Distracted Driver Detection Report

Jie Gu(jg6617) Fan Zhang(fz2068)

INTRODUCTION

Driving a car is a complex task, and it requires complete attention. Distracted driving is any activity that takes away the driver's attention from the road including texting, taking on the phone, operating the radio or reaching behind. Several studies have identified three main types of distraction: visual distractions (driver's eyes off the road), manual distractions (driver's hands off the wheel) and cognitive distractions (driver's mind off the driving task). It is reported from The National Highway Traffic Safety Administration (NHTSA) that 36,750(37 thousand) people died in motor vehicle crashes in 2018, and 20% of it was due to distracted driving. More importantly it is possible that such accidents could be prevented if some alerting device is used. It is also reported that many states now have laws against texting, and other distractions while driving. If an algorithm which could detect distracted driving behavior is designed, a timely alert could be sent to prevent dangerous driving behavior and to save lives.

DATA DESCRIPTION

The whole training dataset contains 17263 images categorized in 10 classes from State Farm[1]. The ten classes include safe driving, eating or drinking, hair and makeup, reaching behind and talking to passengers, texting with right or left hand, talking on mobile phones with right or left hand and adjusting radio. Examples for each class are shown in fig 1. These images are actually screenshots from different driving videos. The videos are taken from 31 participants driving different types of cars during different times of a day. For train-test split details, please refer to the challenges section.

CHALLENGES

The first problem is how to split the dataset. If we do the random train test split, there will be data league since the photos are consecutive screenshots from a video, there is a high likelihood that highly similar images for the same driver will show multiple times in the dataset. If these images go to both train and test, the accuracy will be surprisingly high for the test result, which is not desirable. In this case, we split the train and test based on driver id. The images for one driver will either go to the train or the test, but not both. By this we could get the split right. The second challenge is the limited training data size. It is known that building a good machine learning model requires a lot of training data to capture robust representation of unknown input distribution. In this project we have only no more than 20 thousand images. Therefore, it is difficult to produce a model that could become generic. However, we could leverage knowledge learnt from related tasks for this detection. By which it refers to transfer learning. Through transfer learning, trained weights in the source network could capture a representation of the input that can be transferred by fine-tuning the weights or retraining the final dense layer of the network on the new task. To better improve the transferability, proper source model and the degree of finetuning should be determined. The last challenge is image class confusion. There are many images in the dataset that are almost similar while representing quite different behavior. In such cases, we applied image augmentation techniques to get multiple looks at the same image through different angles, which gives us more opportunity to catch what the driver is exactly doing even though the difference is subtle. Furthermore, since the training set had no more than 20 thousand images, it is preferable to get more images from the training set to avoid overfitting. Image Augmentation creates more images from the original by performing random actions such as rotation, shifting of width and zoom.

APPROACHES

Convolutional neural networks have shown impressive progress in recent years in various tasks such as image classification and object detection. For this project, we first tried some CNN models with batch normalization and dropout. Since the accuracy is not promising on both train and test dataset, we increased training dataset by image augmentation, which only leads to little improvement on the test accuracy. After realizing the limitation of the small size of the training dataset, we decide to apply transfer learning to do the feature extraction. Firstly, we choose VGG as our pretrained model. It is a deep and simple network which mainly uses convolutional techniques along with zero-padding, dropout, max-pooling and flattening. We use the pre-trained ImageNet model weights for initialization and then fine tune all the layers of the network. Initial layers act as feature extractor and the last layer is SoftMax classifier which classifies the images into one of the ten categories. To maximize the benefit the model could get from transfer learning, we added a few extra layers before the SoftMax layer to further help the model adapt to our use case, which includes global average pooling layer, dropout layer, dense layer, etc. to

further improve the model accuracy, we tried different experiments on trainable layers to decide which layer should be frozen and which should be fine-tuned. Resnet50 is the second architecture we tried. It is an extension of the VGG16 model with 50 layers. To counter the issue of difficulty in training a deeper network, feedforward neural networks with "shortcut connections" with reference to the layer inputs have been introduced. While RESNET was created with the intention of getting deeper networks, Xception was created for getting wider networks by introducing depthwise separable convolutions. By decomposing a standard convolution layer into depthwise and pointwise convolutions, the number of computations reduces significantly [2]. It turns out that Xception gives the best performance. Among all the architectures, we tried various optimizers including sgd and adam. On the model which gives the best accuracy, we compare the running time and cost on different GPUs.

BEST MODEL: XCEPTION

Just as we mentioned, the best model for our problem is Xception. Thus, in this section, we will briefly introduce the convolutional neural network architecture Xception, which is based entirely on depthwise separable convolution layers. It relies on prior work on convolutional neural networks, especially vgg16, the Inception architecture and depthwise separable convolutions. The Xception architecture has 36 convolutional layers, and they are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules (Franc¸ois Chollet, 2017). Then, these modules are further divided into 3 flows: 4 in entry flows, 8 in middle flows, and 2 in exit flow. The data just goes through these flows in order.

However, simply applying the Xception model on our dataset will result in overfitting. On one hand, the Xception model has 36 convolutional layers; on the other hand, we do not have much training data, more precisely, only around 20,000 photos in our training dataset. To avoid the problem of overfitting, we use the following two strategies. Firstly, we use a pretrained xception model on imagenet. After that, data augmentation techniques like rotation, brightness, and cutout are applied to our training data. Additionally, we also choose the P100 GPU for our model. More details will be provided in the evaluation section.

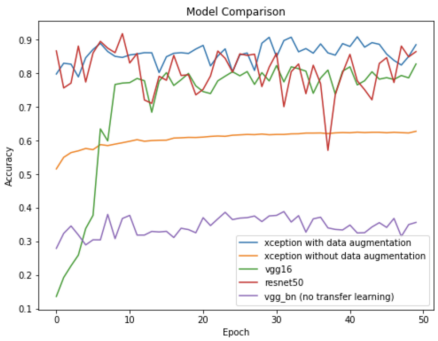
OUTPUT DEMO

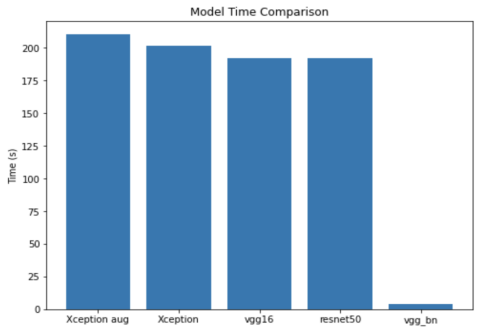
For a complete deep learning project, only showing the evaluation metrics like accuracy is not enough. Thus, for the purpose of better visualization, we also show a single input and its corresponding output demo as below.

	Label	Probability
0	safe driving	4.055822e-09
1	texting - right	2.828490e-08
2	talking on the phone - right	9.999999e-01
3	texting - left	6.249179e-09
4	talking on the phone - left	2.436502e-08
5	operating the radio	2.453163e-09
6	drinking	2.665410e-08
7	reaching behind	2.400815e-08
8	hair and makeup	8.925212e-09
9	talking to passenger	9.506997e-10

EVALUATION

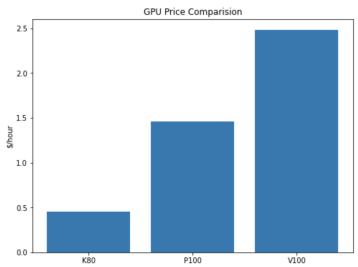
First of all, we make a comparison on the accuracies and training time of different neural network models on our data. The left plot below shows the validation accuracy as a function of the number of epochs for the models we tried, and the right plot is about the average training time on our data per epoch for these models.

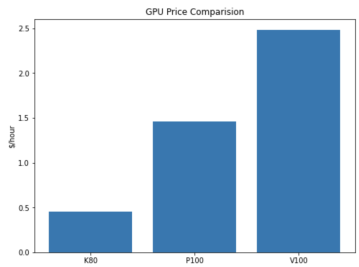




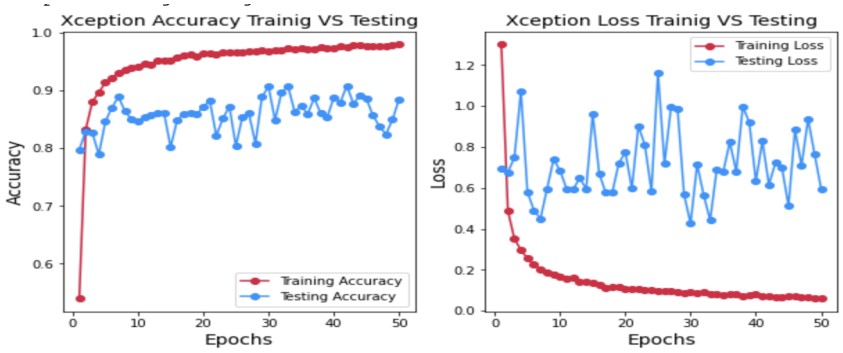
In the left plot, the vggbn is the only model without transfer learning here. We can see the performance is really bad here, that is, the accuracy is just around 30 percent here. We also try other models without transfer learning, but they are actually no better than this model. So, this shows that transfer learning is of great significance for our data. From the line representing vgg16 here, we can see the accuracy is not good at the beginning, but then it increases quickly. The red line is resnet50, and the top blue one is xception. All of these three models have data augmentation before training. Although they all seem to work well on our data, the Xception is above another two lines at most times. The problem of resnet50 here is that its performance is not stable. More precisely, around 35 epochs, there is a large fluctuation in its line, which may come from overfitting. Another thing needs to be noted is that we also include the Xception without data augmentation (green line in the middle), so we can obviously see the impact of data augmentation here. It improves the accuracy from around 60 percent to 90 percent. On the right part, Vgg_bn runs really fast, and this is because its structure is simple and it is not a transfer learning. Other four models are similar. Although the plot shows that the resnet50 is faster than Xception here, but even for the same model, the runtime will fluctuate. Therefore, we cannot draw the conclusion that Xception is slower. Their training time is actually similar. Based on both accuracy and time, we finally choose the Xception model.

After deciding the model, we make an evaluation on the GPUs for our model in the following plots. On the right, it is obvious that K80 takes about 700 seconds for one epoch in the xception, and both P100 and V100 takes about 200 seconds. But on the right, we can see V100 is more expensive than P100, so we finally choose P100 for our model.





Now, let's see the detailed performance of Xception. For Both accuracy and loss, we can see an obvious improvement on the training set as the number of epoch increases. For the testing data, both the accuracy and loss reach to the best around 30 and 35 epochs. The problem is that the improvement is not that obvious. We also try different optimizers and learning rate schedules, but they do not work here. The reason may be that we do not have enough pictures in our testing data, but on the other hand we have 10 labels in totals. These factors may bring randomness to our prediction, leading to the fluctuation here.



It is also helpful to figure out in which part our model is more likely to make mistakes. So we have accuracy for each label here. For most labels, our model has a reasonable performance, but for the label 'talking to passenger', the accuracy is pretty low. In order to find the reason behind this, we have an example of misclassification for this label on the right. This picture indeed belongs to 'talking to passengers', but our model classifies it as 'safe driving'. Now this low accuracy makes more sense for us because the talking is hard to recognize here.

	Label	Accuracy
0	safe driving	0.832524
1	texting - right	0.935103
2	talking on the phone - right	0.979885
3	texting - left	0.994987
4	talking on the phone - left	0.929293
5	operating the radio	0.947891
6	drinking	0.880102
7	reaching behind	0.991018
8	hair and makeup	0.689759
9	talking to passenger	0.392157

CONCLUSION

In this final section, we will make a conclusion on our project and mention the future work we could do. Generally Speaking, the Pretrained Xception Model on Imagenet is the ideal model for State Farm Distracted Driver Detection Data based on the accuracy and runtime

performance. Although the size of our training data is small, data augmentation can help avoid the problem of overfitting. As for GPUs, P100 is the best choice for our model based on the price and runtime. The shortcoming of our model is that although overall performance of the model is good, it cannot work well on some labels like ‘talking to passengers’. More research can be done on how to distinguish between two similar pictures with different labels. Besides, maybe we can obtain more data similar to our data to enlarge both of training and validation data, or find an Xception Model pretrained on data similar to State Farm Distracted Driver Detection Data, which may brings a higher accuracy.

Reference

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

[2]Forson, E. (2017, June 11) “How I tackled my first kaggle challenge using deep learning”
https://towardsdatascience.com/how-i-tackled-my-first-kaggle-challenge-using-deep-learning-part-1-b0da29e1351

[3]Franc¸ois Chollet (2017) “Xception: Deep Learning with Depthwise Separable Convolutions”
https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf