Final Project Report for CS175, Spring 2021

Project Title: Distracted Driver Detection
Project Number: 36

Team:

Ryo Matsumura, 14537604, rmastum1@uci.edu

1. **Problem**

   According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver in the U.S. In 2018, over 2,800 people were killed and an estimated 400,000 were injured in crashes involving a distracting driver. As we join in the Kaggle project, State Farm Distracted Driver Detection, we use deep learning technology to classify the images of distracted drivers and identify their behavior to improve the alarming statistics.

   This project is to predict each input image provided by State Farm as the test dataset and to give the sets of probabilities being classified in the 10 classes as an output. We use the test data with no metadata such as creation dates. We train our model with the provided train dataset and predict the test dataset. Our final evaluation of this project is to submit the list of likelihood for each test image and

2. **Related Work**

   In the Kaggle project, State Farm Distracted Driver Detection, we have many people attending this project and sharing their codes. The one article of this project we firstly looked at was “Distracted Driver Detection using Deep Learning” by Satya Naren Pachigolla. In his article, he used VGG16, RESNET 50, Xception, and Mobilenet. We have also learned about the solution to Data Leakage, Image Augmentation, Extra layer, and Optimizer. We brought these ideas to our model to improve the accuracy.

   The paper about the Xception, one of the models used in the first article, was “Xception: Deep Learning with Depthwise Separable Convolutions” by Francois Chollet. Since this is the original paper, the author described the concept and the process of fine tuning. We learned the basic Xception model can be improved by choosing fully connected layers and an optimizer with proper parameters. Thus, we decided to choose the Xception model which has a base line and can be optimized for our project.

3. **Data Sets**

   The data sets are provided by State Farm Distracted Driver Detection, Kaggle Project. The images are captured “in a car with a driver doing something in the car”. State Farm stated “ these experiments in a controlled environment - a truck dragging the car around on the streets - so these drivers weren’t really driving.”

   The training dataset consists of 22,424 unique images. These images are categorized into 10 classes, c0 to c9, which are safe driving, talking on the phone, drinking, and etc. Each class has 1900 to 2500 images. There are 10 classes of images with the same person:

Figure 1: 10 Classes of Images from the Train Dataset

Since we only have around 2,000 for each class, and some classes like “talking on the phone” and “hair and makeup” have similar images, Image Augmentation is helpful for our model to avoid overfitting. We added images, for example:

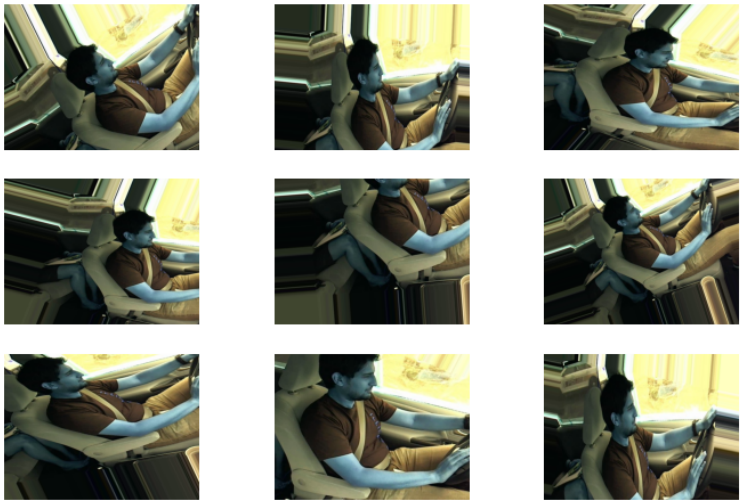

Figure 2: Image Augmentation

In the driver_imgs_list.csv, there are three columns, person, class, and image name. We used a person column to divide train data for cross validation.

subject		classname		img
p021	6%	c0	11%	22424 unique values
p022	5%	c3	10%	
Other (19954)	89%	Other (17589)	78%	
p002		c0		img_44733.jpg
p002		c0		img_72999.jpg

Figure 3: driver_imgs_list.csv

The testing dataset consists of 72,726 unique images without any label. The train and test data are split on the drivers, such that one driver can only appear on either train or test set.

4. Approach

We chose the Xception model as our main model. We used the pre-trained Xception model with ImageNet in Keras.

Xception is a CNN model with depthwise separable convolution. In a normal CNN like VGG16, it uses spatial convolution such as 3x3 convolution for output with 64 channels at a time. Let the previous layer be 3, then we have (3x3x3+1)x64=1794 parameters. On the other hand, Xception uses pointwise convolution and depthwise separable convolution which reduces the number of computations to train. Pointwise convolution consists of 1x1 convolutions with 64 channels which is (1x1x3+1)x64=256, and depthwise separable convolution applies 3x3 convolution for each of 64 channels separately which is (3x3x1+1)x1=10 and 10x64=640 parameters. Overall, the number of parameters is reduced significantly.

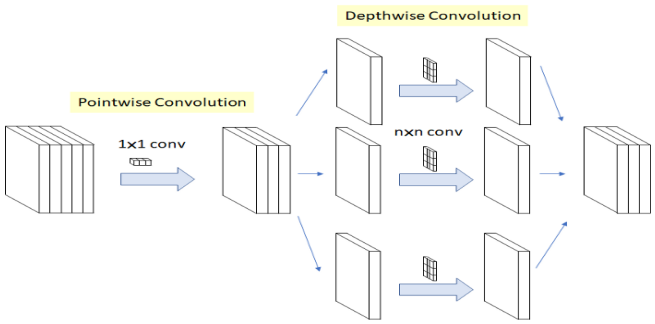

Figure 4: Depthwise Separable Convolution in Xception

The difference between the Inception and Xception is that in the Inception model, 1x1 convolution is followed by a spatial convolution like 3x3 convolution which operates on non-overlapping segments of the output channels. The Xception model, called an extreme version of the Inception model, also operates 1x1 convolution, and then it operates a spatial convolution on each output channel instead of segments of the output channels.

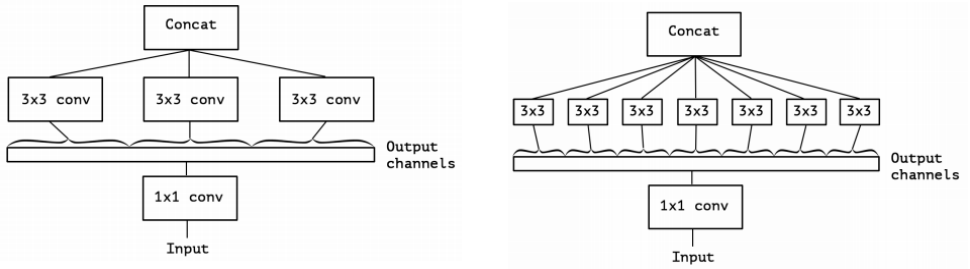

Figure 5: Difference between Inception and Xception in simplified models

This resulted in reducing the number of parameters from the Inception model, and more significantly, it improved the performance on ImageNet and JFT datasets according to the “Xception: Deep Learning with Depthwise Separable Convolution”.

For preprocessing, we resized the train dataset from 640x480 to 299x299 which is the default input size of Xception. Since our train dataset is around 2,000 images per class, we used image augmentation to increase the train dataset size (Figure 2). The functions, ImageDataGenerator() and flow(), will modify the train dataset in each epoch

so that it doesn't consume our memory unlike creating another dataset. We trained our model for 30 epochs with the train dataset, so it could result in a 30 times larger train dataset at most.

Due to the RAM and GPU memory limitation on the kaggle notebook, we faced the run time memory issue. We decided to use the gc function and trained our model separately to solve this problem. Since we first allocated a part of the train dataset in memory and then preprocessed it. The train dataset and validation test dataset were allocated again, and this sometimes exceeds the limitation . Therefore, we used gc to clean and free memory. We also trained our model at separated kernels. We trained it with the first 500 images from each class, a total of 5,000 images, and saved the model weight by using the function, ModelCheckpoint(), which runs when the model improves a validation loss at an epoch. This function allows us to save our best model even though we trained through all epochs and had no improvement at later epochs.

5. Software

We have implemented our Xception model in python with Keras. The main code is written in the kaggle notebook to use GPU for training.

a. For our model implementation, we used the pre-trained Xception model in Keras. We tested different numbers and types of fully connected layers and decided to add three Dense layers followed by Batch normalization and Dropout. At the end, we also added a Dense layer with 10 filters which are our output classes and softmax activation. There is our model diagram:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xcept
ion/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 2s 0us/step
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 299, 299, 3)]     0
_________________________________________________________________
xception (Functional)        (None, 2048)              20861480
_________________________________________________________________
dense (Dense)                (None, 1024)              2098176
_________________________________________________________________
batch_normalization_4 (Batch (None, 1024)              4096
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                32832
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)                256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 23,524,338
Trainable params: 23,466,610
Non-trainable params: 57,728
_________________________________________________________________
```

Figure 6: Our Final Xception Model

There is the list of function I implemented:

Function	Input	Output
Image augmentation	Train dataset	Modified train dataset
Base Xception model and extra layers	Keras core functions	Our Xception model
Extra layers comparison	Different Xception models	Trained model and performance graph
Xception training	Xception model and optimizer	Trained model weight
Sample image augmentation	An original train image	Modified train images
Optimizer comparison	List of optimizer and untrained model	Trained model and performance graph

Table 1: List of My Implementation

b. I used the code from others listed below. Link references are in the Reference section:

Function	Input	Output	Reference
Input function that reads and reshapes the train dataset	driver _image_list. csv and train dataset	List of labeled images	Yashpatel7172
Preprocessing function that divides train dataset for cross validation	List of labeled images	Validation Train and test dataset	Yashpatel7172
Graphical results using matplotlib	model.history	graph	Yashpatel7172
Submission function that saves our prediction	Model and test dataset	submission.cvs	Abdul Mannan

Table 2: List of Others' Implementation

6. Experiments and Evaluation

We divided experiments into additional fully connected layers, optimizers, and final evaluation. For the evaluation, we used cross validation to see the validation loss and accuracy. We randomly picked four people and set them as a validation group. Each person appears in 5% of the total images, so the validation dataset is expected to be 20% of the whole train datasets.

6.1. Fully connected layers

We tested how the fully connected layers would affect the performance of cross validation.

a. We added fully connected layers after the base of the Xception model pre-trained with ImageNet. We had batch normalization and dropout following dense layers with a relu activation. We started testing one dense layer with a softmax activation, and then adding a block of dense layer with batch normalization and dropout(0.5).

b. On the 5,000 train dataset for 10 epochs, the Xception model in Figure 6 had the best performance.

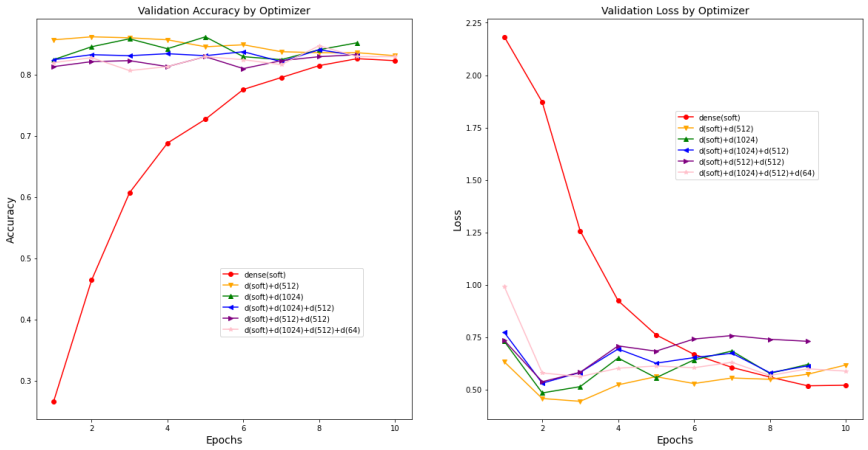

Figure 7: Performance of Xception with Different Extra layers

6.2. Optimizers

We tested which optimizer is suitable for our model. We had SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam.

a. We tested each optimizer on 5,000 train images with 10 epochs. SGD, RMSprop, and Adagrad optimizers used different learning rates.
b. SGD(lr=0.01), Adagrad(lr=0.01) are the two best optimizers in terms of accuracy and loss excluding RMSprop(lr=0.005) because of the huge loss. After tuning parameters for both SGD and Adagrad, SGD(lr=0.01) had the most stable improvements over the training.

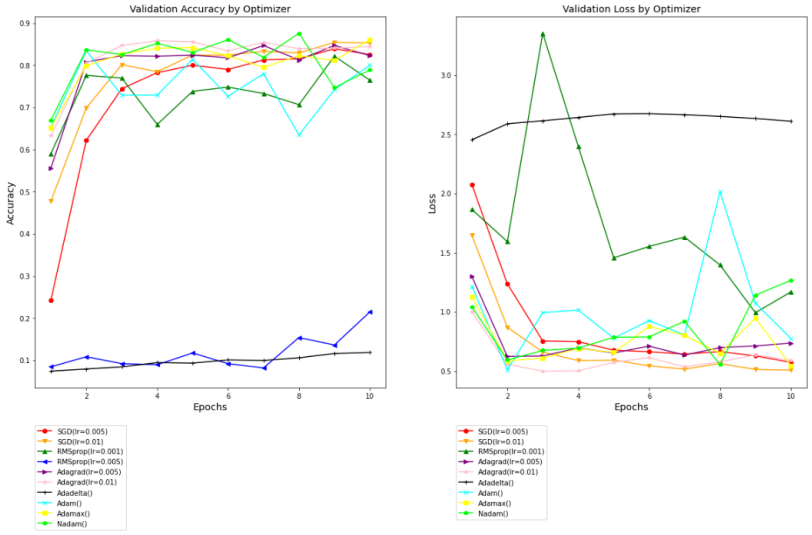

Figure 8: Performance of Xception with Each Optimizer

6.3. Final Evaluation

The final evaluation is to submit our prediction on the test dataset to the State Farm Distracted Driver Detection, Kaggle project.

a. Training our final model used 500 images per class and a total of 5,000 train images. We trained the same model five times separately for all the train dataset by resetting a kernel each time. After training our model, we tested it with the test dataset and predicted the likelihood of being classified into 10 classes. We submitted the result to the kaggle project, State Farm Distracted Driver Detection.
b. As we trained our model, we had seen the improvement, and in the final training, we were close to 98% on validation accuracy.

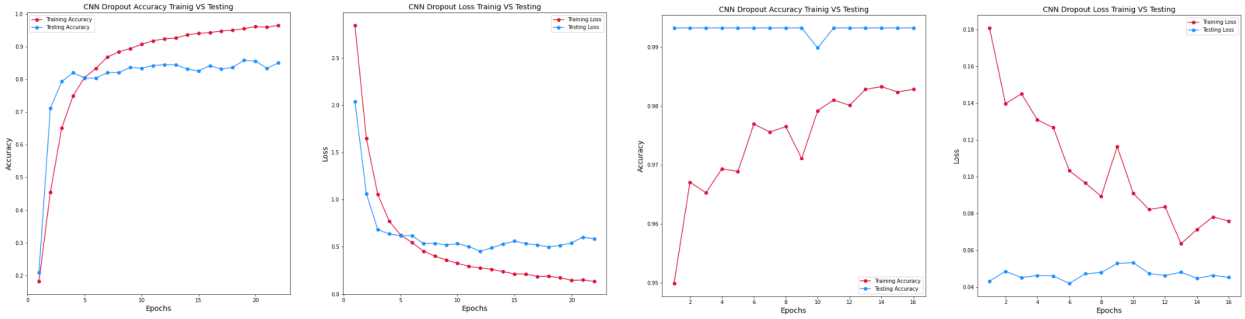

Figure 9: First Training and Fifth Training Results

We submitted our likelihood over the test dataset and got 4.41125 multi-class logarithmic loss score on State Farm Distracted Driver Detection.

7. Conclusion

Our Xception model performed poorly on the kaggle project, State Farm Distracted Driver Detection. This is mainly due to the evaluation problem. When we evaluated our model, we used cross validation. However, cross validation only checks the label and gives true or false. Multi-class logarithmic loss evaluates each probability in the class. Therefore, our model is not optimized well on the evaluation.

For the improvement, we first have to implement a multi-class logarithmic loss function to evaluate our model. Even if we continue to work on fine tuning, we cannot get a good score on the kaggle project. The second suggestion is to organize our code because we haven't worked on the kaggle notebook, and we couldn't manage files and versions well. Our code is quite different between versions. The third suggestion is to find a better way to deal with the memory usage. We should check the memory usage and keep them not exceeding the limit so that we don't have to reset kernels, and it would help overall project progress.

For the future direction, we would like to implement a real time distracted driver detection system using a 360 degree dashboard camera. In recent years, Japan has suffered from road rage, and thus more people use a dashboard camera. This trend occurs in many countries. The more people use a dash cam, the more opportunities we have to implement a real time distracted driver detection system. Since dash cam can give a set of images about the same behavior with extra information such as time. At this time, we only trained and tested images with no metadata. However, if we could use the images from front and the real time information to train, we would have better classification with metadata rather than training with only classes.

8. Reference

- Francois Chollet, “Xception: Deep Learning with Depthwise Separable Convolution”. https://arxiv.org/pdf/1610.02357.pdf

- Satya Naren Pachigolla, “Distracted Driver Detection using Deep Learning”. https://towardsdatascience.com/distracted-driver-detection-using-deep-learning-e893715e02a4

- Sik-Ho Tsang, “Review: Xception - With Depthwise Separable Convolution, Better Than Inception-v3(Image Classification)”. https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

- Keras Documentation. https://faroit.com/keras-docs/1.2.0/

- Yashpatel7172, Xception-Extra Layer. https://www.kaggle.com/yashpatel7172/xception-no-extra-layer

- Abdul Mannan, MobileNet_DLP https://www.kaggle.com/manan12/mobilenet-dlp