optical flow as very similar to our video-to-speed problem, but not exactly the same. Rather than use a pre-trained optical flow model such as Flownet, a very large network, we wanted to design and train our own lightweight optical flow model specific to this problem. To experiment with optical flow, we got a pre-trained optical flow network working and tried using it as a feature extractor before our network. Below is included sample output from that network on one pair of image frames.

Figure 2. *Flownet output (right) for subsequent frames of driving footage (left, blended).*

However, calculating the dense optical flow of every pair of images in our dataset would have had extremely high overhead and we were not convinced we required the full dense optical flow to estimate the speed; *i.e.*, we hypothesized that a custom network might learn the optical flow of the white lines in the road and use

this flow without looking at the flow of every pixel in the image. We studied the architecture of such optical flow networks, however, and decided that the two-stream feature extractor at the beginning of the network was important.

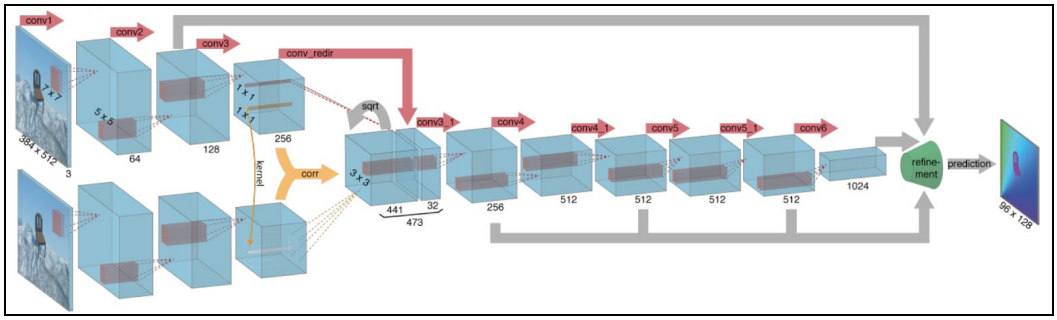

Figure 3. *Flownet Architecture. Notice the two-stream feature extractor.*

III. FEATURE EXTRACTION WITH VGG-19

In keeping with this architecture, we decided to first try a simple CNN architecture with a two-stream feature extractor at the beginning. We convinced ourselves that pre-training on ImageNet would be valuable because scale is an important part of the problem—*i.e.*, how big cars, trees, signs typically are. Hence we initially used the first few layers of a pre-trained VGG-19 network as a two-stream feature extractor, mimicking the Flownet architecture and hoping to extract features like scale. We fed each image through a pretrained VGG-19 model, up to the third block, and then concatenated the results.

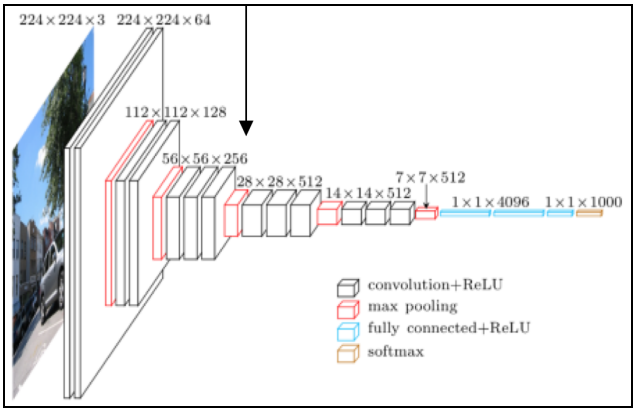

Figure 4. *VGG-19 architecture. The arrow marks the cutoff point for our architecture.*

We then fed the resulting volume through our trainable custom CNN. However, this approach failed for two reasons.

1. VGG-19 is an image classifier, and so it has been trained to extract semantic understanding. However, since subsequent frames are semantically very similar, the feature extraction with VGG-19 actually got rid of crucial spatial information. Using VGG-19 as a feature extractor led to a model that could not distinguish between visually similar images, such as subsequent frames of driving video.

2. Additionally, VGG-19 has rapid downsampling with several max-pool layers. This architecture caused a loss of resolution in the images. For optical flow, and this video-to-speed problem, high spatial resolution is very important.

Thus we decided not to use a pre-trained feature extractor and to just let the network learn everything on its own.

IV. NETWORK ARCHITECTURE

Below is included a graphic of our final architecture. We begin by concatenating the volumes representing both images, and feeding the resulting volume into several convolutional layers followed by several fully connected layers. The convolution layers use 64 3x3 filters with a stride of 2 and the ReLU activation function. The first three fully connected layers have 4096 neurons each and also use the ReLU activation function. The last fully connected layer has 15 neurons and no activation function. We apply the softmax function to this layer's output to get the class probabilities across the 15 possible classes (discretized speeds).

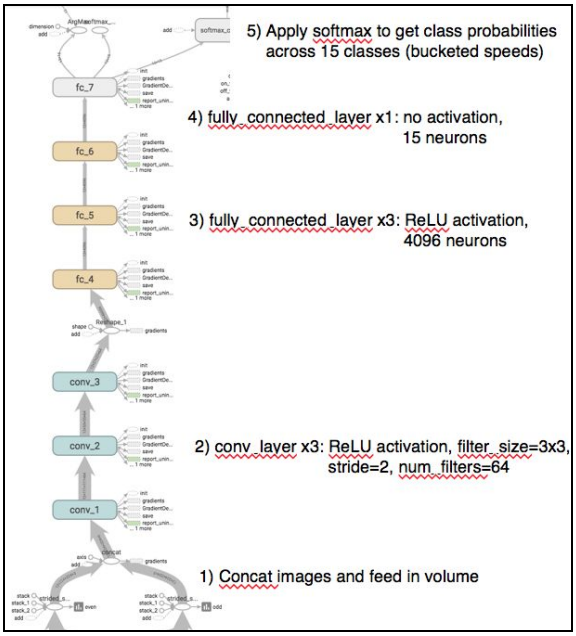

Figure 5. *Our final video to speed CNN architecture.*

V. TRAINING

We used FloydHub, a cloud GPU provider, to train our model quickly. We decided to use the TensorFlow Estimator interface to make checkpointing easy (we wanted to be able to start and stop on Floydhub without losing training progress). We used the cross entropy loss between the true bucket (discretized during preprocessing) and our predicted bucket. We used a gradient descent optimizer with learning rate 0.001. Larger learning rates caused divergence.

We trained for over 9,000 batches on Floydhub (about 15 epochs). We used TensorBoard to monitor the progress and output loss/accuracy graphs (included below). We also checkpointed the model every 60 seconds. We attempted to train locally, but doing so took on the order of seconds per batch, which was unacceptably slow.

Figure 6. *Plot of training set loss as a function of training iterations.*

Figure 7. *Plot of training set accuracy as a function of training iterations.*

As expected with training on mini-batches, our accuracy on each batch varies but tends upward. By the end of the 15 epochs, we were able to completely fit the train set. We then evaluated on our unseen test set. Our trained model achieved 91% accuracy on the test set. Given the varied nature of roads in the train/test sets, we feel confident the network would generalize further to unseen road environments. We did not get to try retraining with regularization (such as by using dropout); in our next iteration we would like to try regularizing the network.

VI. INTERPRETING RESULTS

To better understand our results, we took the gradient of the loss with respect to each pixel of an input image. We then visualized the gradients by coloring the pixels by the magnitude of the norm of their gradients (brighter here means a larger gradient). In other words, we are left with a saliency map of which pixels the network focuses on to make its decision.

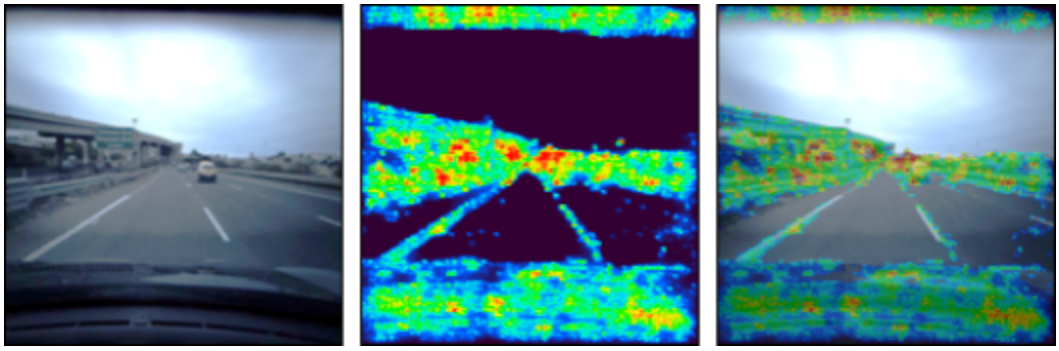

Figure 8. *Gradient of each pixel with respect to the network prediction to highlight where the network is looking.*

As you can see, the network learned to ignore the road and the sky. It learned to focus on the white lines in the road (something we hypothesized it would). Surprisingly, it also focuses on the hood of the car, the dashboard, and the top of the windshield. After investigating this phenomenon more, we believe the car itself shakes more at higher speeds and thus the network has learned to factor in the shaking of the car. How neat!

VII. NEXT STEPS

At the moment we have not analyzed the throughput of the model. We want to confirm the model can make predictions quickly enough so that it can practically be used to output a car's speed in real time. If not, we will modify the model to improve its throughput.

We would also like to apply regularization during training, such as by using dropout, and see how this impacts performance on the training and test sets. Lastly, we would like to test our trained model on footage taken from diverse driving conditions to see how well it generalizes.

VIII. REFERENCES

E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In CVPR, 2017.

Fischer, Philipp, Dosovitskiy, Alexey, Ilg, Eddy, Hausser, Philip, Hazrba, Caner, Golkov, Vladimir, van der ¨ Smagt, Patrick, Cremers, Daniel, and Brox, Thomas. Flownet: Learning optical flow with convolutional neural networks. In ICCV, 2015.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

Websites
github.com/pathak22/pyflow
comma.ai