Distracted Driver Detection

Project Overview

In this peoject, it gives driver images which were taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc). The goal is to predict the likelihood of what the driver is doing in each picture.

The project is essentially a supervised classification learning problem. The training dataset is already well labeled for classification. The goal of the model is to learn the features of different driving behaviours such that an unseen driving picture can be correctly identified.

Ideas

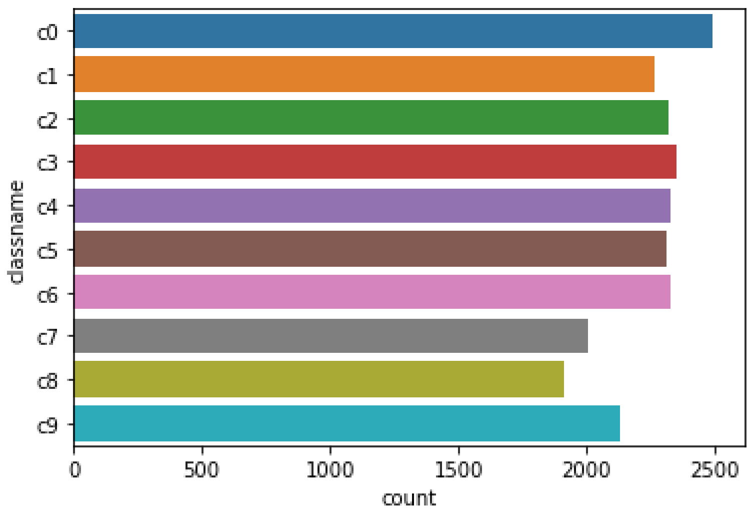
Since the project is a classification task, there are many good, established networks for classification tasks, such as MobileNet, VGG, ResNet and so on. Based on the principles of migration learning, I decided to use the MobileNet2 network here, freezing the underlying neurons and modifying only the top layer to make it fit our project requirements.

Data analysis

The dataset consists of a training set and a test set, all of which are still images taken from the in-car camera footage. The training set has been classified based on the label-classes [c0,...,c9], the meanings of which are: - c0:driving safely; -c1:typing with right hand; -c2:talking on the phone with right hand; -c3:typing with left hand; -c4:talking on the phone with left hand; -c5:tuning the radio; -c6:drinking a drink; -c7:taking the back things; -c8:fixing hair and make-up; -c9:talking to other passengers.

In [33]:
```
sns.countplot(y='classname', data=driver_imgs_list)
```

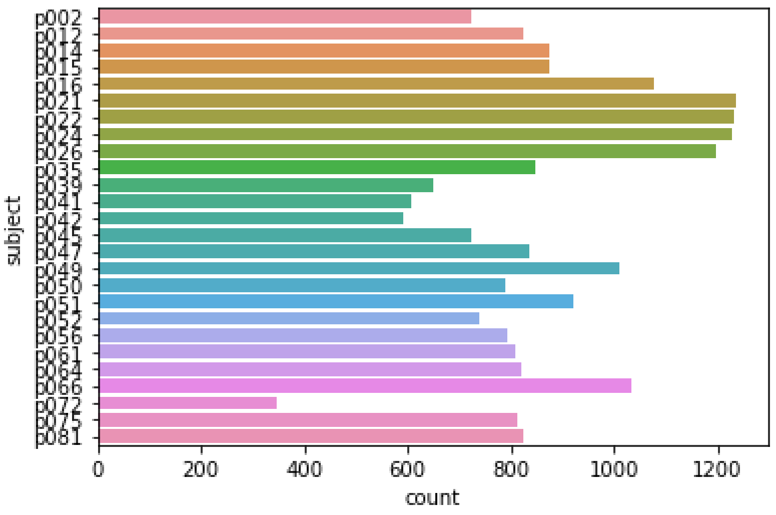
Out[33]:
```
<AxesSubplot:xlabel='count', ylabel='classname'>
```



In [34]:
```
sns.countplot(y='subject', data=driver_imgs_list)
```

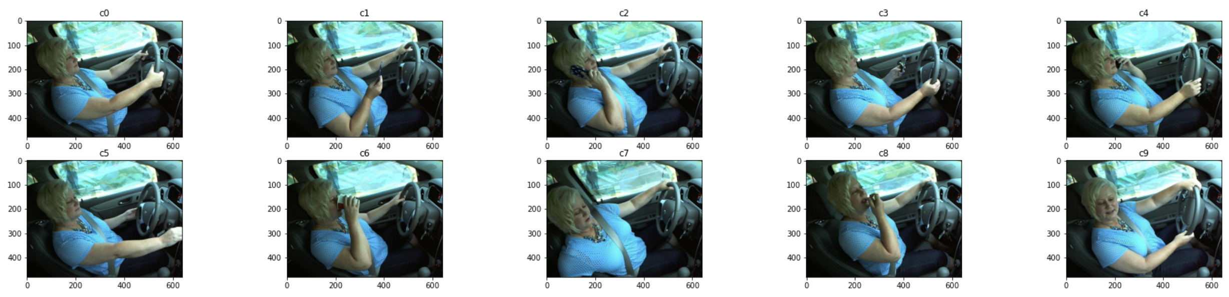
```
<AxesSubplot:xlabel='count', ylabel='subject'>
```

Out[34]:



The images in the training set with the same classification are in the same folder and the distribution of the number of images in each classification is shown in the figure above. The test set was not classified and contained a total of 79,721 2D images. In addition to the dataset, the project also provided a list of driver ids, which contains the driver ids of the images in the dataset and the corresponding classifications. A sample of the randomly extracted images is shown below.

In [35]:



Data preprocess and model training

First, set the size of the picture. Since we only have 30 images per category in our validation set, we set the batch size here to be smaller, just set it to 10. Then Use buffer preceding images from disk to avoid I / O blocking. ref:: https://tensorflow.google.cn/guide/data_performanc

Although the data is pre-processed, it is not enough to feed the model directly and must be augmentationed. Here I used the random flip and random rotation. The data also went through preprocess before entering MobileNetV2.

Then, create the base model from the pre-trained model MobileNet V2. Constructing the model and load pre-trained base model and pre-trained weights. Stacking classification layers on top. The first layer is the global average pooling layer, ref:: https://www.cnblogs.com/hutao722/p/10008581.html.

And then, set the perdiction layer. Apply the tf.keras.layers.Dense layer to convert these features into one prediction per image. Before training the model, it needs to be compiled. And we need to define the input, loss functin,

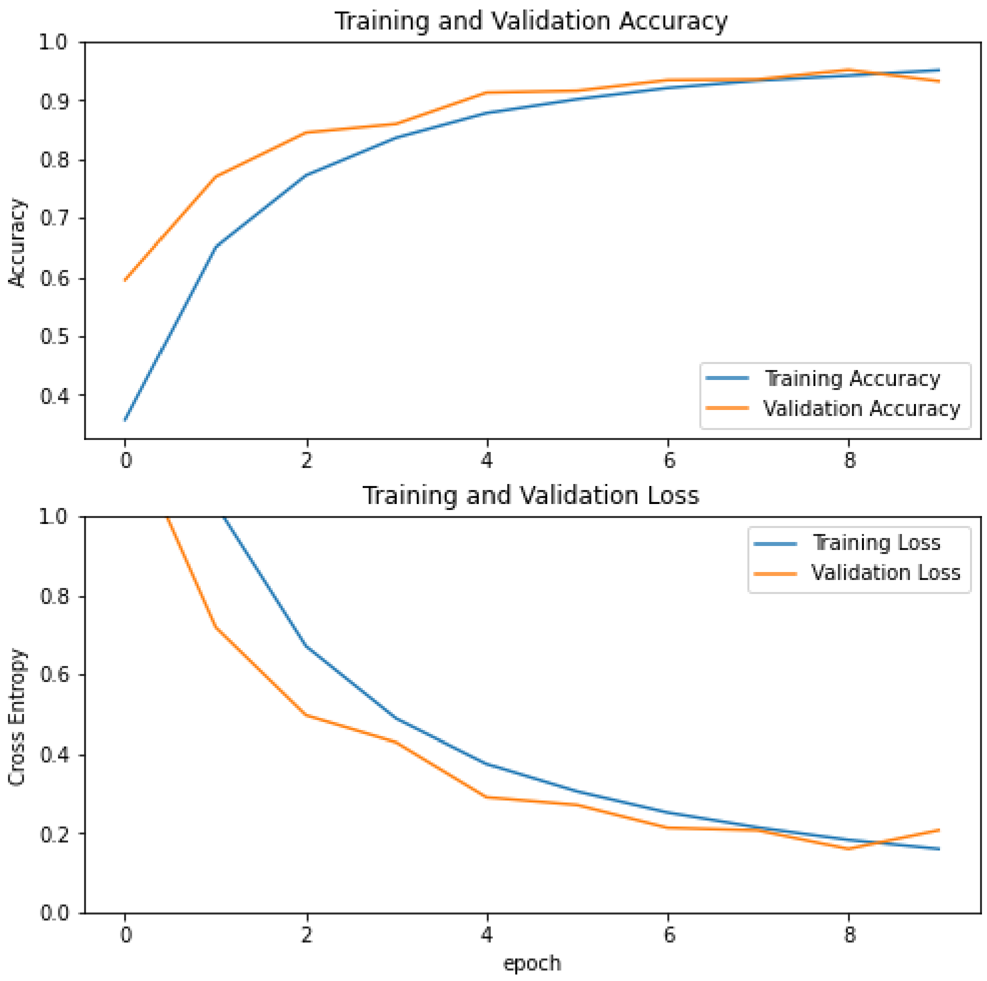
learning rate and optimizer. We also need to freeze all the layers before the fine_tune_at layer, and only train the top 100 of the data.

Finally, start training. After 10 cycles of training, an accuracy of approximately 94% can be seen on the validation set. We can see that the loss and accuracy are both have a normal trend of change. We can also visualise this trend of change.

In [18]:
```
# start training
total_epochs =  10
history = model.fit(train_dataset,
                    epochs=total_epochs,
                    validation_data=valid_dataset)
```

```
Epoch 1/10
692/692 [==============================] - 159s 191ms/step - loss: 2.1417 - accuracy:
0.2429 - val_loss: 1.2367 - val_accuracy: 0.5952
Epoch 2/10
692/692 [==============================] - 69s 99ms/step - loss: 1.1528 - accuracy: 0.
6101 - val_loss: 0.7191 - val_accuracy: 0.7699
Epoch 3/10
692/692 [==============================] - 67s 96ms/step - loss: 0.7306 - accuracy: 0.
7516 - val_loss: 0.4968 - val_accuracy: 0.8448
Epoch 4/10
692/692 [==============================] - 68s 97ms/step - loss: 0.5136 - accuracy: 0.
8281 - val_loss: 0.4287 - val_accuracy: 0.8597
Epoch 5/10
692/692 [==============================] - 81s 116ms/step - loss: 0.3947 - accuracy:
0.8701 - val_loss: 0.2905 - val_accuracy: 0.9129
Epoch 6/10
692/692 [==============================] - 71s 101ms/step - loss: 0.3180 - accuracy:
0.8981 - val_loss: 0.2710 - val_accuracy: 0.9160
Epoch 7/10
692/692 [==============================] - 66s 95ms/step - loss: 0.2603 - accuracy: 0.
9175 - val_loss: 0.2133 - val_accuracy: 0.9341
Epoch 8/10
692/692 [==============================] - 74s 107ms/step - loss: 0.2263 - accuracy:
0.9297 - val_loss: 0.2068 - val_accuracy: 0.9353
Epoch 9/10
692/692 [==============================] - 66s 95ms/step - loss: 0.1890 - accuracy: 0.
9396 - val_loss: 0.1599 - val_accuracy: 0.9515
Epoch 10/10
692/692 [==============================] - 64s 92ms/step - loss: 0.1639 - accuracy: 0.
9499 - val_loss: 0.2070 - val_accuracy: 0.9324
```

In [20]:



The model that was trained will be stored. This model can now be used to predict whether a driver is distracted or not.

Brief reflection

In this project, I learned that simple CNN models are not capable of handling multi-classification problems with such large databases, it is straightforward to use pre-trained models for training and data testing, and to determine optimization solutions. In this process I read the best examples from the kaggle competition, and the official documents of tensorflor. I also learned how to perform migration learning and fine-tune the model. I also dealt with a number of problems with setting up the environment, such as finding a suitable version for libraries used.