Machine Learning 2 Individual Report

Zeqiu Zhang

Introduction

Distracted driving is one of the main reasons for car accidents. According to CDC's data, it causes about 425,000 people injured and 3,000 people killed every year. In order to improve these alarming statistics, and better insure drivers, by testing whether dashboard cameras can automatically detect drivers engaging in distracted behaviors. This project aims at predicting drivers' 10 possible distraction movements based on the train dataset using Computer Vision models.

In this report, we will first describe the dataset and details of models we used. Then we will show how we set up the experiments like how to implement the network. Afterward, the results will be presented and interpreted. Lastly conclusions will be drawn and we will show possible improvements in the future.

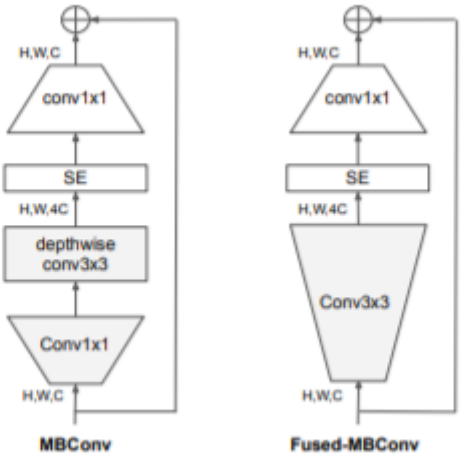
Individual Work

In this project, I did research of EfficientNetV2, tuning batch size and finding epoch numbers. Besides, I wrote code for kfold and ensemble part, read data and process data part, if else choice of using different split method. For the final models, I ran EfficientNetV2B3 while Zhuohan ran EfficientNetV2B2.

I did research on EfficientV2. EfficientNetV2 is a different version of EfficientNet. Instead of scaling all the stages equally, it adapted a non-uniform scaling strategy to get the architecture. And the architecture of EfficientNetV2 looks like this:

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Here are the architectures of MBConv and Fused-MBConv.



The SE module in this picture is a module of attention mechanism, which attaches a weight to the feature map and gets adaptive average pooling.

For batch size, I tried different common ones like 8, 16, 32 and compared them to decide which one to use for this project. Because the data size is large, considering GPU memory I didn't try batch size larger than 32. At last I used 16. Unlike other people in this competition who read all the data into CPU and then feed them into GPU, we read data to CPU using batch too which saves a lot of time for reading data and won't require a large size of CPU.

Then, we chose two kinds of splitting methods: regular train-test split and Kfold split. And I revised this part of code and wrote into if else form so we can choose the method of split. The regular split just splits data into one train & val group, we can use this group to test our pretrained model efficiently. The Kfold split will split the dataset into at least two train & val groups, so they allow us to do cross validation and leverage the possible overfitting issue. The split ratio for regular split is 0.2, but for Kfold, this will change depending on the number of folds we split (when fold is 5, ratio is 0.2).

After getting all the models, I ensemble them together and make prediction based on the average prediction on test data and got the final submission file.

Results

First we start with a relatively smaller size model, ResNet50, and use it as the baseline to tuning parameters like batch size and learning rate, which could save some time and computation power. There is no obvious difference between batch 8 and batch 16, while they are better than batch 32. The reason why large batch size doesn't perform well might be because of the limit of GPU memory. For efficiency, we chose batch 16 for experiments afterward.

Due to the large amount of test data, we also considered using data augmentation to increase the number of our training samples. In order to create images that are similar with test data, we used random contrast, random brightness, and random crop to simulate different drivers' images. However, even if we only augment 10% of training data with small random parameters and combine them with original train data, our

validation accuracy still drops significantly. Therefore, we canceled the augmentation part.
InceptionV3 showed a nice score at first but its validation accuracy is low and private score might also go down after more tries, so we discard that model thinking InceptionV3 is unstable. ResNet152V2 is a model many other participants in the competition used but didn't perform well according to the following table result. Likewise, DenseNet also didn't perform very well, the reason of which might be that it didn't have sophisticated architectures like MBConv in EfficientNet which could get desired performance with less parameters. However, we expected EfficientNetV2M and EfficientNetV2L to get higher scores because they are of larger size but they didn't perform better. The reason might be that they need larger image sizes to show their edges. However, model ensemble requires the image size to be the same, so we didn't change it for them. Besides, it is really time consuming to train that V2L model.
So we finally settled down for EfficientNetV2B2 and EfficientNetV2B3.They gave us good performance and efficient convergence speed. We got better results after using Kfold Ensemble on them individually, so we decided to ensemble the 10 models together (5 EfficientNetV2B2 folds and 5 EfficientNetV2B3 folds) to get the final result and it's the highest score we got. This score in the competition could get the 69th position, so there should be something more sophisticated to do for improvements.

Pretrained Model	Val_accuacy	Private Score
Densenet121	0.9969	0.32719
	0.9967	0.36186
Densenet169	0.9971	0.34589
Densenet201	0.9967	0.36915
InceptionV3	0.9946	0.28210
	0.9958	0.37053
	0.9943	0.39464
Resnet152V2	0.9962	0.59611
EfficientNetB2	0.9969	0.34956
EfficientNetV2B2	0.9967	0.31996
	0.9969	0.29559
	0.9962	0.35109
	Kfold Ensemble	0.20779
EfficientNetV2B3	0.9965	0.30356
	0.9966	0.30581
	Kfold Ensemble	0.22650
EfficientNetV2M	0.9958	0.32306
EfficientNetV2L	0.9944	0.38809
Model Ensemble		**0.20481**

Summary and conclusions

In this project, I learned the following stuff:
1. combine kfold and minibatch together
2. ensemble models together to get average prediction which could get better results than any single one
3. If using Kfold, remember to add .shuffle before using batches to feed to GPU, in our case, the first batch will be all images from c0 if not shuffled, which caused really bad results at first when we used Kfold.

In conclusion, EfficientNetV2 performed the best among all the models we tried. Kfold and ensembling different models could further improve the performance better than any single model. This project also told us that it is necessary to read data into a CPU using batches could save time for loading and save the memory of a CPU. Data augmentation in our case didn't help to increase the performance, which is also a surprise. And the error analysis also might help us to improve the model, if finding some similarities between images we didn't predict correctly. And for future improvement, we should consider some more sophisticated augmentation like Mixup or CutMix which we didn't use could help to raise the score. Another improvement that might have an effect will be using larger image sizes for running EfficientNetV2M and EfficientNetV2L. If they show better performance, ensemble them together could get a higher score.

Besides, in this dataset, one driver could have a lot of images and these images are the image of the driver in a different timelines, which could form a complete gif. If that information could feed to the model, maybe people can get better results. That model may need a recurrent architecture to capture the sequence information because the previous input would have an influence on the output.

Percentage of code found from Internet is about 70/430 (roughly counting).

References

https://www.kaggle.com/c/state-farm-distracted-driver-detection/data Kaggle: State Farm Distracted Driver Detection

Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning* (pp. 6105-6114). PMLR.

Tan, M., & Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*.

State Farm distracted driver detection. Kaggle. (n.d.). Retrieved December 6, 2021, from https://www.kaggle.com/c/state-farm-distracted-driver-detection/discussion/22631.

State Farm distracted driver detection. Kaggle. (n.d.). Retrieved December 6, 2021, from https://www.kaggle.com/c/state-farm-distracted-driver-detection/discussion/22906.

State Farm distracted driver detection. Kaggle. (n.d.). Retrieved December 6, 2021, from https://www.kaggle.com/jiaodong/vgg-16-pretrained-loss-0-23800.