Detection of distracted driver using Convolutional Neural Networks

Thamizhiniyan Pugazhenthi - 200941620

Introduction

Machine learning is the study of computer algorithms that may improve themselves over time by gaining experience and using data. Machine learning algorithms create a model based on training data to make predictions or judgments without having to be explicitly programmed to do so. Like the description, we will build a model to detect the distracted driver with Machine Learning using Convolutional Neural Networks and train the model with the training and validation set before predict them with the test images given.

Dataset description

The dataset had set of training and test images. The training data splitted into ten classes from c0 to c9. The 10 classes to predict are:

```
c0: normal 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
```

Along with the set of images, two csv files were presented to assist our project. One with the details about the name of the images along with the class and another sample csv to show the submission format of the project.

Libraries

The libraries were imported to support our project. The assistance of tensorflow and keras is vital to proceed ahead. With matplot to plot charts and pandas to perform csv read and write operations.

In [43]:

```
import os
from os.path import join
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import pandas as pd
```

Model

The Convolutional Neural Network was constructed with input size of (100,100) with the '3' represents 'rgb' format of the image. With Batch normalization, we can standardize the data in between convolutional layers. Maxpooling is to find out the maximum value from the region covered by filter and the data will be converted to one dimensional array using flatten and dropout will help us to prevent overfitting. The hidden dense layers were added to improve efficiency and with the final dense layer represents output with 10 classes. The optimizer 'adam' was used to compile the model.

In [3]:

```
cnnmodel = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', padding = 'same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', padding = 'same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu', padding = 'same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
    ])
cnnmodel.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
cnnmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 98, 98, 32)        896       
_________________________________________________________________
batch_normalization (BatchNo (None, 98, 98, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 49, 49, 64)        18496     
_________________________________________________________________
batch_normalization_1 (Batch (None, 49, 49, 64)        256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 49, 49, 64)        36928     
_________________________________________________________________
batch_normalization_2 (Batch (None, 49, 49, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856     
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 128)       512       
_________________________________________________________________
flatten (Flatten)            (None, 73728)             0         
_________________________________________________________________
dropout (Dropout)            (None, 73728)             0         
_________________________________________________________________
dense (Dense)                (None, 1024)              75498496  
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800    
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130      
=================================================================
Total params: 76,159,754
Trainable params: 76,159,178
Non-trainable params: 576
_________________________________________________________________
```

Data preprocessing

The working directory was set to acccess the folder contains training images.

In [4]:

```
workingdir = os.path.abspath('')
trainingdirectory = os.path.join(workingdir + '/state-farm-distracted-driver-detection/imgs/train/')
```

Train and validation dataset split

Image generator was built to get access of images from the training folder. The data augmentation was used to generalize the model with horizontal flip, width and heigth shift range and rotation range. To improve the model, the training data were splitted into training and validation data in the ratio of 80:20. From the generator, the training and validation set can be accessed in the batch size of 64.

In [5]:

```
trainingdataimage = ImageDataGenerator(rescale = 1./255, height_shift_range = 0.2,
                                        width_shift_range = 0.2, shear_range = 0.2, rotation_range = 40, zoom_r
ange = 0.2,
                                        fill_mode = 'nearest', horizontal_flip = True, validation_split = 0.2)
trainingset = trainingdataimage.flow_from_directory(trainingdirectory,
                                                      target_size = (100, 100), batch_size = 64,
                                                      class_mode = 'categorical', subset = 'training', sh
uffle = True)
validationset = trainingdataimage.flow_from_directory(trainingdirectory,
                                                      target_size = (100, 100), batch_size = 64,
                                                      class_mode = 'categorical', subset = 'validation',
shuffle = True)
```

```
Found 17943 images belonging to 10 classes.
Found 4481 images belonging to 10 classes.
```

Model fit

The training and validation images made to fit with the CNN model on 60 iterations with steps per iteration will be length of the image generator of training set and length of image generator of validation set was denoted as validation steps.

In [6]:

```
history =  cnnmodel.fit(trainingset, epochs = 100, steps_per_epoch = len(trainingset),
                        validation_data = validationset, verbose = 1, validation_steps = len(validationset))
```

```
Epoch 1/100
281/281 [==============================] - 148s 488ms/step - loss: 2.4414 - accuracy: 0.1635 - val_l
oss: 4.3076 - val_accuracy: 0.1029
Epoch 2/100
281/281 [==============================] - 122s 435ms/step - loss: 1.8286 - accuracy: 0.3173 - val_l
oss: 1.9818 - val_accuracy: 0.2584
Epoch 3/100
281/281 [==============================] - 130s 464ms/step - loss: 1.4419 - accuracy: 0.4667 - val_l
oss: 1.4742 - val_accuracy: 0.4720
Epoch 4/100
281/281 [==============================] - 138s 491ms/step - loss: 1.1670 - accuracy: 0.5744 - val_l
oss: 1.1714 - val_accuracy: 0.5775
Epoch 5/100
281/281 [==============================] - 138s 490ms/step - loss: 0.9647 - accuracy: 0.6579 - val_l
oss: 1.0323 - val_accuracy: 0.6523
Epoch 6/100
281/281 [==============================] - 141s 502ms/step - loss: 0.7974 - accuracy: 0.7266 - val_l
oss: 1.0666 - val_accuracy: 0.6746
Epoch 7/100
281/281 [==============================] - 139s 493ms/step - loss: 0.6497 - accuracy: 0.7790 - val_l
oss: 1.3936 - val_accuracy: 0.5791
Epoch 8/100
281/281 [==============================] - 137s 486ms/step - loss: 0.5769 - accuracy: 0.8069 - val_l
oss: 0.9682 - val_accuracy: 0.6954
Epoch 9/100
281/281 [==============================] - 151s 538ms/step - loss: 0.5109 - accuracy: 0.8285 - val_l
oss: 0.9519 - val_accuracy: 0.7148
Epoch 10/100
281/281 [==============================] - 137s 487ms/step - loss: 0.4517 - accuracy: 0.8492 - val_l
oss: 0.8535 - val_accuracy: 0.7626
Epoch 11/100
281/281 [==============================] - 133s 473ms/step - loss: 0.4151 - accuracy: 0.8638 - val_l
oss: 0.5685 - val_accuracy: 0.8228
Epoch 12/100
281/281 [==============================] - 131s 467ms/step - loss: 0.3923 - accuracy: 0.8735 - val_l
oss: 0.9384 - val_accuracy: 0.7340
Epoch 13/100
281/281 [==============================] - 133s 474ms/step - loss: 0.3725 - accuracy: 0.8794 - val_l
oss: 0.4266 - val_accuracy: 0.8592
Epoch 14/100
281/281 [==============================] - 133s 474ms/step - loss: 0.3609 - accuracy: 0.8825 - val_l
oss: 0.4366 - val_accuracy: 0.8576
Epoch 15/100
281/281 [==============================] - 134s 477ms/step - loss: 0.3286 - accuracy: 0.8933 - val_l
oss: 0.8717 - val_accuracy: 0.7393
Epoch 16/100
281/281 [==============================] - 136s 484ms/step - loss: 0.3130 - accuracy: 0.9001 - val_l
oss: 0.7410 - val_accuracy: 0.7967
Epoch 17/100
281/281 [==============================] - 132s 469ms/step - loss: 0.3092 - accuracy: 0.8998 - val_l
oss: 0.3593 - val_accuracy: 0.8806
Epoch 18/100
281/281 [==============================] - 134s 477ms/step - loss: 0.2896 - accuracy: 0.9070 - val_l
```

```
oss: 0.4192 - val_accuracy: 0.8688
Epoch 19/100
281/281 [==============================] - 133s 472ms/step - loss: 0.2694 - accuracy: 0.9137 - val_l
oss: 0.3361 - val_accuracy: 0.8998
Epoch 20/100
281/281 [==============================] - 129s 458ms/step - loss: 0.2699 - accuracy: 0.9161 - val_l
oss: 0.3338 - val_accuracy: 0.8940
Epoch 21/100
281/281 [==============================] - 129s 459ms/step - loss: 0.2624 - accuracy: 0.9152 - val_l
oss: 0.3663 - val_accuracy: 0.8877
Epoch 22/100
281/281 [==============================] - 131s 467ms/step - loss: 0.2501 - accuracy: 0.9185 - val_l
oss: 0.4382 - val_accuracy: 0.8639
Epoch 23/100
281/281 [==============================] - 130s 461ms/step - loss: 0.2519 - accuracy: 0.9197 - val_l
oss: 0.5820 - val_accuracy: 0.8384
Epoch 24/100
281/281 [==============================] - 139s 495ms/step - loss: 0.2426 - accuracy: 0.9240 - val_l
oss: 0.3229 - val_accuracy: 0.9034
Epoch 25/100
281/281 [==============================] - 143s 510ms/step - loss: 0.2282 - accuracy: 0.9280 - val_l
oss: 0.3494 - val_accuracy: 0.8940
Epoch 26/100
281/281 [==============================] - 138s 491ms/step - loss: 0.2282 - accuracy: 0.9289 - val_l
oss: 0.3791 - val_accuracy: 0.8851
Epoch 27/100
281/281 [==============================] - 132s 471ms/step - loss: 0.2201 - accuracy: 0.9288 - val_l
oss: 0.2320 - val_accuracy: 0.9252
Epoch 28/100
281/281 [==============================] - 135s 479ms/step - loss: 0.2207 - accuracy: 0.9312 - val_l
oss: 0.3931 - val_accuracy: 0.8826
Epoch 29/100
281/281 [==============================] - 135s 481ms/step - loss: 0.2179 - accuracy: 0.9318 - val_l
oss: 0.2930 - val_accuracy: 0.9092
Epoch 30/100
281/281 [==============================] - 131s 467ms/step - loss: 0.2057 - accuracy: 0.9370 - val_l
oss: 0.4461 - val_accuracy: 0.8652
Epoch 31/100
281/281 [==============================] - 138s 490ms/step - loss: 0.2061 - accuracy: 0.9357 - val_l
oss: 0.2509 - val_accuracy: 0.9185
Epoch 32/100
281/281 [==============================] - 136s 483ms/step - loss: 0.2046 - accuracy: 0.9359 - val_l
oss: 0.3411 - val_accuracy: 0.8969
Epoch 33/100
281/281 [==============================] - 132s 470ms/step - loss: 0.1978 - accuracy: 0.9385 - val_l
oss: 0.4390 - val_accuracy: 0.8788
Epoch 34/100
281/281 [==============================] - 132s 469ms/step - loss: 0.1877 - accuracy: 0.9420 - val_l
oss: 0.4584 - val_accuracy: 0.8630
Epoch 35/100
281/281 [==============================] - 137s 489ms/step - loss: 0.1880 - accuracy: 0.9418 - val_l
oss: 0.2382 - val_accuracy: 0.9255
Epoch 36/100
281/281 [==============================] - 134s 477ms/step - loss: 0.1781 - accuracy: 0.9434 - val_l
oss: 0.3078 - val_accuracy: 0.9114
Epoch 37/100
281/281 [==============================] - 133s 471ms/step - loss: 0.1858 - accuracy: 0.9437 - val_l
oss: 0.3713 - val_accuracy: 0.8873
Epoch 38/100
281/281 [==============================] - 134s 476ms/step - loss: 0.1820 - accuracy: 0.9449 - val_l
oss: 0.2263 - val_accuracy: 0.9250
Epoch 39/100
281/281 [==============================] - 134s 477ms/step - loss: 0.1809 - accuracy: 0.9453 - val_l
oss: 0.4650 - val_accuracy: 0.8726
Epoch 40/100
281/281 [==============================] - 138s 489ms/step - loss: 0.1663 - accuracy: 0.9474 - val_l
oss: 0.1690 - val_accuracy: 0.9489
Epoch 41/100
281/281 [==============================] - 134s 475ms/step - loss: 0.1723 - accuracy: 0.9473 - val_l
oss: 0.3327 - val_accuracy: 0.9005
Epoch 42/100
281/281 [==============================] - 135s 480ms/step - loss: 0.1550 - accuracy: 0.9540 - val_l
oss: 0.3882 - val_accuracy: 0.8846
Epoch 43/100
281/281 [==============================] - 132s 469ms/step - loss: 0.1609 - accuracy: 0.9493 - val_l
oss: 0.3929 - val_accuracy: 0.8922
Epoch 44/100
281/281 [==============================] - 136s 485ms/step - loss: 0.1642 - accuracy: 0.9512 - val_l
oss: 0.2316 - val_accuracy: 0.9284
Epoch 45/100
281/281 [==============================] - 134s 476ms/step - loss: 0.1565 - accuracy: 0.9503 - val_l
oss: 0.2183 - val_accuracy: 0.9366
Epoch 46/100
```

```
281/281 [==============================] - 135s 481ms/step - loss: 0.1549 - accuracy: 0.9537 - val_l
oss: 0.2164 - val_accuracy: 0.9299
Epoch 47/100
281/281 [==============================] - 134s 476ms/step - loss: 0.1645 - accuracy: 0.9500 - val_l
oss: 0.2303 - val_accuracy: 0.9331
Epoch 48/100
281/281 [==============================] - 134s 477ms/step - loss: 0.1439 - accuracy: 0.9574 - val_l
oss: 0.7071 - val_accuracy: 0.8511
Epoch 49/100
281/281 [==============================] - 135s 480ms/step - loss: 0.1481 - accuracy: 0.9561 - val_l
oss: 0.1810 - val_accuracy: 0.9440
Epoch 50/100
281/281 [==============================] - 134s 477ms/step - loss: 0.1380 - accuracy: 0.9586 - val_l
oss: 0.2181 - val_accuracy: 0.9355
Epoch 51/100
281/281 [==============================] - 133s 471ms/step - loss: 0.1419 - accuracy: 0.9570 - val_l
oss: 0.1824 - val_accuracy: 0.9460
Epoch 52/100
281/281 [==============================] - 134s 476ms/step - loss: 0.1348 - accuracy: 0.9591 - val_l
oss: 0.3641 - val_accuracy: 0.9016
Epoch 53/100
281/281 [==============================] - 130s 463ms/step - loss: 0.1455 - accuracy: 0.9556 - val_l
oss: 0.1847 - val_accuracy: 0.9453
Epoch 54/100
281/281 [==============================] - 130s 461ms/step - loss: 0.1391 - accuracy: 0.9585 - val_l
oss: 0.2141 - val_accuracy: 0.9384
Epoch 55/100
281/281 [==============================] - 128s 455ms/step - loss: 0.1330 - accuracy: 0.9585 - val_l
oss: 0.1290 - val_accuracy: 0.9621
Epoch 56/100
281/281 [==============================] - 125s 446ms/step - loss: 0.1398 - accuracy: 0.9593 - val_l
oss: 0.4562 - val_accuracy: 0.8753
Epoch 57/100
281/281 [==============================] - 126s 449ms/step - loss: 0.1296 - accuracy: 0.9607 - val_l
oss: 0.3413 - val_accuracy: 0.9141
Epoch 58/100
281/281 [==============================] - 127s 453ms/step - loss: 0.1247 - accuracy: 0.9628 - val_l
oss: 0.2970 - val_accuracy: 0.9125
Epoch 59/100
281/281 [==============================] - 125s 445ms/step - loss: 0.1313 - accuracy: 0.9622 - val_l
oss: 0.4540 - val_accuracy: 0.8732
Epoch 60/100
281/281 [==============================] - 128s 454ms/step - loss: 0.1260 - accuracy: 0.9628 - val_l
oss: 0.3031 - val_accuracy: 0.9217
Epoch 61/100
281/281 [==============================] - 128s 454ms/step - loss: 0.1252 - accuracy: 0.9628 - val_l
oss: 0.1651 - val_accuracy: 0.9514
Epoch 62/100
281/281 [==============================] - 126s 449ms/step - loss: 0.1312 - accuracy: 0.9608 - val_l
oss: 0.2239 - val_accuracy: 0.9322
Epoch 63/100
281/281 [==============================] - 128s 457ms/step - loss: 0.1216 - accuracy: 0.9637 - val_l
oss: 0.1756 - val_accuracy: 0.9536
Epoch 64/100
281/281 [==============================] - 129s 460ms/step - loss: 0.1215 - accuracy: 0.9653 - val_l
oss: 0.2039 - val_accuracy: 0.9386
Epoch 65/100
281/281 [==============================] - 128s 456ms/step - loss: 0.1260 - accuracy: 0.9619 - val_l
oss: 0.3576 - val_accuracy: 0.8996
Epoch 66/100
281/281 [==============================] - 128s 454ms/step - loss: 0.1215 - accuracy: 0.9624 - val_l
oss: 0.1782 - val_accuracy: 0.9509
Epoch 67/100
281/281 [==============================] - 131s 467ms/step - loss: 0.1115 - accuracy: 0.9659 - val_l
oss: 0.2101 - val_accuracy: 0.9384
Epoch 68/100
281/281 [==============================] - 143s 508ms/step - loss: 0.1031 - accuracy: 0.9671 - val_l
oss: 0.1660 - val_accuracy: 0.9527
Epoch 69/100
281/281 [==============================] - 140s 498ms/step - loss: 0.1147 - accuracy: 0.9671 - val_l
oss: 0.1323 - val_accuracy: 0.9621
Epoch 70/100
281/281 [==============================] - 136s 483ms/step - loss: 0.1102 - accuracy: 0.9682 - val_l
oss: 0.1676 - val_accuracy: 0.9484
Epoch 71/100
281/281 [==============================] - 140s 497ms/step - loss: 0.1079 - accuracy: 0.9662 - val_l
oss: 0.2142 - val_accuracy: 0.9391
Epoch 72/100
281/281 [==============================] - 142s 506ms/step - loss: 0.1108 - accuracy: 0.9666 - val_l
oss: 0.4616 - val_accuracy: 0.9063
Epoch 73/100
281/281 [==============================] - 145s 515ms/step - loss: 0.1196 - accuracy: 0.9643 - val_l
oss: 0.3285 - val_accuracy: 0.9221
```

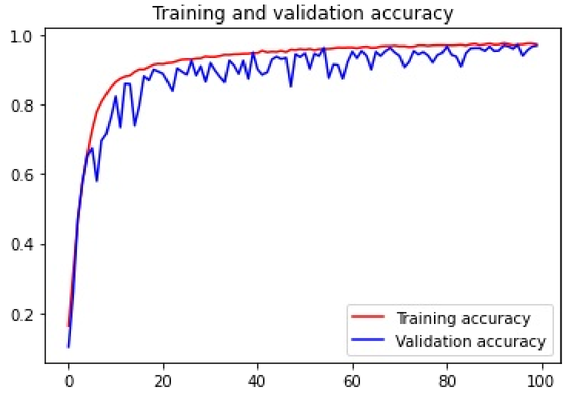
```
Epoch 74/100
281/281 [==============================] - 147s 524ms/step - loss: 0.1172 - accuracy: 0.9652 - val_l
oss: 0.1829 - val_accuracy: 0.9531
Epoch 75/100
281/281 [==============================] - 140s 498ms/step - loss: 0.1041 - accuracy: 0.9691 - val_l
oss: 0.1952 - val_accuracy: 0.9429
Epoch 76/100
281/281 [==============================] - 137s 488ms/step - loss: 0.1039 - accuracy: 0.9691 - val_l
oss: 0.1683 - val_accuracy: 0.9498
Epoch 77/100
281/281 [==============================] - 138s 490ms/step - loss: 0.1089 - accuracy: 0.9675 - val_l
oss: 0.2088 - val_accuracy: 0.9397
Epoch 78/100
281/281 [==============================] - 133s 472ms/step - loss: 0.0998 - accuracy: 0.9690 - val_l
oss: 0.2884 - val_accuracy: 0.9197
Epoch 79/100
281/281 [==============================] - 125s 444ms/step - loss: 0.1076 - accuracy: 0.9695 - val_l
oss: 0.2008 - val_accuracy: 0.9411
Epoch 80/100
281/281 [==============================] - 128s 455ms/step - loss: 0.0975 - accuracy: 0.9696 - val_l
oss: 0.1886 - val_accuracy: 0.9473
Epoch 81/100
281/281 [==============================] - 129s 458ms/step - loss: 0.1060 - accuracy: 0.9691 - val_l
oss: 0.1043 - val_accuracy: 0.9659
Epoch 82/100
281/281 [==============================] - 128s 454ms/step - loss: 0.1043 - accuracy: 0.9690 - val_l
oss: 0.1850 - val_accuracy: 0.9418
Epoch 83/100
281/281 [==============================] - 132s 468ms/step - loss: 0.0996 - accuracy: 0.9695 - val_l
oss: 0.2020 - val_accuracy: 0.9368
Epoch 84/100
281/281 [==============================] - 137s 487ms/step - loss: 0.0953 - accuracy: 0.9707 - val_l
oss: 0.4018 - val_accuracy: 0.9081
Epoch 85/100
281/281 [==============================] - 129s 460ms/step - loss: 0.1078 - accuracy: 0.9683 - val_l
oss: 0.1658 - val_accuracy: 0.9469
Epoch 86/100
281/281 [==============================] - 130s 464ms/step - loss: 0.0933 - accuracy: 0.9722 - val_l
oss: 0.1341 - val_accuracy: 0.9605
Epoch 87/100
281/281 [==============================] - 128s 456ms/step - loss: 0.0908 - accuracy: 0.9737 - val_l
oss: 0.1350 - val_accuracy: 0.9614
Epoch 88/100
281/281 [==============================] - 128s 454ms/step - loss: 0.0997 - accuracy: 0.9693 - val_l
oss: 0.1343 - val_accuracy: 0.9612
Epoch 89/100
281/281 [==============================] - 127s 453ms/step - loss: 0.0946 - accuracy: 0.9720 - val_l
oss: 0.1742 - val_accuracy: 0.9531
Epoch 90/100
281/281 [==============================] - 127s 453ms/step - loss: 0.0882 - accuracy: 0.9743 - val_l
oss: 0.1188 - val_accuracy: 0.9656
Epoch 91/100
281/281 [==============================] - 127s 453ms/step - loss: 0.0970 - accuracy: 0.9712 - val_l
oss: 0.1561 - val_accuracy: 0.9529
Epoch 92/100
281/281 [==============================] - 127s 453ms/step - loss: 0.0937 - accuracy: 0.9716 - val_l
oss: 0.1481 - val_accuracy: 0.9540
Epoch 93/100
281/281 [==============================] - 127s 450ms/step - loss: 0.0831 - accuracy: 0.9756 - val_l
oss: 0.1140 - val_accuracy: 0.9690
Epoch 94/100
281/281 [==============================] - 127s 452ms/step - loss: 0.0887 - accuracy: 0.9730 - val_l
oss: 0.1110 - val_accuracy: 0.9661
Epoch 95/100
281/281 [==============================] - 127s 451ms/step - loss: 0.1044 - accuracy: 0.9690 - val_l
oss: 0.1364 - val_accuracy: 0.9594
Epoch 96/100
281/281 [==============================] - 128s 454ms/step - loss: 0.0971 - accuracy: 0.9713 - val_l
oss: 0.1050 - val_accuracy: 0.9705
Epoch 97/100
281/281 [==============================] - 129s 459ms/step - loss: 0.0897 - accuracy: 0.9720 - val_l
oss: 0.2181 - val_accuracy: 0.9391
Epoch 98/100
281/281 [==============================] - 129s 458ms/step - loss: 0.0870 - accuracy: 0.9746 - val_l
oss: 0.1498 - val_accuracy: 0.9551
Epoch 99/100
281/281 [==============================] - 129s 460ms/step - loss: 0.0836 - accuracy: 0.9747 - val_l
oss: 0.1217 - val_accuracy: 0.9656
Epoch 100/100
281/281 [==============================] - 129s 460ms/step - loss: 0.0947 - accuracy: 0.9720 - val_l
oss: 0.1144 - val_accuracy: 0.9683
```

Plot to show training accuracy vs validation accuracy

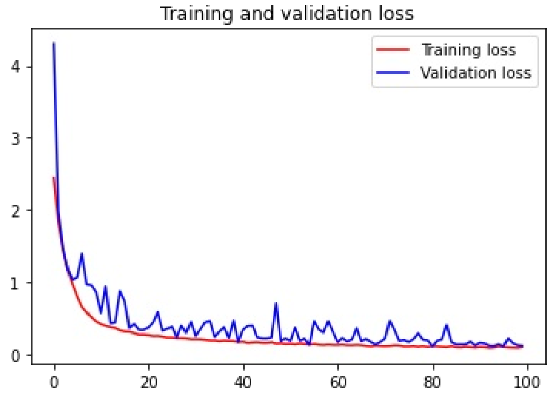
The two plots were designed to visualize the learning curve of the model. One plot concentrated on Training and validation accuracy over 60 iterations and next one concentrated on training and validation loss.

In [7]:

```
accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
loss = history.history['loss']
validation_loss = history.history['val_loss']

epochs = range(len(accuracy))

plt.plot(epochs, accuracy, 'red', label='Training accuracy')
plt.plot(epochs, validation_accuracy, 'blue', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()
plt.show()

plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, validation_loss, 'blue', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()
plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

Test data prediction

Like training data, image generator were built for test data along with its directory. The important step of the project is the prediction of test images with the learning the CNN model has undergone with training and validation images.

In [8]:

```
testparentdirectory = os.path.join(workingdir + '/state-farm-distracted-driver-detection/imgs/')
testdataimage = ImageDataGenerator(rescale = 1./255)
testdata = testdataimage.flow_from_directory(testparentdirectory, classes=['test'], target_size = (100,100))
testoutput = cnnmodel.predict(testdata, verbose = 1)
```

```
Found 79726 images belonging to 1 classes.
2492/2492 [==============================] - 425s 170ms/step
```

Preparing output dataframe

The sample submission csv was read by pandas to prepare the format of output. With image and images name taken from the csv, the prediction values was replaced with the original value of csv in the same format and convert it to a dataframe to export it easily into a csv file.

In [9]:

```
specimencsv = pd.read_csv(os.path.join(workingdir + '/state-farm-distracted-driver-detection/sample_submission.cs
v'))
result = {'img':list(specimencsv.values[:,0]),}
for value in range(0,10):
    result['c' + str(value)] = list(testoutput[:,value])
```

In [10]:

```
testoutput = pd.DataFrame(result)
```

Exploratory data analysis

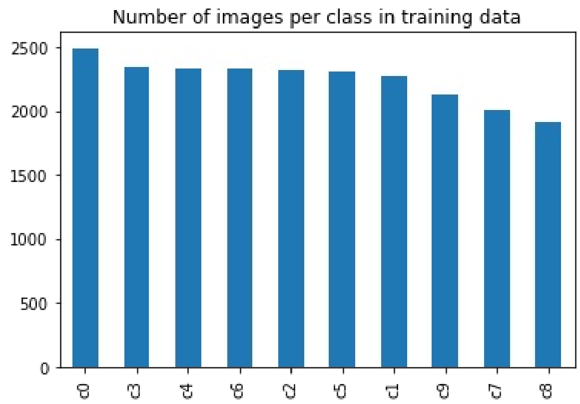
The number of images in each class were depicted with a bar plot.

In [45]:

```
imagescsv = pd.read_csv(os.path.join(workingdir + '/state-farm-distracted-driver-detection/driver_imgs_list.csv')
)
imagescsv.classname.value_counts().plot(kind = 'bar')
plt.title('Number of images per class in training data')
```

Out[45]:

```
Text(0.5, 1.0, 'Number of images per class in training data')
```



In [35]:

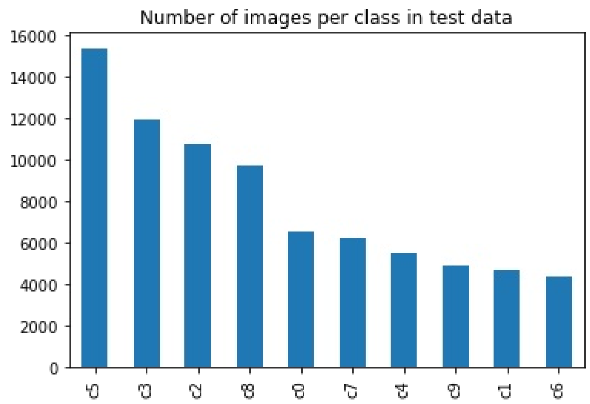
```
testvalue = testoutput[['c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9']].idxmax(axis = 1)
```

In [44]:

```
testvalue.value_counts().plot(kind = 'bar')
plt.title('Number of images per class in test data')
```

Out[44]:

```
Text(0.5, 1.0, 'Number of images per class in test data')
```



Preparing output file

Then, the csv had been written from the 'testoutput' dataframe.

In [13]:

```
testoutput.to_csv('Testoutput.csv', index = False, encoding='utf-8')
```

Conclusion

We built a model using Convolution Neural Networks with data augumentation, Batch Normalization and Dropout to increase the efficiency of the model. Thus, the prediction of test images from the model with the learning of training and validation images was successfully exported to a csv file with over 97% accuracy.

References

1. https://www.tensorflow.org/tutorials/keras/classification (https://www.tensorflow.org/tutorials/keras/classification)
2. https://www.tensorflow.org/tutorials/images/cnn (https://www.tensorflow.org/tutorials/images/cnn)
3. https://charon.me/posts/keras/keras2/ (https://charon.me/posts/keras/keras2/)
4. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator)
5. https://www.tensorflow.org/guide/keras/train_and_evaluate (https://www.tensorflow.org/guide/keras/train_and_evaluate)