BBM416 - FUNDAMENTALS OF COMPUTER VISION
DISTRACTED DRIVER DETECTION
FINAL REPORT

Sena Çakırlı
2200356074

Zübeyde Civelek
2200356814

Mehmet Akif Özgür
2200356832

1 INTRODUCTION

Driver distraction has been identified as one of the leading causes of accidents on the roads. Distractions can arise from various factors, such as mobile phone usage, drinking, operating instruments, facial makeup, and social interaction. The consequences of driver distraction can be severe, resulting in injuries, loss of lives, and property damage. To mitigate these risks and enhance road safety, it is crucial to develop effective methods to detect and classify driver distractions in real-time.

The aim of our project is to build a machine learning model using computer vision techniques to classify driver distractions. By leveraging the power of computer vision, we intend to develop an automated system that can accurately identify and categorize different types of distractions exhibited by drivers during their journeys. The ultimate goal is to enable real-time detection and alert mechanisms that can assist drivers in maintaining focus and minimizing the occurrence of accidents caused by distractions.

Our motivation for undertaking this project stems from the potential impact it can have on enhancing road safety and reducing the number of accidents caused by driver distractions. By developing an efficient and accurate driver distraction detection system, we aim to contribute to the larger theme of improving transportation safety and reducing human errors on the roads. We believe that the application of computer vision techniques can play a significant role in addressing this problem, paving the way for more intelligent and proactive driving assistance systems.

In summary, our project aims to address the problem of driver distraction by developing a machine learning model using computer vision techniques.

2 METHOD

We utilized the Image Classification method to address the Distracted Driver problem. In order to apply this method, we gathered labeled data, performed preprocessing operations on the data, selected an appropriate model for our method, and trained this model. We, then tested the trained data with data that is not used for training and evaluated the accuracy of our model's image classification using various metrics. Let's take a closer look at the processes we undertook for the Image Classification method:

2.1 DATA COLLECTION

We have access to a dataset that greatly benefits us in solving our problem. We utilized this dataset, which can be accessed through the provided link, for our problem. The dataset comprises 9 different classes of distracted drivers and 1 class of focused drivers. We tried to aim to accurately classify the images using the model we trained.

2.2 Data Preprocessing

To achieve better classification in addressing our problem, we performed certain operations on our data. For this project, we resized the images, and since our test and train images are captured from the same camera and angle, we did not employ any data augmentation techniques. Our dataset had different numbers of images in different classes to have a balanced dataset, we brought them to same number and added extra images we removed from the train dataset to test dataset.

2.3 Model

2.3.1 Model Selection

We employed the Convolutional Neural Network (CNN) architecture, which is widely used in Image Classification. We used this architecture, which has proven successful with many CNN-based models such as AlexNet, ZFNet, GoogLeNet, ResNet, etc., to make a good classification in our project. CNN is a deep learning algorithm commonly used in image processing, which takes images as input. This algorithm consists of different layers that capture features in the images and classify them. By passing through layers such as Convolutional Layer, Pooling, and Fully Connected, the image undergoes various operations to become compatible with the deep learning model.

Convolutional Layer: Used for feature detection

Pooling (Down-sampling) Layer: Reduces the number of weights and checks suitability

Flattening Layer: Prepares data for a classic neural network

Fully-Connected Layer: Utilized in classification with a standard neural network

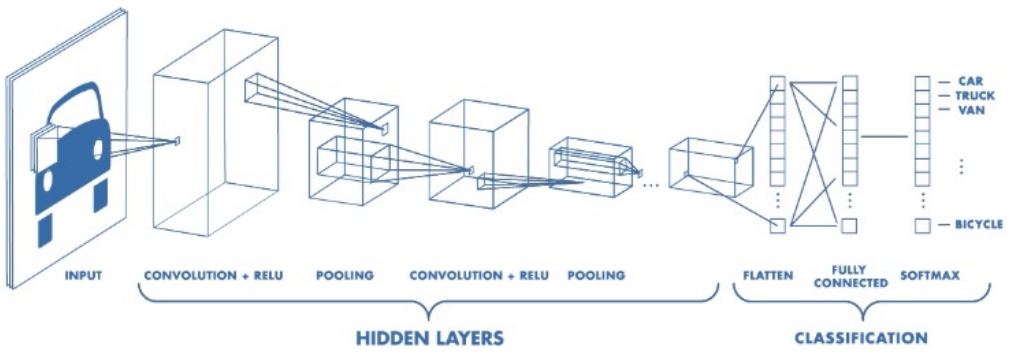

Figure 1: Convolutional Neural Network Architecture

2.3.2 Our Model

We made our model using Keras framework that comprise of 6 convolution, 6 max-pooling, and 4 dense layers. We used RMSprop (Root Mean Squared Propagation) as optimizer. The RMSprop optimizer restricts the oscillations in the vertical direction, and allow us to increase our learning rate and our algorithm to take larger steps in the horizontal direction converging faster. We used Categorical Cross-entropy as loss function. Categorical cross-entropy is used when true labels are one-hot encoded. We used ReLU activation function for all layers except the final full connected layer, which we used Softmax activation function.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 254, 254, 32)      896

 max_pooling2d_10 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 conv2d_11 (Conv2D)          (None, 125, 125, 64)      18496

 max_pooling2d_11 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 conv2d_12 (Conv2D)          (None, 60, 60, 128)       73856

 max_pooling2d_12 (MaxPooli  (None, 30, 30, 128)       0
 ng2D)

 conv2d_13 (Conv2D)          (None, 28, 28, 128)       147584

 max_pooling2d_13 (MaxPooli  (None, 14, 14, 128)       0
 ng2D)

 conv2d_14 (Conv2D)          (None, 12, 12, 256)       295168

 max_pooling2d_14 (MaxPooli  (None, 6, 6, 256)         0
 ng2D)

 conv2d_15 (Conv2D)          (None, 4, 4, 512)         1180160

 max_pooling2d_15 (MaxPooli  (None, 2, 2, 512)         0
 ng2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dense_26 (Dense)            (None, 1024)              2098176

 dense_27 (Dense)            (None, 512)               524800

 dense_28 (Dense)            (None, 128)               65664

 dense_29 (Dense)            (None, 10)                1290

=================================================================
Total params: 4406090 (16.81 MB)
Trainable params: 4406090 (16.81 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2: The Model

2.4 Model Performance

We attempt to predict the classes of the data that are not used during the training phase using the trained model. We compare these predictions with the correct labels and calculate how accurately our model classifies the images. To measure the performance of the model, we utilize performance evaluation metrics such as accuracy, precision, recall, and F1 score.

3 Experimental Settings

3.1 Dataset

For our project, we have utilized the State Farm Distracted Driver Detection dataset, which is available on Kaggle. This dataset is widely used in the field of driver distraction recognition and classification tasks. It comprises a large collection of labeled images, each representing a driver engaged in a specific distraction activity. The dataset consists of ten different distraction classes, including activities such as talking on the phone, texting, operating the radio, drinking, reaching behind, applying makeup, talking to a passenger, using a laptop, looking at the rear-view mirror, and looking outside the window. The dataset contains 15.200 labeled train images and 3.040 test images.

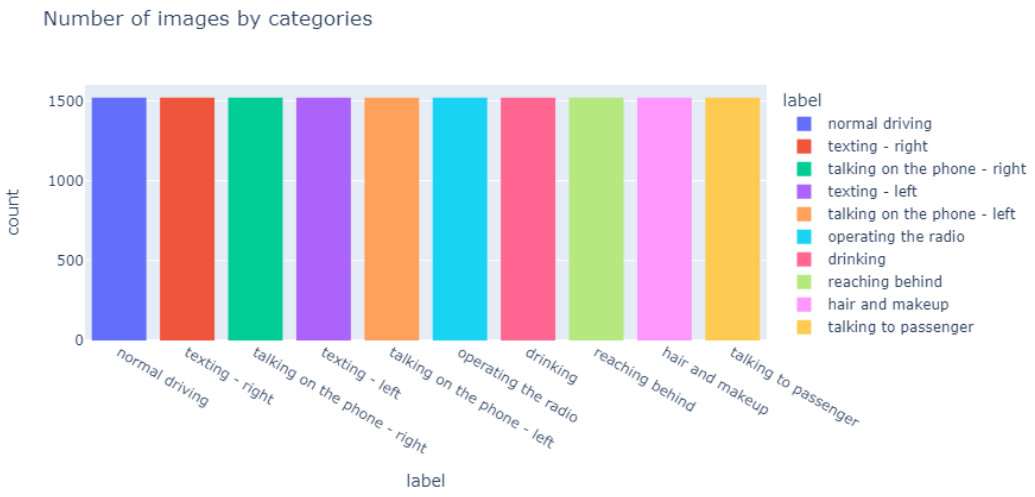

Figure 3: Distribution of The Dataset

3.2 Environments

To conduct our experiments, we have utilized a MacBook Pro 14" with an M1 Pro chip. The specific configuration of our machine includes an 8-core CPU, a 14-core GPU, 16GB of RAM, and a 512GB SSD. The operating system used for our experiments is macOS.

We have implemented our machine learning models using the Python programming language, leveraging popular libraries such as TensorFlow and Keras for deep learning tasks. Additionally, we have utilized OpenCV for image preprocessing and manipulation, used Matplotlib and Seaborn for visualization and OS Module for reading data.

3.3 Evaluation

To evaluate the performance of our driver distraction detection method, we have employed the following evaluation metrics:

Accuracy: We calculated the overall accuracy of our model in correctly classifying driver distractions. It measures the proportion of correctly classified images over the total number of images in the test set.

Precision and Recall: We computed the precision and recall values for each distraction class. Precision measures the percentage of correctly classified instances for a specific class, while recall indicates the proportion of correctly classified instances out of all instances belonging to a particular class.

F1 Score: The F1 score combines precision and recall into a single metric, providing a balanced measure of performance. It is calculated as the harmonic mean of precision and recall.

Confusion Matrix: We generated a confusion matrix to gain insights into the performance of our model across different distraction classes. The confusion matrix displays the number of instances classified correctly and incorrectly for each class, enabling us to analyze any patterns of misclassification.

By utilizing these evaluation metrics, we aim to assess the accuracy, precision, recall, and overall performance of our driver distraction detection model. These metrics help us understand the strengths and weaknesses of our approach and guide us in further refining and optimizing our models and algorithms.

4 Experimental Results

We resized all images to (256, 256) to have a uniform dataset and calculate them easily. We used 128 as batch size, as a result we had 95 batches. We used (3, 3) kernel size for all convolution layers in the model.

We used early stopping that depends on validation accuracy value to stop over-fitting. So even though we aim to train our model with 50 epochs, it stopped after 7 epochs. After training the model with train dataset we gained 0.9701 accuracy and 0.1163 loss for training, and 0.9102 accuracy and 0.3525 loss for validation. After testing the model with test dataset we gained 0.9105 accuracy and 0.3471 loss for testing.

We split our training dataset with 80-20 percentage for train and validation, ended up with 12.160 images for train and 3.040 images for validation. Training our model took 75 minutes 19.8 seconds and testing it with 3.040 images took 23 seconds.

```
history = model.fit(
    train_ds,
    epochs=50,
    validation_data=val_ds,
    callbacks=[early_stopping])
✓ 75m 19.8s

Epoch 1/50
95/95 [==============================] - 489s 5s/step - loss: 10.4874 - accuracy: 0.1781 - val_loss: 1.6732 - val_accuracy: 0.3280
Epoch 2/50
95/95 [==============================] - 587s 6s/step - loss: 1.2464 - accuracy: 0.5633 - val_loss: 0.5732 - val_accuracy: 0.8197
Epoch 3/50
95/95 [==============================] - 629s 7s/step - loss: 0.5097 - accuracy: 0.8341 - val_loss: 0.3242 - val_accuracy: 0.8957
Epoch 4/50
95/95 [==============================] - 650s 7s/step - loss: 0.2738 - accuracy: 0.9200 - val_loss: 0.1616 - val_accuracy: 0.9576
Epoch 5/50
95/95 [==============================] - 745s 8s/step - loss: 0.1708 - accuracy: 0.9516 - val_loss: 0.1508 - val_accuracy: 0.9618
Epoch 6/50
95/95 [==============================] - 730s 8s/step - loss: 0.1648 - accuracy: 0.9595 - val_loss: 0.1394 - val_accuracy: 0.9645
Epoch 7/50
95/95 [==============================] - 691s 7s/step - loss: 0.1163 - accuracy: 0.9701 - val_loss: 0.3525 - val_accuracy: 0.9102
```

Figure 4: Train Results

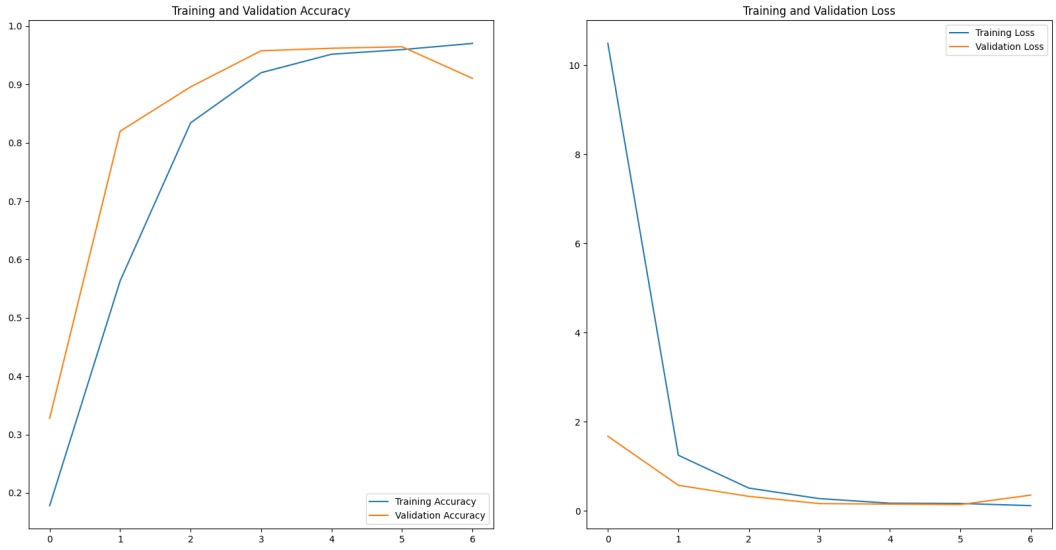

Figure 5: Learning Curves

```
test_loss, test_accuracy = model.evaluate(test_ds)

✓ 23.0s

95/95 [==============================] - 23s 240ms/step - loss: 0.3471 - accuracy: 0.9105
```

Figure 6: Test Results

	precision	recall	f1-score	support
normal driving	0.84	0.94	0.89	304
texting - right	0.95	0.98	0.96	304
talking on the phone - right	0.95	0.99	0.97	304
texting - left	0.99	0.54	0.70	304
talking on the phone - left	0.98	0.93	0.95	304
operating the radio	0.98	0.98	0.98	304
drinking	0.99	0.92	0.95	304
reaching behind	0.99	0.94	0.97	304
hair and makeup	0.71	0.96	0.81	304
talking to passenger	0.86	0.93	0.89	304
accuracy			0.91	3040
macro avg	0.92	0.91	0.91	3040
weighted avg	0.92	0.91	0.91	3040

Figure 7: Classification Report

Figure 8: Confusion Matrix

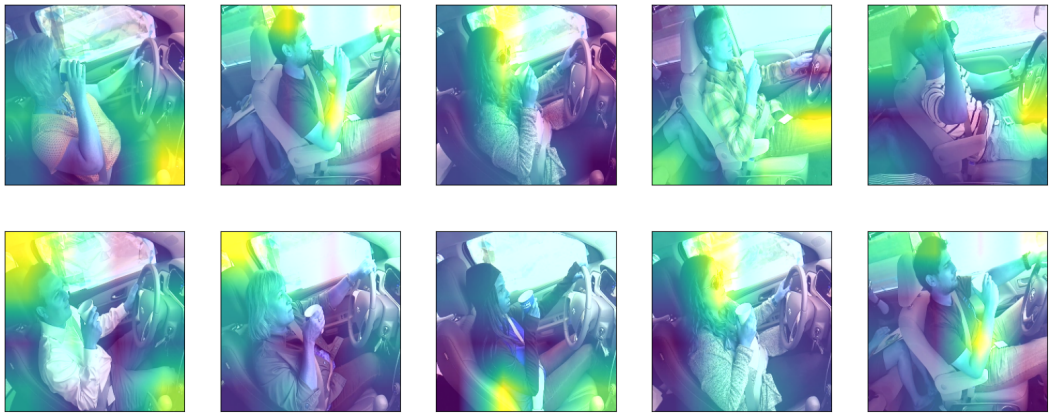

Figure 9: Drinking Class HeatMap

Figure 10: Hair and Makeup Class HeatMap

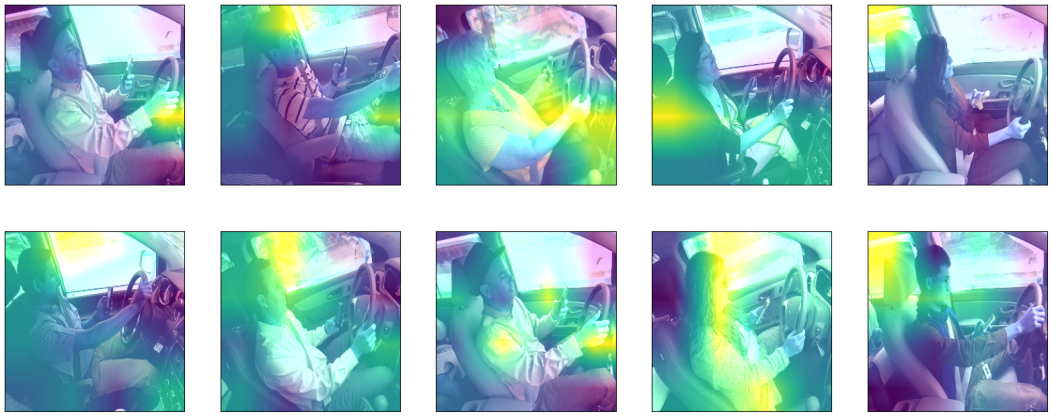

Figure 11: Texting - Left Class HeatMap

5 Conclusions

Based on the confusion matrix, the best-predicted classes were determined to be ”texting-left”, ”drinking” and ”reaching behind”. On the other hand, the worst-predicted classes were ”hair and makeup” and ”talking to passenger”. Furthermore, the most commonly mispredicted classes was ”texting-left” when the actual activities were ”normal driving”, ”hair and makeup” and ”talking to passenger”. Further improvements may be required in those area.

The project demonstrated that the model’s prediction run-time was fast, indicating its suitability for real-time scenarios.

The project encountered hardware limitations that restricted the exploration of various CNN models and constrained the dataset size. This reduction in resources may have had an impact on the overall performance of the model.

Overall, this project has established a foundation for activity recognition in driving scenarios. However, further refinement and expansion are necessary to achieve higher levels of accuracy and robustness. By addressing hardware limitations, refining the dataset, and focusing on improving classification accuracy, future iterations of the model can contribute significantly to the advancement of computer vision techniques in the field of driving activity recognition.