Distracted Driver Detector

Ke Chen, JiaJun Gu, Yiming Ju, and Sen Wang [*]

Abstract

With the popularization of vehicles and high traffic accident rate, driving safety has gradually been paid attention to by people. Since almost traffic accidents are caused by drivers' distraction, it's necessary to detect these potentially dangerous actions in advance so as to reduce the probability of traffic accidents.

The Kaggle challenge State Farm Distracted Driver Detection aims to detect the behavior of a driver given a photo taken by the dashboard camera. The dataset is classified into 10 classes of dangerous driving behaviors. And our model is based on EfficientNet method. We compared the performance of different pre-trained models and finally chose EfficientNet-b5. Through data preprocessing, driving images are converted into metrics with clear and uniform format which can be read by pytorch. Then different learning rate and epochs are used in the training. To avoid overfitting, k-fold cross validation is used in the training. In order to achieve the highest predicting accuracy on the test dataset, we trained a classifier and take the interaction between human and objects into account. Our model can be used to recognize the behavior of the driver with a relatively high accuracy. The performance of our project ranks 47/1440 in the competition.

1. Introduction

Driving safety is of great importance in our daily life. According to Karnas Law Firm, around 3,000 car accidents occurs each day all over the world. In addition to alcohol and drug use, distracted behaviors of a driver also account for a large proportion of the accidents. Statistics from the CDC motor vehicle safety division reveal that one-fifth of car accidents were caused by distracted drivers.

Motivated by the desire to enhance driving safety, our group decide to participate in the Kaggle challenge *State Farm Distracted Driver Detection* (for detailed information, see `https://www.kaggle.com/competitions/state-farm-distracted-driver-detection/data`). The competition was held by State Farm in 2016, the largest property and casualty insurance company in the United States. The aim of the challenge is to automatically detect the distracted behaviors of drivers with dashboard cameras.

In this competition, we are given individual images, and we need to correctly label the images in the test dataset. Therefore, the problem setting is indeed an action recognition task on still images, and this is generally more challenging than action recognition in videos for lack of motion information. A naive way to deal with this is to train a classifier using complete images, while many literatures focus a lot on the interaction between human and objects so as to concentrate on more important information. In our setting, as we still need to take some actions without objects, such as "talking to passenger" and "safe driving" into consideration, we have to be careful when applying existing methods.

We preprocessed the dataset, selected the appropriate pre-trained model, constructed our own machine learning model, and compared different optimizers. After training and validation, our model can be used to recognize the behavior of the drivers with a relatively high accuracy.

2. Related Work

Most of the literatures we found pay much attention on human-object interactions. Girish et al. [2] provides a solution for recognition of actions that includes people's interaction with objects. They implement this by focusing on the region where human and objects both appear. By training a classifier on 4096-dimensional vectors which represent the components they need, and are achieved by a CNN model trained beforehand, they are able to perform action recognition with the best accuracy of 81.97%.

Yao et al. [5] models human actions by two elements, attributes that depict human actions, and parts that represent objects. In this study, they propose a so-called "dual sparsity reconstruction framework" based on sparse coding and compressed sensing to represent meaningful contents of images. Experiments on the PASCAL action dataset and "Stanford 40 Actions" dataset show that their method has a state-of-art performance.

Delaitre et al. [1] propose discriminate training model

[*]This submission is the project proposal for course ECE4880J in 2022SU. This course is held by UM-SJTU Joint Institute, Shanghai Jiao Tong University. The mentor of this course is Dr. Siheng Chen.

that introduces features representing interaction between human and objects, and discriminatively selects person-objective intersection pairs. Experiments on the Willow-action dataset and the PASCAL VOC 2010 action classification dataset yield the mean average precion of around 62.2%.

Maji et al. [4] work mainly focus on using poselet to solve the problems of recognizing the action of human in those images that only contain part of a person or only have few images to be trained, his work is still meaningful for us. According to the results from Maji, the pose itself alone could not directly indicate the actions, models of learning action's appearance by restricting the training examples of poselet to category is needed. His work indicate that we should also focus more on appearance.

Gkioxari et al. [3] work develop a part-based system that applying the CNN to part of the image to figure out the partial feature of the image. He define the object categories as "person+action" and get nice performance. It alongs with Maji give us inspiration that we could focus more on the object.

Yao et al. [6] detect the actions by combining the discriminative feature mining and randomization to discover image patches. Its gives that using randomization could be a useful method for avoiding overfitting.

Based on the literatures, in addition to the general idea of training a classifier, it is reasonable for us to take into account the human-object interaction in our method.

3. Approach

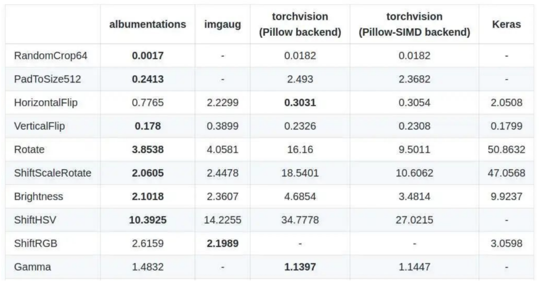
3.1. Preprocess

In the experimental stage, in order to make sure that the performance is not caused by different random number seeds, we first determine a fixed random number seed to ensure that the random numbers generated by the random function in `random`, `numpy`, and `pytorch` are identical each time. We created an `activity_map` dictionary, mapping the possible behaviors of drivers to the corresponding sequence number, and adds two columns to our dataframe, which are:

- `class_num`, which uses int to represent the corresponding class of each image.
- `File_path`, which records the path of each image, so as to facilitate the subsequent reading of each file.

We have defined two classes to realize the main data preprocessing procedure.

- `DataTransform`. This class is used to transform and enhance pictures. In this process, we mainly used `albumentations` package. This package is designed for image preprocessing. As shown in the figure below, it has a much faster speed than other methods, which greatly improves the efficiency of our data preprocessing. We flip, resize, and normalize the images of the training set. And for validation set, flipping and resizing makes no difference on the result, so these two steps are omitted. And finally, we use the ToTensorV2 function to convert the images into an acceptable input format for the pytorch model.



	albumentations	imgaug	torchvision (Pillow backend)	torchvision (Pillow-SIMD backend)	Keras
RandomCrop64	**0.0017**	-	0.0182	0.0182	-
PadToSize512	**0.2413**	-	2.493	2.3682	-
HorizontalFlip	0.7765	2.2299	**0.3031**	0.3054	2.0508
VerticalFlip	**0.178**	0.3899	0.2326	0.2308	0.1799
Rotate	**3.8538**	4.0581	16.16	9.5011	50.8632
ShiftScaleRotate	**2.0605**	2.4478	18.5401	10.6062	47.0568
Brightness	**2.1018**	2.3607	4.6854	3.4814	9.9237
ShiftHSV	**10.3925**	14.2255	34.7778	27.0215	-
ShiftRGB	2.6159	**2.1989**	-	-	3.0598
Gamma	1.4832	-	**1.1397**	1.1447	-

Figure 1. The speed of albumentations vs. other packages

- `Dataset`. This class is used to store the preprocessing results and needs to be used with the `DataTransform` class. By calling `pull_item` function within the class, it will automatically transform and store image information. The images before and after preprocess are shown in the following figures.

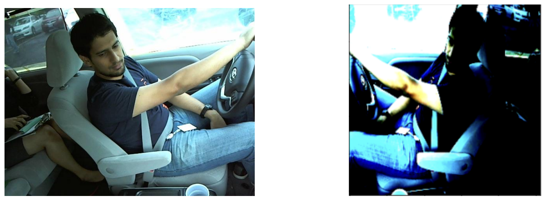


Figure 2. Before and after preprocess

Then we use `torch.utils.data.DataLoader()` to create data-loaders for both training set and validation set to combine a dataset and a sampler, and provide an iterable over the given dataset. After preprocess, we can get iterable and normalized datasets for both training and validation.

3.2. Pre-trained model selection

Transfer learning is a technique that aims to adapt an existing model to be endowed with a different but related function that solves a different problem. It can be used as a starting point in some similar but different tasks and the "child" model can be trained to high accuracy with a much smaller dataset compared to the "parent" model. Since an overall re-training would essentially cost extensive computation resources, fine-tuning re-trains selected layers of the pre-trained model while freezing other layers'parameters. Thus a good pre-trained model matters.

Criteria for selecting pre-trained models:

1. Accuracy.
2. Speed of model training and predictions.

From `EfficientNet`, we compared the performance of `EfficientNet − b2`, `EfficientNet − b3`, `EfficientNet − b4`, and `EfficientNet − b5` with the first 1000 images in the training set. The performance regarding accuracy and training time is as shown in the figure below. We can see that the accuracy of `EfficientNet − b5` is relatively high and the training time is not that large. Actually, the accuracy will be even higher if we use `EfficientNet − b6` or `EfficientNet − b7`, but as trade-off, the training time increases greatly. For `EfficientNet − b5`, its accuracy is already more than 99%. Considering both the accuracy and the training time, we will choose `EfficientNet − b5` as our pre-trained model.

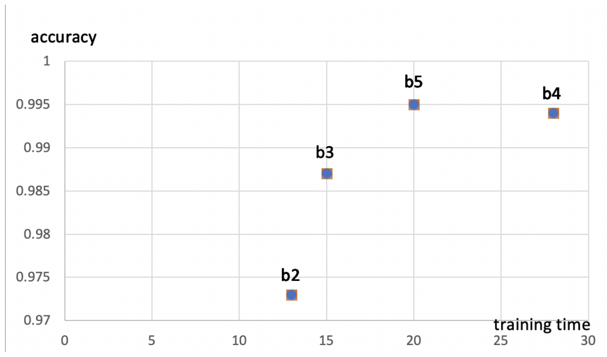


Figure 3. Comparison of different pretrained models

3.3. Train

The training process is shown in the following part, which applies cross validation. The whole dataframe is divided into several folds through the function `sklearn.model_selection.StratifiedKFold`. In each fold the data is preprocessed and then trained. Optimizer can be chosed through `torch.optim`. For each train, if the validation loss is smaller than the previous best one, the current parameters will be saved through the function `save_point` so as to avoid the waste of resources and repeated training.

3.3.1 K-Fold cross validation

In the process of training, the problem of overfitting often occurs, that is, the model can match the training data well, but cannot predict the data outside the training set well. Thus data is often divided into training set and validation set to evaluate the training effect of the model. It divides the original data into K groups (k-fold), takes each subset data as a validation set, and the rest of the K-1 subset data as the training set. In this way, K models can be obtained. The results of these K models were evaluated in the validation set respectively, and the final Error MSE(Mean Squared Error) is added and averaged to obtain the cross-validation error. Cross-validation makes effective use of limited data, and the evaluation results can be as close as possible to the performance of the model on the test set.

We compared the performance of 3-fold and 5-fold cross validation. The final score(the lower, the better) of 3-fold is 0.22057(private score, using 81% of the test data) and 0.24386(public score, using 19% of the test data), while the score of 5-fold is 0.18479(private score) and 0.18528(public score). We can see clearly that the more folds of cross validation we conduct, the better the performance. Theoretically, the model will be more adaptive if we use more folds. However, conducting more folds mains larger training time. Since the time limit on the HPC, we finally used 5-fold as our results.

Algorithm 1: Model Training (k-fold)

```
1  for each fold do
2      preprocess the data
3      for each epoch do
4          for train/validation do
5              for each image do
6                  update weights and loss
7              end
8          end
9          if $best\_val\_loss > epoch\_val\_loss$ then
10             save_checkpoint()
11         else
12     end
13 end
```

3.3.2 Efficient Net

Compared to the common design in ConvNet finding the best layer architecture, Efficient Net doesn't change the predefined architecture in the baseline network. It just adjusts width, depth and resolution through compound scaling rather than adjusting one of them individually since scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models. From Figure 3, we can see the comparison between the scaling of width, depth, resolution respectively and compound scaling. Wider and higher resolution networks tend to be able to capture more fine-grained features and deeper network can capture richer and more complex features.

The compound scaling method uses a coefficient ϕ to uniformly scales network width, depth and resolution.

$$\begin{aligned} depth &: d = \alpha^{\phi} \\ width &: w = \beta^{\phi} \\ resolution &: r = \gamma^{\phi} \\ s.t. \alpha &* \beta^2 * \gamma^2 \approx 2 \\ \alpha \geq 1, &\beta \geq 1, \gamma \geq 1 \end{aligned} \tag{1}$$

where α, β, γ are constant. Since the FLOPS of a convolution operation is proportional to d, w^2, r^2 and with compound scaling it will increase by $(\alpha * \beta^2 * \gamma^2)^{\phi}$, which is about 2^{ϕ}.

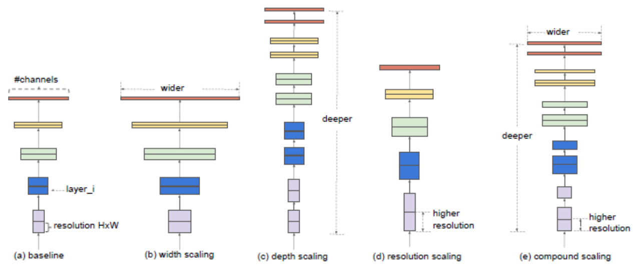


Figure 4. Model Scaling of Efficient Net

3.3.3 Pipeline

Our model is baesd on mobile converted bottleneck Revolution (MBConv module) and Squeeze-and-Excitation Network (SENet module). Firstly, we conducted 1×1 point by point convolution through MBConv, and then transform the dimension of the output channel according to the expansion ratio and perform the k$\times$k dimension depth-wise revolution. After that, the SENet module was used for Squeeze and Excitation. In the squeeze process, the feature image will be compressed through the global average pooling in the direction of the output channel to obtain the global features, so that we can learn the relationship between each channel according to the global feature dimension (D) and the activation ratio (R). Then through sigmoid activation function, the weights of different channels can be obtained, and the final feature image can be calculated. This operation can make the model focus more on the informative channels and pay less attention on those unimportant channels. After that, the channel can be restored to its original dimension through 1×1 point by point convolution. Finally, using the methods of drop connect and skip connection, the training depth can be randomized, so as to reduce the training time.[1]

[1]This method is inspired by the paper "Deep Networks with Stochastic Depth", which makes the model have random depth, shortens the training time, and improves the performance of the model

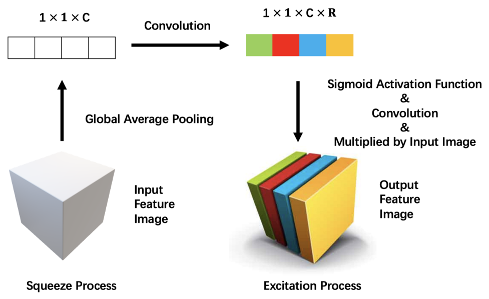


Figure 5. Sketch Map for Squeeze and Excitation

3.4. Test

The test process is shown in Algorithm 2. It applies the model trained in the previous process to predict the class which the test image belongs to. Since we have several models in the training part, each image will be inferred through all these models. Finally the image can obtain the class label through calculating the mean value of these models.

Algorithm 2: Test

```
1 for each fold do
2     for each image in the test fold do
3         obtain the predicted probability
4     end
5 end
6 calculate the mean result
7 obtain the predicted label
```

4. Experiments

4.1. Data

The datasets that the organizer provides to us are enough for our training. It is provided here. The 4.31GB data contains around 20,000 images for training, and around 80,000 images for testing. A preliminary analysis on the training data shows that the images are from 26 unique drivers, and each driver has around 850 images on average. The training data has already been correctly labeled and restored in separate folders. The labels are:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left

- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

There are about 2,000 images in each of the categories above. The images are well-organized, so we only need to load them and convert them to arrays or tensors for future use.

4.2. Experimental details

4.2.1 Model configurations

To apply the pretrained EfficientNet model, python package $efficientnet\_pytorch$ is installed. According to Table 1, with version updates, the size of parameters and the accuracy both increase. Considering our limitations of size, accuracy, and speed, we choose "efficientnet-b5" and set the parameter $num\_classes$ to be 10 classes.

Name	Params	Top-1 Acc.
efficientnet-b0	5.3M	76.3
efficientnet-b1	7.8M	78.8
efficientnet-b2	9.2M	79.8
efficientnet-b3	12M	81.1
efficientnet-b4	19M	82.6
efficientnet-b5	30M	83.3
efficientnet-b6	43M	84.0
efficientnet-b7	66M	84.4

Table 1. Details about the models

4.2.2 Training time

For 3-fold cross validation method, the average training time for each epoch is 321.4630 seconds. For 5-fold cross validation method, the training time is 336.8946 seconds per epoch. For both of them, each fold have 10 epoch, therefore, the training time for each fold is almost the same, which is about 50 minutes. However, the total training time should be:

$$time\ for\ each\ fold \times number\ of\ folds$$

Therefore, the time for 3-fold is about 3 hours and for 5-fold is about 4 hours.

4.2.3 hyperparameters

```
class args:
model_name = EfficientNet-b2/EfficientNet-b3...
input_size = 256
num_classes = 10
batch_size = 32/64
epochs = 10/50/100
folds = 5
lr = 10e-3/10e-4/decay
train = True/False
```

1. model_name: the pretrained model we used
2. input_size: uniform image size in data pre-process
3. num_classes: the number of behavior classes
4. batch_size: the number of training examples utilized in one iteration
   Advantages:
   (a) require less memory
   (b) train faster

   Disadvantage:
   (a) less accurate
5. epochs: the number times that the learning algorithm will work through the entire training dataset
   Too small: not yet fitting
   Too large:overfitting
6. folds: the number of folds that the dataset is divided so as to implement k-folds
7. lr: learning rate
   Too small: train slowly
   Too large: lead to divergence
   learning rate decay: start with a large learning rate then decays it multiple times. Because at the beginning with a large learning rate, we can get a relatively good weight. Then through gradually adjusting the learning rate, the more optimal result will be obtained.

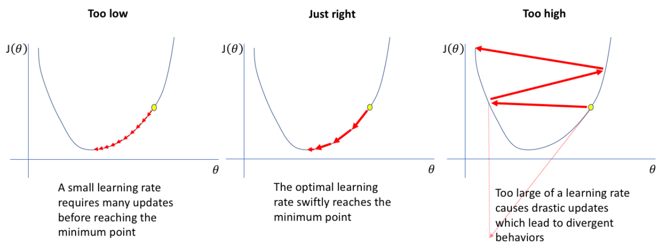

Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Figure 6. Low and high learning rate

8. train: whether the model is used to train or test

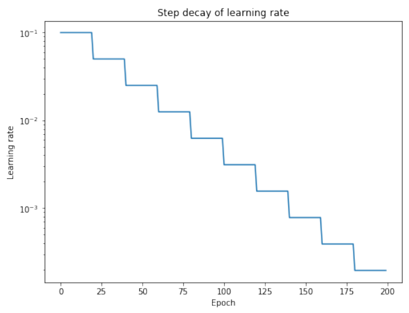


Figure 7. Learning rate decay

4.3. Evaluation method

1. Accuracy:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Number of images in the test set}}$$

   Expected accuracy is greater than 90%.

2. Log Loss:

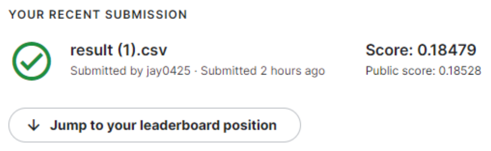
$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij})$$

   where N is the number of images in the test set, M is the number of image class labels, y_{ij} is 1 if prediction is correct and 0 otherwise, p_{ij} is the probability the observation belongs to the correct class. It's the evaluation metrics used in official leaderboard and expected accuracy is lower than 0.15.

3. Model size: Since the model size is not our primary concern, the principle is to reduce the model size without much impact on the first two evaluation metrics.

4.4. Results

As we mentioned before, the host of the competition split the test set into private set and public set. The private set contains 81% of the test images, and the public set contains the rest 19%. Our score is shown in the figure below, where the 'Score: 0.18479' means the private score. Our model ranks 47 out of 1440 submissions, which ranks in top 3.3% among all the competitors.


YOUR RECENT SUBMISSION

result (1).csv
Submitted by jay0425 · Submitted 2 hours ago

Score: 0.18479
Public score: 0.18528

Figure 8. Prediction Result (Logarithmic Loss)

4.5. Ablation studies

The most important hyperparameters are the choice of pre-trained model and the number of the folds. The good choice of pre-trained model can reduce the training time, increase the predicting accuracy, and save the space, so as to save the computational resources. For the number of folds, it is a trade-off between training time and the adaptation of the model. If the number of fold is larger, it can better avoid overfitting, however, it will increase the training time.

5. Conclusion

This paper showed how we improve the performance of efficientnet for this problem. From the previous Table 1, we could see that the accuracy increases along with the size of the parameters. The larger size of parameters would lead to larger use of memory and finally results in slower training speed. Due to our limitations of time and computation power, a trade-off between time and accuracy is unavoidable. Then we conduct experiment on small dataset and determine to use efficientnet-b5 as our model. The models that have larger parameter size than b5 increase few accuracy while it increase the time spent a lot.

References

[1] Vincent Delaitre, Josef Sivic, and Ivan Laptev. Learning person-object interactions for action recognition in still images. In *NIPS*, 2011.

[2] Deeptha Girish, Vineeta Singh, and Anca Ralescu. Understanding action recognition in still images. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1523–1529, 2020.

[3] Georgia Gkioxari, Ross Girshick, and Jitendra Malik. Actions and attributes from wholes and parts. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2470–2478, 2015.

[4] Subhransu Maji, Lubomir Bourdev, and Jitendra Malik. Action recognition from a distributed representation of pose and appearance. In *CVPR 2011*, pages 3177–3184, 2011.

[5] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *2011 International Conference on Computer Vision*, pages 1331–1338, 2011.

[6] Bangpeng Yao, Aditya Khosla, and Li Fei-Fei. Combining randomization and discrimination for fine-grained image categorization. In *CVPR 2011*, pages 1577–1584, 2011.