Mars Challenge Report

Attempt 0: trying an online approach

It is usually a good idea to do some surveys of existing works before designing and implementing some solutions for a problem. There is indeed a comma.ai challenge which is quite similar to this challenge: https://github.com/commaai/speedchallenge. Looking through the proposed solutions for this challenge, there was one solution that caught my attention since the author reported a <1 MSE: https://github.com/onesamblack/deep-velocity-estimation. I have tried and the author's code actually produced a <1% MSE for the Mars challenge also.

Unfortunately (or fortunately?), the author seems to have made a mistake in loading data for training:

```
if config.shuffle_dataset:
  np.random.seed(config.random_seed)
  np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]
```

Basically, the author shuffle the indices before splitting them into train & val groups, then for each indices he loaded several consecutive data as input → there were overlaps between train & val groups. This means the model will have access to validation data during training phase. To test my analysis, I simply updated the above codes to:

```
train_indices, val_indices = indices[split:], indices[:split]
if config.shuffle_dataset:
  np.random.seed(config.random_seed)
  np.random.shuffle(train_indices)
```

The updated codes shuffle train indices after splitting the indices; as a result, during training MSE got stuck at a very high value

Attempts 1: initial design

Based on the surveys of works done on solutions for comma.ai challenge, these seem to be the key findings:

- Training directly on raw frames seems to be difficult. This may be because of lack of training data, and the tendency of model to rely on minute details of frames to estimate the speed.
- Training using optical flows generated from the frames seem to be quite effective. This can give a good starting point for me to approach this Mars challenge.

By checking more than 10 solutions available online, there are 3 solutions with reported MSE results less than 5% (below results are for comma.ai challenge):

- One solution reported 1.089 MSE: https://github.com/satyenrajpal/speed-prediction-challenge. However, this solution split data into 90% training and 10% validation, which seems dangerous as the model may overfit quite easily.
- Another solution reported 3.1 MSE: https://github.com/experiencor/speed-prediction. They used 3D CNN with a multiple frames look back window as input.
- Another approach in https://github.com/laavanyebahl/speed-estimation-of-car-with-optical-flow, which reported MSE of <1 using optical flows, combination of last 4 optical flows + original frame as input, and a quite simple model with convolution and dense layers.

After quickly checking out the above solutions, I design my initial approach as follows:

- Fully separate the training & validation sets of frames (70% training + 30% validation)
- Simple model with convolution, activation (ELU) and dense layers since the amount of provided data is not very large.
- Support multi-GPU training (since I currently have access to multiple GPUs)
- Using optical flows as inputs

To generate the optical flows, I used an approach called Global Motion Aggregation (GMA) in: https://arxiv.org/pdf/2104.02409.pdf. This approach has a 2nd best result for KITTI 2015 (test) on paperswithcode.com (the approach with the best result did not have their source codes open) when I checked. I used their GMA KITTY checkpoint to generate the optical flows.

I also ran some training using the average of more than one optical flow as input to the model; however, this performed worse than simply using the last optical flow as input. This is probably because the further we go back in time, the less correlation we have between the past optical flow and the current speed → taking average of more than one optical flow would decrease the accuracy.

Using GMA to generate optical flows, and using a combination of 1 optical flow + 1 frame (10:1 ratio) at a time as input to the model and training for 500 epochs, I achieved a validation MSE = **0.8714839434090844**

Even though I already reached this validation MSE < 1 after some investigations and trying, I still wanted to try several methods to improve the accuracy more.

Attempts 2: reduced learning rate, optical flow only, random flip

Since the model seems to over-fit too quickly, I tested using reduced learning rate (LR) reduced from 1e-4 → 1e-5.

Also, to test if the combination of 1 optical flow + 1 frame is indeed good, I also tested using only optical flow.

Furthermore, I included a test using horizontal flip to check its effectiveness.

Approach	Validation MSE
LR 1e-4 → 1e-5, optical flow + frame	0.8486217868766841
LR 1e-4 → 1e-5, optical flow only	0.9040022840992944
LR 1e-4 → 1e-5, optical flow only, random horizontal flip	0.9022695297932556

From the above 2 results, I got a new best MSE of **0.8486217868766841**. It is also a little interesting that random horizontal flip does not improve validation MSE; maybe this is because of low amount of data → random flipping may not provide generalization improvement for validation set.

Attempt 3: adding depth

Instead of simple averaging, another way to use multiple flows is to 'stack' multiple frames to create an additional 'depth' dimension of the input, and change the model to 3D CNN.

Approach	Validation MSE
Depth = 2 (2 pairs of optical flows + frames), 3D CNN	1.032505099732294

Attempt 4: reducing speed estimation between 2 frames

As the vehicle's speed cannot change too much between frames, I tried adding an additional loss value from the difference between 2 speed predictions from attempt 3. I tested using a fixed threshold, which means if the 2 speed predictions' difference is smaller than the threshold then the additional loss value = 0. Otherwise, the bigger the difference, the higher the loss, which is added to the original MSE loss for training.

- In the provided labels.txt, around 97% of consecutive speeds have differences of < 0.05, and around 85% have differences of < 0.02.

Approach	Validation MSE
Depth = 2 (2 pairs of optical flows + frames), 3D CNN, added diff loss with threshold = 0.05	0.9370202121973827
Depth = 2 (2 pairs of optical flows + frames), 3D CNN, added diff loss with threshold = 0.02	0.9579420340487409

Even though adding additional diff loss improved the accuracy of the 3D CNN model from attempt 3, it is still not as good as using a single optical flow and 2D CNN.

Attempts 5: BatchNorm & CELU

To further improve the accuracy of the model, I tried adding a Batch Normalization layer and using Continuous Differentiable Exponential Linear Units (CELU) instead of ELU.

Approach	**Validation MSE**
LR 1e-5, single optical flow + image, BatchNorm	0.7340982287664188
LR 1e-5, single optical flow + image, BatchNorm, CELU (instead of ELU)	0.7339890647723

So my best attempt is using a relatively simple 2D CNN with Convolution, BatchNorm, ELU, and Dense layers with optical flow + image (ratio 10:1) as inputs, and 1e-5 learning rate. This attempt got **0.7339890647723** validation MSE with a 30% validation set.

Some other (unsuccessful) attempts

Depth estimation: I have tried using monocular depth estimation for speed estimation, but the accuracy is significantly worse than using optical flow. This is probably because for speed estimation, detecting movements of objects inside frames is necessary and optical flow is a more direct way of doing this.

Brightness augmentation: since augmenting images then generating optical flows during data loading would create a significant amount of overhead; I have tried augmenting images with brightness values of [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5] then generating optical flows before training (brightness value of 1.0 means the original image). During training, the model would take an image with a brightness value randomly chosen from the aforementioned 11 values; however, this did not provide validation accuracy improvement. This can be because of the lack of data, so there is not enough variation in validation data to show improvements when using brightness (and flip) augmentations.

Residual blocks: I have also tried using residual blocks, similar to inside ResNet architecture, in my model but the validation accuracy decreased. This may be because of the lack of data, so a more complicated architecture with higher amount of parameters may overfit the data more easily.

Blocking/smoothening moving objects/pixels: I also considered another approach to further improve the accuracy by dealing with objects that are moving inside the frames. Since optical flows analyze the movement of pixels between frames, optical flows from moving objects may make it harder for the model to estimate the vehicle speed. However, simply setting some objects to a fixed color (e.g. black or white) did not work. I also tried ‘smoothening’ pixels that are moving either too fast or too slow relatively to others, but this also did not work. Cropping images to avoid some moving objects (e.g cars) inside frames did not work either. The reason why ‘smoothening’ or cropping did not work may be the lack of data, as these approaches sacrifice the variation in the data making it even harder for the model to generalize.