State Farm Distracted Driver Detection

Team Members

Group : **15**

- Fatima Samir
- Alaa Saleh Mohamed
- Asmaa Saeed
- Omar Ahmed Mohamed
- Mohamed salama

Present to :

Dr.Ahmed El-Sallab

Content

- Dense Model
- CNN Model
- Data Augmentation
- Transfer learning
- Visualization

State Farm Distracted Driver Detection Data

- About driver images, each taken in a car with a driver doing something in the car .

- Goal is to predict the likelihood of what the driver is doing in each picture.

Pre-processing

“

- Load images and Normalize them
- Resizing images into (150*150) images
- Encode labels by using categorical mode

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator

    # All images will be rescaled by 1./255
    train_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
            # This is the target directory
            train_dir,
            # All images will be resized to 150x150
            target_size=(150, 150),
            batch_size=24,
            # Since we use binary_crossentropy loss, we need binary labels
            class_mode='categorical')

    validation_generator = test_datagen.flow_from_directory(
            validation_dir,
            target_size=(150, 150),
            batch_size=20,
            class_mode='categorical')

    Found 20424 images belonging to 10 classes.
    Found 2000 images belonging to 10 classes.
```

Dense Model

- Using neural network in which every neuron in a layer is connected to every neuron the consequent layer.

Model Parameters

- Using 4 dense layers
- at the first layers using relu activation function ,data classified to 10 classes so the last layer is softmax with 10 neurons.

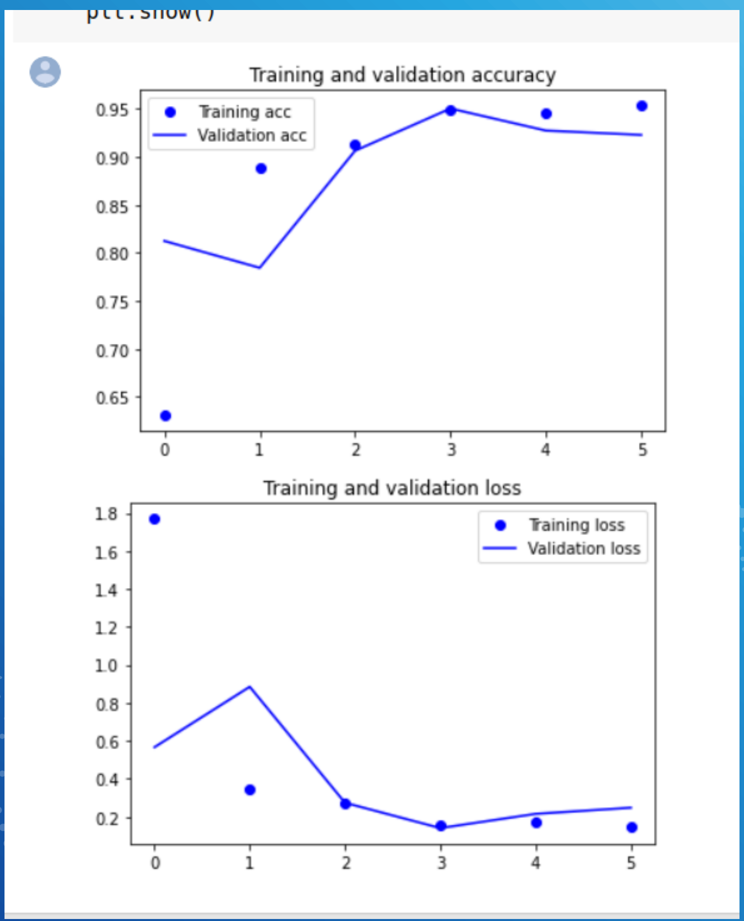
Training Model

Training is done with 10 epochs and batch size, using Adam optimizer and categorical cross entropy as the loss function.

```
from tensorflow.keras import models
from tensorflow.keras import layers
import tensorflow as tf

model = models.Sequential()
model.add(tf.keras.layers.Reshape((150*150*3,), input_shape=(150,150,3)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape_1 (Reshape)         (None, 67500)             0

 dense_8 (Dense)             (None, 512)               34560512

 dense_9 (Dense)             (None, 256)               131328

 dense_10 (Dense)            (None, 128)               32896

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 34,726,026
Trainable params: 34,726,026
Non-trainable params: 0
```

Evaluation



After training the model for 6 epochs, the training accuracy settled at `95.3 `% and the validation accuracy settled around `92.3`% which indicates that there is no overfitting .

CNN Model

“

Baseline CNN model

```
[ ]  from tensorflow.keras import layers
     from tensorflow.keras import models

     model2 = models.Sequential()
     model2.add(layers.Conv2D(32, (3, 3), activation='relu',
                              input_shape=(150, 150, 3)))
     model2.add(layers.MaxPooling2D((2, 2)))
     model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model2.add(layers.MaxPooling2D((2, 2)))
     model2.add(layers.Conv2D(128, (3, 3), activation='relu'))
     model2.add(layers.MaxPooling2D((2, 2)))
     model2.add(layers.Conv2D(128, (3, 3), activation='relu'))
     model2.add(layers.MaxPooling2D((2, 2)))
     model2.add(layers.Flatten())
     model2.add(layers.Dense(512, activation='relu'))
     model2.add(layers.Dense(10, activation='softmax'))
```

Transfer Learning

“

After apply baseline CNN model, lets using Transfer Learning concept - with fine tuning - on our dataset and see the output.

- We will use the VGG16 architecture
- Using the "convolutional base" of the VGG16 model to extract "features" and then,
- Add new classifier on top of the output “densely-connected classifier”.

Transfer Learning

```
[ ] conv_base = VGG16(weights='imagenet',
                      include_top=False,
                      input_shape=(150, 150, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

```
from tensorflow.keras import models
from tensorflow.keras import layers
conv_base.trainable =   False
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
[ ] model.summary()
```

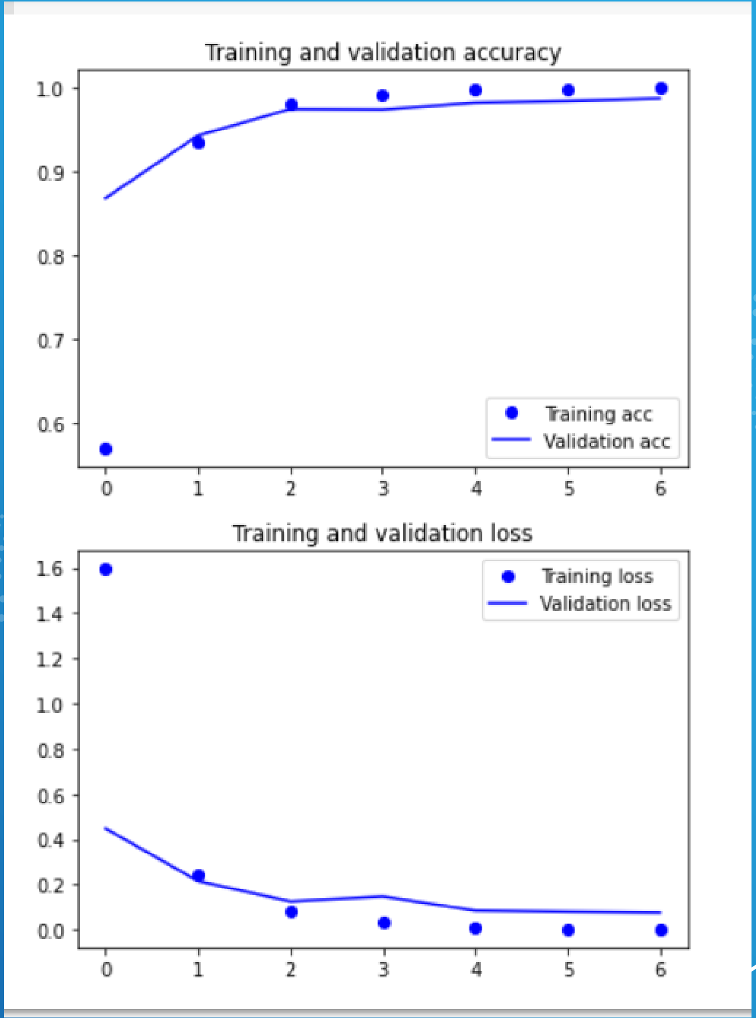
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 32)                262176

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 14,977,194
Trainable params: 262,506
Non-trainable params: 14,714,688
_________________________________________________________________
```

Transfer Learning

Evaluation :-

The graphs show the output accuracy and loss for 6 epochs on Training and validation Dataset.

- We notice with using The Transfer learning the model is more accurate which achieved 0.9855 , 0.0907 accuracy and loss respectively.



Visualization

```
import cv2

# We use cv2 to load the original image
img = cv2.imread('/content/data/imgs/train/c0/img_100337.jpg')

# We resize the heatmap to have the same size as the original image
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

# We convert the heatmap to RGB
heatmap = np.uint8(255 * heatmap)

# We apply the heatmap to the original image
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

# 0.4 here is a heatmap intensity factor
superimposed_img = heatmap * 0.4 + img

# Save the image to disk
cv2.imwrite('output.jpg', superimposed_img)
```

```
[ ] from google.colab.patches import cv2_imshow
    cv2_imshow(superimposed_img)
```