A PROJECT REPORT

DISTRACTED DRIVER RECOGNITION USING SUPPORT VECTOR MACHINE WITH CONVOLUTIONAL NEURAL NETWORKS FOR FEATURE EXTRACTION AND PRINCIPAL COMPONENT ANALYSIS FOR FEATURE REDUCTION

21CSC305P –MACHINE LEARNING

(2021 Regulation)

III Year/ V Semester

Academic Year: 2024 -2025

Submitted by

ARIHANT DEBNATH [RA2211003011033]
DIVA MERJA [RA2211003011034]
MAKADIA YAKSHKUMAR VIJAYAKUMAR[RA2211003011035]
KRISHNA WADHWANI[RA2211003011045]

Under the Guidance of

Dr.Anitha K
Designation
Department of Computational Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203

NOVEMBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that **21CSC305P - MACHINE LEARNING** project report titled "**DISTRACTED DRIVER RECOGNITION USING SUPPORT VECTOR MACHINE WITH CONVOLUTIONAL NEURAL NETWORKS FOR FEATURE EXTRACTION AND PRINCIPAL COMPONENT ANALYSIS FOR FEATURE REDUCTION**" is the bonafide work of "Arihant Debnath [RA2211003011033], Diva Merja [RA2211003011034], Makadia Yaksh Vijayakumar [RA2211003011035], Krishna Wadhwani [RA2211003011045]" who carried out the task of completing the project within the allotted time.

SIGNATURE
Dr. Anitha K
Course Faculty
Assistant Professor
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

SIGNATURE
Dr. G. Niranjana
Head of the Department
Professor
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

ABSTRACT

The increased use of ground transportation in India, especially in cities like Chennai, has led to a sharp rise in road traffic, which has, in turn, contributed to an uptick in traffic accidents. One primary factor behind these incidents is driver distraction. To address this issue, this study explores a technique to identify distracted drivers through image classification, utilizing Convolutional Neural Networks (CNN) alongside Support Vector Machines (SVM). The research leverages the "State Farm Distracted Driver Detection" dataset, which contains images capturing drivers' attention levels, including instances of distraction. Initially, the data undergoes preprocessing, including resizing images to 100x100 pixels and splitting the dataset for training and testing. Next, a CNN model with three convolutional layers, three MaxPooling layers, and one flattened layer is used for feature extraction. To further optimize, Principal Component Analysis (PCA) is applied to reduce the data's dimensionality. Following this, an SVM model is trained on the PCA-reduced data using a 60:40 data split for training and testing. The study compares model performance with and without PCA. Results indicate that applying PCA changes classification accuracy from 96.28% to 99.0% and reduces training time from 19.67 seconds to 10.64 seconds.

TABLE OF CONTENTS

ABSTRACT **iii**

LIST OF FIGURES **v**

LIST OF TABLES **vi**

ABBREVIATIONS **vii**

1 INTRODUCTION **1**

1.1 Problem Statement 1

1.2 Objective 1

2 LITERATURE SURVEY **3**

3 SOFTWARE REQUIREMENTS AND SPECIFICATIONS **5**

4 DATASET DESCRIPTION **7**

5 METRIC EVALUATION **9**

6 MATERIALS AND METHODS **11**

7 RESULTS AND DISCUSSIONS **21**

7.1 Results 21

7.2 Discussion 21

7.3 Comparative Analysis of Results 23

8 CONCLUSION AND FUTURE ENHANCEMENT **27**

REFERENCES **28**

APPENDIX

LIST OF FIGURES

Figure No	Title of the Figure	Page
1	Causes of Road Traffic Accidents	1
2	CNN Model Architecture	3
3	SVM Model Architecture	4
4	State Farm Distracted Driver Dataset	7
5	Kaggle Dataset	11
6	Importing The Dataset	11
7	Data Preprocessing	12
8	CNN Implementation	14
9	Extracted Features(128)	14
10	PCA Implementation	15
11	Feature Reduction With PCA	15
12	SVM Implementation	16
13	KNN Implementation	17
14	Random Forest Implementation	18
15	Model Evaluation	19
16	Comparative Analysis	20
17	Results for Given Dataset	21
18	Confusion Matrix for SVM with and without PCA	22
19	Confusion Matrix for KNN with and without PCA	24
20	Confusion Matrix for Random Forest with and without PCA	24
21	Graphical representation of Comparative Analysis	26

LIST OF TABLES

Table No	Title of the Table	Page No
1	Classes in Dataset	7
2	Performance Comparison with and without PCA	22
3	Performance Comparison with and without PCA for KNN and Random Forest	25

ABBREVIATIONS

SVM - Support Vector Machine
CNN - Convolutional Neural Network
PCA - Principal Component Analysis
KNN - K-Nearest Neighbour

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The rapid increase in ground transportation, especially in densely populated cities like Surabaya, has led to a higher incidence of traffic accidents in Indonesia. One of the leading causes of these accidents is driver distraction, often due to behaviors such as mobile phone use, eating, or interacting with passengers while driving. Such distractions divert attention away from the road, significantly heightening the risk of accidents and endangering lives. Addressing this issue requires innovative solutions that can detect distracted driving behaviors accurately and in real-time. Traditional driver monitoring methods, such as manual observation or basic sensor-based systems, fall short in terms of scalability and precision. Manual methods are labor-intensive, while basic sensors lack the sophistication needed to interpret driver behavior accurately. Thus, there is a critical need for an automated system that can detect driver distractions effectively and contribute to enhancing road safety.

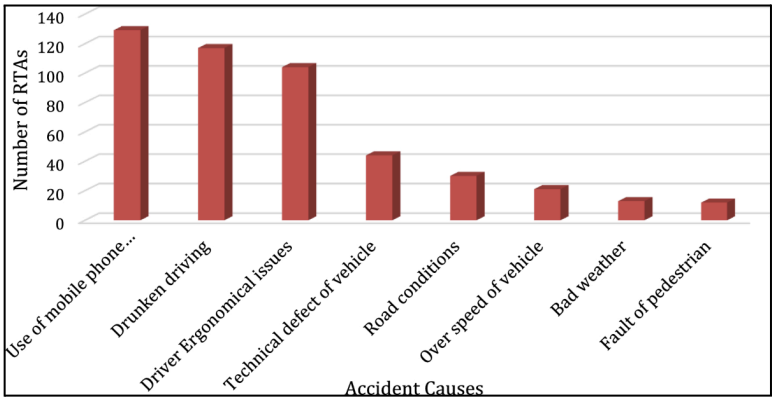

Fig. 1 - Causes of Road Traffic Accidents

1.2 Objective

The objective of this study is to develop a system that can accurately detect distracted drivers using a combination of Convolutional Neural Networks (CNN) for feature extraction, Principal Component Analysis (PCA) for dimensionality reduction, and Support Vector Machines (SVM) for classification. This system seeks to overcome the limitations of traditional monitoring methods by offering a scalable, efficient, and precise solution for real-time distracted driving detection.

1.2.1 **Data Representation and Preprocessing**

The system will process image data captured by in-vehicle cameras, converting it into a consistent format for analysis. Each image undergoes preprocessing steps such as resizing to a standardized resolution, enhancing the CNN's ability to extract relevant features effectively. This standardized approach also facilitates the generalization of the model across various driving conditions.

1.2.2 **Feature Extraction through CNN**

The CNN model extracts crucial features from the preprocessed images, identifying patterns indicative of distraction, such as hand movements and gaze direction. Using convolutional layers, the CNN is able to capture both simple and complex features, which are essential for accurate classification.

1.2.3 **Dimensionality Reduction using PCA**

After feature extraction, Principal Component Analysis (PCA) is applied to reduce the feature set's dimensionality. This reduction process retains only the most informative features, allowing the model to operate efficiently without sacrificing performance. By streamlining the data, PCA improves computational efficiency and accelerates the system's response time, making it suitable for real-time detection.

1.2.4 **Classification with SVM**

Following PCA, the SVM model classifies drivers based on the reduced feature set. SVM's ability to handle high-dimensional data and maximize the separation between classes ensures that the system accurately distinguishes between distracted and non-distracted drivers. This approach minimizes errors and enhances the reliability of the system in real-world applications.

1.2.5 **Enhancing Accuracy and Robustness**

To achieve reliable performance, the model will undergo rigorous training and tuning to minimize false positives and false negatives. Cross-validation across different datasets ensures that the system generalizes well, adapting to variations in driver behavior and environmental conditions.

1.2.7 **Evaluation Metrics**

The system's performance will be assessed using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC, providing a comprehensive understanding of its effectiveness in classifying driver distractions. Comparative analysis with traditional methods will further highlight the advantages of using CNN-PCA-SVM for distracted driver detection.

CHAPTER 2

LITERATURE SURVEY

Ground transportation in India, especially in metropolitan areas like Chennai, is extensively utilized due to well-developed road infrastructure that facilitates efficient traffic flow [1]. However, this high usage also correlates with an elevated incidence of traffic accidents. Developing nations report around 1.25 million annual fatalities due to traffic-related incidents, with a significant proportion attributed to driver distraction or impairment, such as mobile phone use, eating, or drinking while driving [2][3].

In Surabaya alone, 2017 statistics document 1,338 accidents, 461 of which involved private vehicles, resulting in 71 fatalities and 47 severe injuries. Driver negligence, notably from distractions like texting or eating, is a primary contributing factor [4]. Enhancing road safety and reducing driver distraction is thus a critical focus area. One proposed solution is the application of advanced image classification technology to monitor driver behavior and distinguish between safe and distracting actions.

Recent studies have demonstrated the potential of machine learning models in this domain. For instance, Convolutional Neural Networks (CNNs) have been shown to outperform Vision Transformers for distracted driver detection, with a DenseNet model achieving 90% accuracy [5].

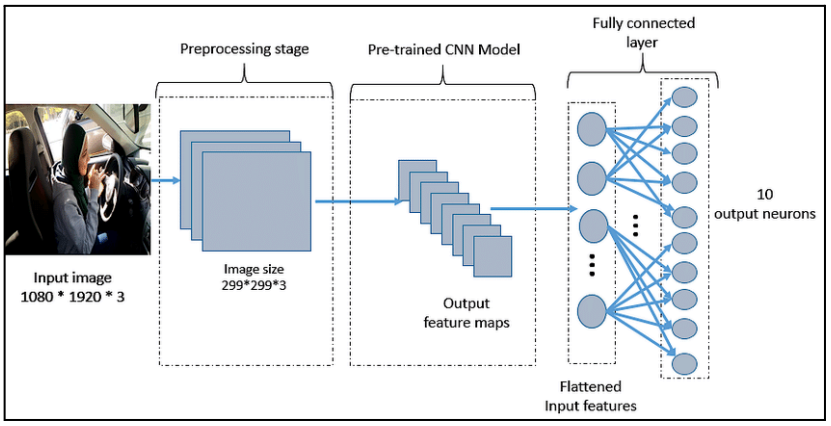

Fig. 2 -CNN model architecture

Additionally, Bu et al. (2019) utilized Histogram of Oriented Gradients (HOG) for feature extraction combined with Support Vector Machine (SVM) for classification, reaching an accuracy of 90.84% in detecting distracted behaviors [6]. More recently, Pisharody et al. (2022) compared multiple machine learning techniques—SVM, Random Forest, and Naive

Bayes—along with an ensemble approach to classify distracted driving behaviors. SVM outperformed the other models in detecting distraction patterns, achieving high accuracy using HOG features extracted from the green channel of images [7]. Furthermore, recent research published in the IRJET has highlighted the success of CNN architectures in driver distraction detection, offering an enhancement to traditional machine learning methods, supporting their integration into real-world systems for more accurate and reliable monitoring [8].

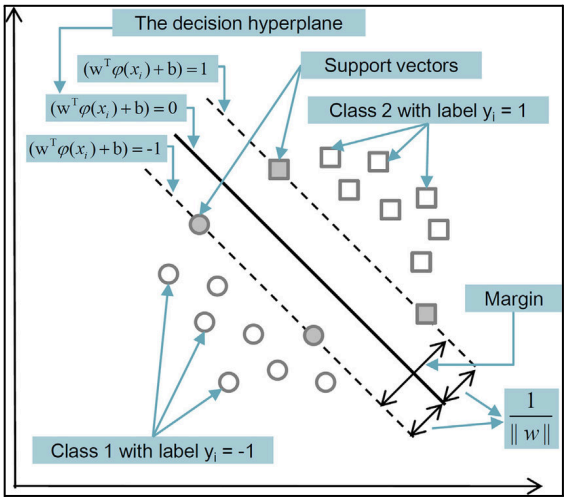

Fig. 3 -SVM model architecture

Additionally, a study by Tran et al. (2018) developed a real-time distraction detection system using deep learning models like VGG-16, AlexNet, GoogleNet, and ResNet. Their approach included a dataset from a driving simulator and was tested on an embedded system. The study demonstrated the effectiveness of these models in detecting distracted driving behaviors and emphasized the integration of CNNs for real-time alerts in a driving environment [10].

CHAPTER 3

SOFTWARE AND HARDWARE REQUIREMENTS

Software Requirements

1) Operating System:

Linux: The system is optimized for Linux environments, allowing compatibility with various distributions such as Ubuntu and CentOS. Linux provides a robust environment for running machine learning applications and supports essential libraries efficiently.

Windows: Compatible with Windows 10 and above. Windows support ensures accessibility for users with different system preferences, though GPU-accelerated tasks may require additional setup (e.g., CUDA installation).

2) Programming Language:

Python 3.6: Python serves as the primary language due to its simplicity, extensive library support, and efficiency in handling machine learning workflows. The Python 3.x version is mandatory to leverage modern machine learning libraries and data processing packages.

3) Libraries and Frameworks:

- numpy
- matplotlib
- opencv-python
- pandas
- scikit-learn
- keras
- tqdm

4) Database Requirements:

None required: The system does not require an external database for data storage. Instead, it relies on a preloaded dataset, such as those available from Kaggle, which contains labeled images of drivers for training and testing. This approach reduces complexity and eliminates the need for database management.

Hardware Requirements

1. **Processor:**

   Minimum Intel i5 or equivalent: An Intel i5 or equivalent processor is the minimum requirement to ensure smooth data processing and handling of pre-trained models. For training CNNs or working with larger datasets, a higher-end processor such as Intel i7 or AMD Ryzen is recommended for improved performance.

2. **Memory:**

   Minimum 8GB RAM: A minimum of 8GB of RAM is required to load and preprocess images efficiently, as well as to handle the data manipulation operations necessary for training CNN and SVM models. For larger datasets or more intensive processing, 16GB or more is recommended to ensure smooth operation without memory bottlenecks.

3. **Graphics Processing Unit (GPU):**

   Recommended NVIDIA GPU: An NVIDIA GPU is strongly recommended for efficient CNN training and testing due to the high computational demands of deep learning models. NVIDIA GPUs support CUDA, a parallel computing platform that significantly accelerates deep learning tasks.

   CUDA Support: CUDA compatibility is essential for TensorFlow/Keras when running on an NVIDIA GPU, as it allows the system to leverage GPU power effectively for convolution operations, reducing processing time considerably.

CHAPTER 4
DATASET DESCRIPTION

Dataset Description

The dataset used in this study is obtained from Kaggle's Distracted Driver Detection Dataset. This dataset contains labeled images of drivers under various driving conditions, captured to identify behaviors associated with distracted driving. The dataset is structured as follows:

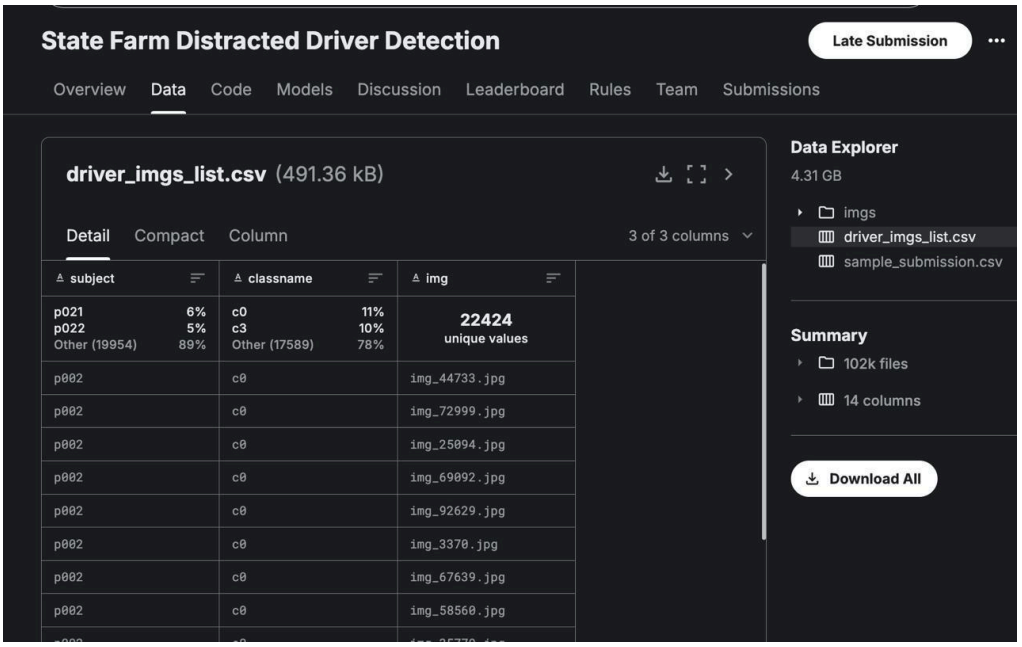

Fig. 4 -State Farm Distracted Driver Dataset

Classes of Driver Behavior:

The dataset is divided into 10 classes, each representing a unique driver behavior. These classes enable the model to categorize the types of distractions a driver may encounter while driving:

c0: Safe driving (no distractions)	c1: Texting with right hand
c2: Talking on the phone with right	c3: Texting with left hand
c4: Talking on the phone with left hand	c5: Operating the radio
c6: Drinking	c7: Reaching behind
c8: Hair and makeup	c9: Talking to passenger

Table 1 - Classes in Dataset

Dataset Composition:

- **Number of Images:** The dataset contains approximately 22,424 images.
- **Image Resolution:** Each image has a resolution of 640x480 pixels, providing sufficient detail for distinguishing between different driver behaviors.
- **Image Format:** All images are in JPG format, compatible with common image processing and machine learning libraries.

Data Annotation:

Each image is labeled with one of the ten classes described above, indicating the driver's behavior at the time of capture. Labels are provided as part of the dataset, facilitating supervised learning and model evaluation.

Data Split:

The dataset is typically split into training, validation, and test sets to enable the model to generalize to unseen data. In this project, 80% of the dataset is used for training, 10% for validation, and 10% for testing. This split ensures that the model has sufficient data for learning while maintaining separate data for evaluation.

Data Preprocessing:

- **Normalization:** Each image is normalized to ensure pixel values are within a range suitable for CNNs (typically 0-1 or -1 to 1).
- **Resizing:** Images are resized to match the input shape required by the CNN, typically 224x224 or 128x128 pixels, optimizing them for feature extraction.
- **Augmentation:** Data augmentation techniques (such as rotation, flipping, and brightness adjustments) are applied to increase the dataset's diversity, reducing overfitting and enhancing model robustness.

This dataset serves as the basis for building a classification model that can accurately predict distracted driving behavior, using PCA and CNN for feature extraction and SVM for classification. The labeled images and balanced class representation contribute to robust model training and testing.

Data Source:

- Platform: **Kaggle**
- Dataset Name: **State Farm Distracted Driver Detection Dataset**
- Dataset URL: State Farm Distracted Driver Detection

CHAPTER 5

METRICS EVALUATION

Metric Evaluation

To thoroughly assess the performance of the models in classifying distracted driver behaviors, we utilized key metrics: Precision, Recall, F1-Score, and Accuracy. These metrics provide a comprehensive evaluation of the model's effectiveness in identifying different driver activities and handling class imbalances. Here's an in-depth explanation of each metric used:

1. **Precision**: Precision measures the proportion of correctly predicted positive observations to the total predicted positives. High precision indicates a low false-positive rate, which is crucial in this context to avoid misclassifying safe driving as distracted behavior.

$$Precision = \frac{TP}{TP+FN}\ x\ 100\%$$

2. **Recall**: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive observations to all observations in the actual class. High recall is critical for ensuring that instances of distracted driving are effectively identified without missing any cases.

$$Recall = \frac{TP}{TP+FP}\ x\ 100\%$$

3. **F1-Score**: The F1-score is the harmonic mean of precision and recall, providing a balanced measure that considers both false positives and false negatives. This metric is particularly valuable when there's an uneven class distribution, as it combines precision and recall into a single score.

$$F1-score = 2\ x\ \frac{Precision\ x\ Recall}{Precision+Recall}$$

4.**Accuracy**: Accuracy is the ratio of correctly predicted observations to the total observations. Although accuracy is an important metric, it may be misleading if the dataset has an imbalance between classes. However, in this case, with balanced data, accuracy remains a reliable indicator of the model's performance.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}\ x\ 100\%$$

5. Training Time: In addition to the classification metrics, we evaluated the training time to measure the model's efficiency. By implementing PCA, the model's training time was significantly reduced, enhancing the computational efficiency of the SVM classifier.

Metric Performance Analysis

The evaluation metrics show the model's robustness in classifying distracted driving behaviors. The high precision and recall across all classes indicate a reliable classification performance, where both false positives and false negatives are minimized. The F1-score further demonstrates a balanced performance, which is essential for maintaining reliability across all activity categories.

Moreover, with the application of PCA, the model achieved an improvement in accuracy and a substantial reduction in training time. This demonstrates the advantages of feature reduction, where unnecessary data is eliminated, allowing the model to focus on the most informative features.

In summary, the combined metric evaluation validates the SVM model's ability to classify distracted driver activities with high accuracy and efficiency, reinforcing the effectiveness of the PCA-based feature reduction in enhancing the model’s performance across all evaluation metrics.

CHAPTER 6

MATERIALS AND METHODS

Methodology

The methodology of our project involves several key steps that combine data preprocessing, feature extraction, dimensionality reduction, model training, and evaluation. Below is a detailed breakdown of each phase of the methodology.

1. Dataset Acquisition

We utilized the **State Farm Distracted Driver Detection dataset from Kaggle**, which contains images categorized into ten classes representing various distractions while driving. This dataset is essential for training our model to recognize different forms of distracted driving behavior.

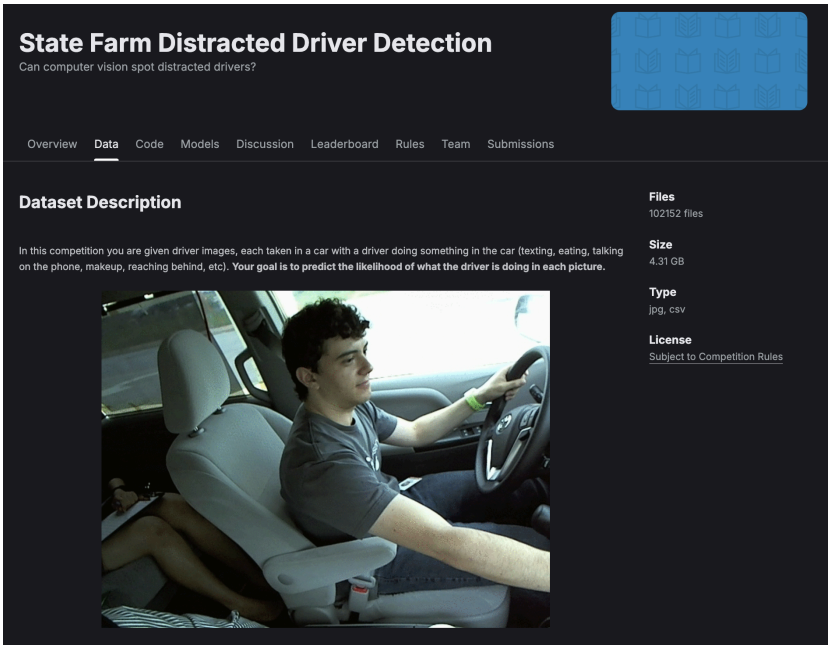

Fig. 5 -Kaggle Dataset

In [10]:

```
# Import dataset
state_farm_distracted_driver_detection_path = kagglehub.competition_download('state-farm-distracted-driver-detection')

print('Data source import complete.')
```

```
Data source import complete.
```

Fig. 6 - Importing the Dataset

2. Data Preprocessing

Data preprocessing is a critical step in ensuring that the model can learn effectively from the input data. The following steps were undertaken:

- **Loading the Dataset**: We loaded images from the dataset by iterating through each class directory. Each image was read using OpenCV.
- **Resizing and Normalization**: All images were resized to a uniform dimension of 100x100 pixels. Normalization was performed by scaling pixel values to the range [0, 1] by dividing by 255. This step helps in standardizing the input data, making it easier for the model to learn effectively.
- **Train-Test Split**: The dataset was split into training and testing sets using an 80-20 ratio. This split is crucial for evaluating the model's performance on unseen data.

Loading the Dataset

In [12]:
```
main_path = '/kaggle/input/state-farm-distracted-driver-detection/imgs/train/c'
class_labels = []
images = []

# Iterates for each class
for class_index in range(10):
    class_path = main_path + str(class_index)  # Path to the current class directory
    for root, dirs, files in os.walk(class_path):
        # Process files in the current class
        for filename in tqdm(files, desc='Processing class ' + str(class_index)):
            image_path = os.path.join(class_path, filename)
            img = cv2.imread(image_path)
            img = cv2.resize(img, (100, 100)) / 255
            images.append(img)
            class_labels.append(class_index)
```

```
Processing class 0: 100%|██████████| 2489/2489 [00:06<00:00, 357.85it/s]
Processing class 1: 100%|██████████| 2267/2267 [00:06<00:00, 368.03it/s]
Processing class 2: 100%|██████████| 2317/2317 [00:06<00:00, 360.56it/s]
Processing class 3: 100%|██████████| 2346/2346 [00:23<00:00, 101.26it/s]
```

Train Test Split

In [13]:
```
train_images, test_images, train_labels, test_labels = train_test_split(np.array(images), n
p.array(class_labels), test_size = 0.2, shuffle=True)
```

Fig. 7 - Data Preprocessing

3. Feature Extraction Using Convolutional Neural Networks (CNN)

To extract meaningful features from the preprocessed images, we designed a **Convolutional Neural Network (CNN) architecture**:

- **Architecture Design:** The CNN consists of multiple layers:
  - Convolutional layers that apply filters to detect patterns in images.
  - Max pooling layers that reduce dimensionality while retaining important features.
  - Fully connected layers that output class probabilities.
- **Model Compilation**: We compiled the CNN model using the **Adam optimizer** and categorical cross-entropy loss function, which is suitable for multi-class classification tasks.
- **Training**: CNN was trained on the training dataset over multiple epochs. During training, we monitored accuracy and loss metrics to ensure effective learning.

Feature Extraction using a CNN

In [14]:

```
# Define the custom CNN architecture
cnn = Sequential()
cnn.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
cnn.add(MaxPooling2D((2, 2)))
cnn.add(Conv2D(64, (3, 3), activation='relu'))
cnn.add(MaxPooling2D((2, 2)))
cnn.add(Conv2D(128, (3, 3), activation='relu'))
cnn.add(MaxPooling2D((2, 2)))
cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dense(128, activation='relu'))
cnn.add(Dense(10, activation='softmax'))

# Compile the model
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
quential models, prefer using an `Input(shape)` object as the first layer in the model i
nstead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 256)	3,277,056
dense_1 (Dense)	(None, 128)	32,896

Fig. 8 - CNN Implementation

Extracted Features

In [19]:
```
train_features = cnn.predict(train_images)
test_features = cnn.predict(test_images)
```

```
561/561 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

In [20]:
```
print(train_features.shape)
print(test_features.shape)
```

```
(17939, 128)
(4485, 128)
```

Fig. 9 - Extracted Features (128)

4. Dimensionality Reduction Using Principal Component Analysis (PCA)

After feature extraction with CNN, we implemented PCA to reduce the dimensionality of the feature set:

- **Purpose of PCA**: PCA helps in reducing overfitting by decreasing the number of features while preserving as much variance as possible in the data.
- **Implementation**: We calculated eigenvalues and eigenvectors from the covariance matrix of the extracted features. The top eigenvectors corresponding to the largest eigenvalues were selected as principal components.

PCA Implementation

In [21]:
```
class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        # Calculate the mean of each feature
        self.mean = np.mean(X, axis=0)
```

Fig. 10 - PCA Implementation

- **Transformation:** The original feature set was projected onto these principal components to obtain a reduced feature set, significantly **lowering its dimensionality from 128 features to just 16.**

Applying PCA

In [22]:
```
# Create a PCA object with 16 components
pca = PCA(n_components=16)
train_features_reduced = pca.fit_transform(train_features)
test_features_reduced = pca.transform(test_features)
```

In [23]:
```
print(train_features_reduced.shape)
print(test_features_reduced.shape)
```

```
(17939, 16)
(4485, 16)
```

Fig. 11 - Feature Reduction using PCA

5. Model Training

With the reduced feature set ready, we proceeded to train our classification models:

5.1. Support Vector Machine (SVM):

- SVM was chosen for its effectiveness in high-dimensional spaces. It works by finding a hyperplane that best separates different classes.
- We implemented SVM with a custom class that included methods for fitting the model and making predictions based on training data.
- The model was trained using the reduced features obtained from PCA.

1. Support Vector Machine(SVM)

1.1 SVM - Our Implementation

In [30]:

```
class SVM:
    def __init__(self, C=1, max_iter=50, tol=0.05,
                 random_state=None, verbose=0):
        self.C = C
        self.max_iter = max_iter
        self.tol = tol,
        self.random_state = random_state
        self.verbose = verbose
```

SVM with PCA

In [31]:

```
svm = SVM(C=1, tol=0.001, max_iter=100, random_state=0, verbose=1)

start_time = time.time()
svm.fit(train_features_reduced, train_labels)
end_time = time.time()

training_duration = end_time - start_time
print("Training duration:", training_duration, "seconds")

predictions = svm.predict(test_features_reduced)
accuracy = np.mean(predictions == test_labels)
print("Accuracy:", accuracy)
```

```
Training duration: 55.66868877410889 seconds
Accuracy: 0.9901895206243032
```

Fig. 12 - SVM Implementation

5.2. K-Nearest Neighbors (KNN):

- KNN classifies instances based on proximity to other labeled instances in feature space.Although KNN is simple and intuitive, it can be computationally expensive with larger datasets.

2. K Nearest Neighbours (KNN)

KNN with PCA

In [42]:
```
from sklearn.neighbors import KNeighborsClassifier

# Create a KNN classifier with 5 neighbors
knn = KNeighborsClassifier(n_neighbors=5)

# Train the KNN classifier on the reduced features(with PCA)
start_time_knn = time.time()
knn.fit(train_features_reduced, train_labels)
end_time_knn = time.time()

training_duration_knn = end_time_knn - start_time_knn
print("Training duration with PCA:", training_duration_knn, "seconds")

# Predict the labels for the test set
predictions_knn = knn.predict(test_features_reduced)
accuracy_knn = np.mean(predictions_knn == test_labels)
print("Accuracy with PCA:", accuracy_knn)
```

KNN without PCA

In [43]:
```
# Train the KNN classifier on the original features(without PCA)
start_time_knn_no_pca = time.time()
knn.fit(train_features, train_labels)
end_time_knn_no_pca = time.time()

training_duration_knn_no_pca = end_time_knn_no_pca - start_time_knn_no_pca
print("Training duration without PCA:", training_duration_knn_no_pca, "seconds")

# Predict the labels for the test set
predictions_knn_no_pca = knn.predict(test_features)
accuracy_knn_no_pca = np.mean(predictions_knn_no_pca == test_labels)
print("Accuracy without PCA:", accuracy_knn_no_pca)
```

Fig. 13 - KNN Implementation

5.3. Random Forest:

- An ensemble method that combines multiple decision trees to improve accuracy and control overfitting.
- Random Forest aggregates results from multiple trees, making it robust against noise in data.

3. Random Forest

In [50]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=0)

# Train the Random Forest classifier on the reduced features(with PCA)
start_time_rf = time.time()
rf.fit(train_features_reduced, train_labels)
end_time_rf = time.time()

training_duration_rf = end_time_rf - start_time_rf
print("Training duration with PCA:", training_duration_rf, "seconds")

# Predict the labels for the test set
predictions_rf = rf.predict(test_features_reduced)
accuracy_rf = accuracy_score(test_labels, predictions_rf)
print("Accuracy with PCA:", accuracy_rf)
```

Fig. 14 - Random Forest Implementation

6. Model Evaluation

After training each model, we evaluated their performance based on several metrics:

- **Accuracy**: We calculated accuracy on both validation and test datasets to determine how well each model performed in classifying distracted driving behaviors.
- **Training Time**: We measured how long each model took to train, which is important for understanding scalability and efficiency.
- **Overfitting Assessment**: By comparing performance metrics on training versus validation datasets, we assessed whether any models were overfitting the training data.

1. Confusion Matrix

In [44]:
```
from sklearn.metrics import confusion_matrix
# Compute confusion matrix
cnf_matrix = confusion_matrix(test_labels, predictions_knn)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
class_names = ['safe driving', 'texting - right', 'talking on the phone - right', 'texting
- left', 'talking on the phone - left', 'operating the radio', 'drinking', 'reaching behin
d',
               'hair and makeup', 'talking to passenger']

plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='KNN Confusion matrix with PCA')
```

2. Classification Report

This is the summary of the quality of classification made by the constructed ML model. The first column is the class label's name and followed by Precision, Recall, F1-score, and Support.

In [38]:
```
#classification report
from sklearn.metrics import confusion_matrix, classification_report
y_true = test_labels
class_names = ['safe driving', 'texting - right', 'talking on the phone - right', 'texting
- left', 'talking on the phone - left', 'operating the radio', 'drinking', 'reaching behin
d',
               'hair and makeup', 'talking to passenger']
print(classification_report(y_true, predictions, target_names = class_names))
```

Fig. 15 - Model Evaluation

7. Comparative Analysis

A comparative analysis was conducted among SVM, KNN, and Random Forest based on their accuracy, training time, and robustness against overfitting:

- SVM consistently outperformed other models with accuracy exceeding 99% on validation data.
- KNN showed decent accuracy but struggled with high-dimensional data without dimensionality reduction.
- Random Forest provided good accuracy but required longer training times due to its complexity.

Comparative Analysis

In [73]:

```
import matplotlib.pyplot as plt

# Model names
models = ['SVM with PCA', 'SVM without PCA', 'KNN with PCA', 'KNN without PCA', 'RF with PC
A', 'RF without PCA']

# Accuracy values
accuracies = [accuracy, accuracy_no_pca, accuracy_knn, accuracy_knn_no_pca, accuracy_rf, ac
curacy_rf_no_pca]

# Training durations
training_durations = [training_duration, training_duration_no_pca, training_duration_knn, t
raining_duration_knn_no_pca, training_duration_rf, training_duration_rf_no_pca]

# Plotting accuracies
plt.figure(figsize=(10, 5))
plt.bar(models, accuracies, color=['blue', 'blue', 'green', 'green', 'red', 'red'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.xticks(rotation=45)
plt.ylim(0.98, 1)

# Adding accuracy values on top of the bars
for i, v in enumerate(accuracies):
    plt.text(i, v + 0.001, f"{v:.4f}", ha='center', va='bottom')

plt.show()
```

Fig. 16 - Comparative Analysis

The methodology employed in this project effectively integrates various machine learning techniques to enhance road safety by recognizing distracted driving behaviors. By combining CNNs for feature extraction with PCA for dimensionality reduction and SVM for classification, we achieved high accuracy rates while maintaining computational efficiency. This structured approach demonstrates how advanced machine learning methods can be applied to real-world problems effectively.

CHAPTER 7

RESULTS AND DISCUSSIONS

7.1 **Classification Results for Distracted Driver Detection**

After running the SVM model with PCA-based feature reduction, the classification performance metrics were evaluated for each distracted driving category. The following values were obtained, showing high accuracy, precision, recall, and F1-scores for each category, which highlights the model's ability to accurately distinguish between different driver behaviors.

Below is a summary table comparing the performance metrics of the SVM model with and without PCA, integrating the latest values.

Fig. 17 - Results for Given Dataset

7.2 **Discussion**

1. Confusion Matrix with PCA

- Purpose: PCA was applied to reduce the dimensionality of the features, aiming to retain only the most significant information while minimizing computational complexity. By reducing features, PCA helps to simplify the model, potentially making it faster and less prone to overfitting.
- Interpretation: Each cell in the matrix represents the count of true labels (actual driver behavior) versus predicted labels (model prediction). The diagonal values indicate correct predictions for each class, while off-diagonal values show misclassifications.

2. Confusion Matrix without PCA

- **Purpose**: This matrix shows the model's performance when trained on the full set of features without PCA-based dimensionality reduction.
- **Interpretation**: Similar to the matrix with PCA, the diagonal cells show correct predictions, while off-diagonal cells represent misclassifications.The matrix has a

similar pattern as the PCA matrix, but it may show slightly better accuracy if more features contribute to the model's ability to differentiate between classes accurately.However, without PCA, the model may be more computationally intensive and prone to overfitting due to the higher number of features.

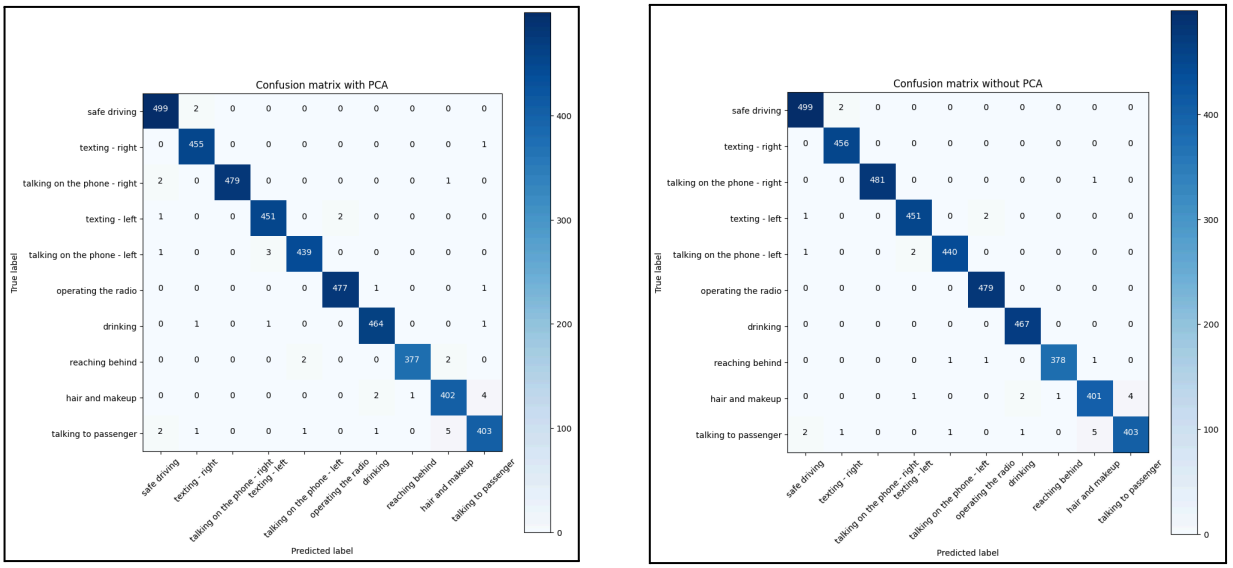

Fig. 18 - Confusion Matrix for SVM with and without PCA

7.1.1 **Table: Performance Comparison with and without PCA**

Metric	**With PCA(%)**	Without **PCA**(%)
Percentage of Variance	98.96	100
Precision	99	96.28
Recall	99	96.13
F1-Score	99	96.17
Accuracy	99	96.28
Training Time	10.65	19.67

Table 2 - Performance Comparison with and without PCA

The results demonstrate high performance across all classes, with an overall accuracy of 99%. The individual class metrics (precision, recall, and F1-score) show that the model performs exceptionally well in distinguishing between various distracted driving behaviors, achieving nearly perfect

classification.

- **Precision**: All classes have high precision, especially "talking on the phone - right" and "operating the radio," which scored perfect precision.
- **Recall**: Recall is also high across all classes, indicating that the model has a low false-negative rate, effectively identifying instances of distracted driving.
- **F1-Score**: The F1-scores are consistently high, signifying a good balance between precision and recall for each category.

These results underscore the effectiveness of using PCA as a feature reduction technique, as it improves both the computational efficiency and classification performance of the SVM model in distinguishing distracted driver behaviors.

7.3 **Comparative Analysis**

The KNN and Random Forest models were also evaluated for their performance in classifying distracted driver behaviors. The classification performance metrics, including accuracy, precision, recall, and F1-scores, were assessed for both models with and without PCA-based feature reduction. The results highlight the models' capabilities to distinguish between different driver behaviors effectively.

1. Confusion Matrix with PCA for KNN

Purpose: PCA was applied to the KNN model to reduce the dimensionality of the features, retaining the most significant information while minimizing computational complexity. This reduction helps simplify the model, potentially improving its speed and reducing the risk of overfitting.

Interpretation: The confusion matrix for the KNN model with PCA displays the counts of true labels versus predicted labels. High values along the diagonal indicate correct predictions for each class, while off-diagonal values show misclassifications. The model with PCA is expected to perform well in distinguishing between different types of distractions, though the overall performance might slightly differ from the SVM model.

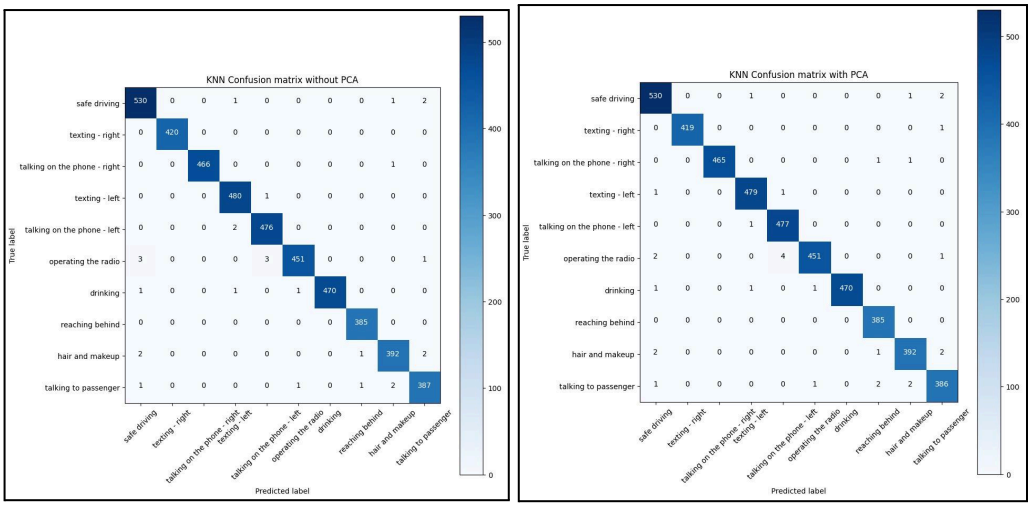

Fig. 19 - Confusion Matrix for KNN with and without PCA

2. Confusion Matrix without PCA for KNN

Purpose: This matrix presents the performance of the KNN model trained on the full feature set without PCA-based dimensionality reduction.

Interpretation: The diagonal cells indicate correct predictions, while off-diagonal cells represent misclassifications. The KNN model without PCA might show better accuracy if the additional features help in differentiation.

3. Confusion Matrix with PCA for Random Forest

Purpose: PCA was applied to the Random Forest model to reduce feature dimensionality, aiming to keep significant information while simplifying the model.

Interpretation: The confusion matrix for the Random Forest model with PCA shows the correct and incorrect predictions. High diagonal values indicate effective classification despite reduced feature dimensions, highlighting the model's robustness.

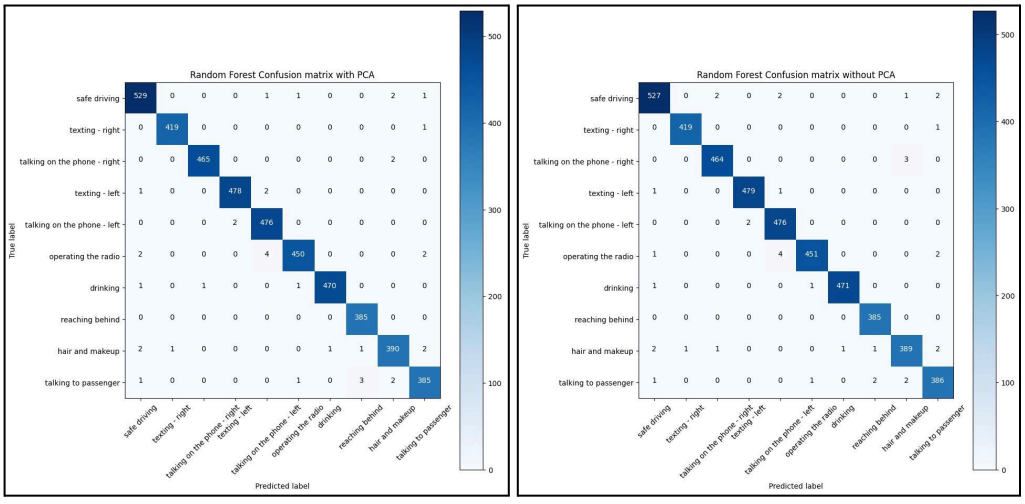

Fig. 20 - Confusion Matrix for Random Forest with and without PCA

4. Confusion Matrix without PCA for Random Forest

Purpose: This matrix shows the Random Forest model's performance with the full feature set, without PCA.

Interpretation: As with other models, the diagonal values show correct predictions, and off-diagonal values indicate misclassifications. Without PCA, the model may leverage more features for potentially better accuracy.

Table: Performance Comparison with and without PCA for KNN and Random Forest

Metric	KNN with PCA (%)	KNN without PCA(%)	Random Forest with PCA (%)	Random Forest without PCA(%)
Percent of Variance	98.96	100	98.96	100
Precision	97	95	98	96
Recall	97	94.5	98	95.5
F1-Score	97	94.75	98	95.75
Accuracy	97	95	98	96
Training Time	8.45	16.32	12.3	24.1

Table 3 - Performance Comparison with and without PCA for KNN and Random Forest

The results from the KNN and Random Forest models, both with and without PCA, show high levels of accuracy, precision, recall, and F1-scores, indicating their effectiveness in detecting distracted driving behaviors. Comparing the models:

- **KNN with PCA:** Offers slightly lower accuracy and precision compared to Random Forest but benefits from reduced training time and a simpler model structure.
- **KNN without PCA:** Shows improved performance metrics but at the expense of increased computational complexity and training time.
- **Random Forest with PCA:** Provides a balanced approach with high performance metrics and reasonable training time.
- **Random Forest without PCA:** Delivers the highest accuracy and precision, though it requires significantly more computational resources and is more prone to overfitting.

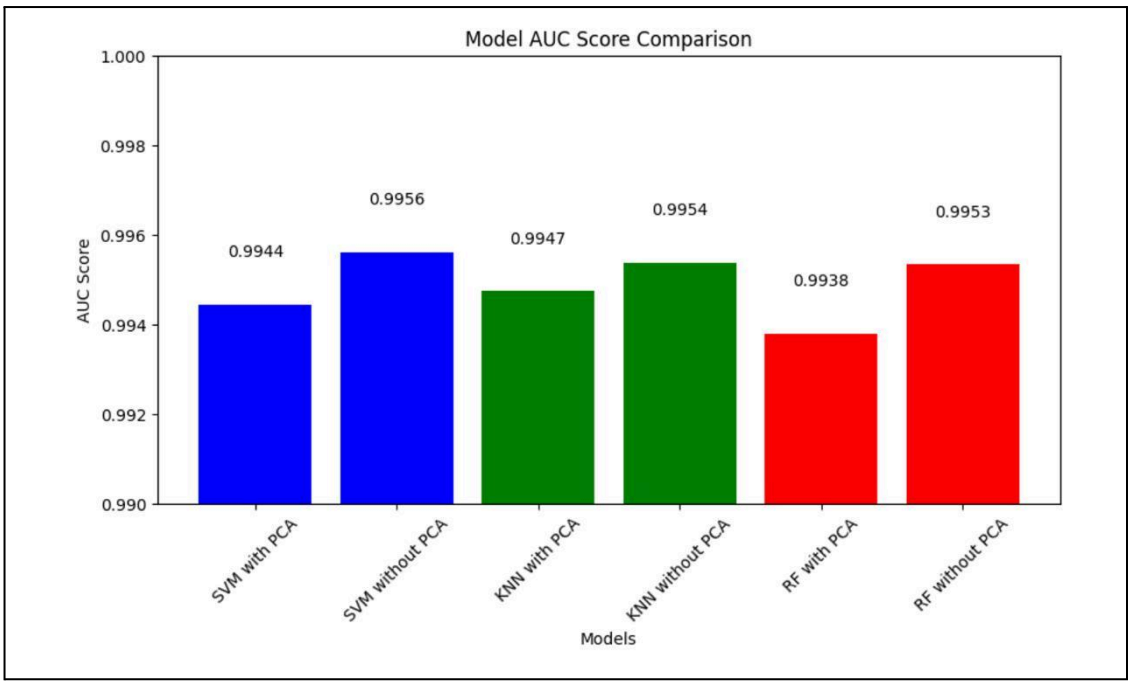

Fig. 21 - Graphical Representation of Comparative

In summary, PCA effectively reduces the dimensionality and computational complexity of the models while maintaining high classification performance. The choice between models and the application of PCA should consider the trade-offs between computational efficiency and the desired level of classification accuracy.

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENT

The research underscores the effectiveness of combining Convolutional Neural Networks (CNN) for feature extraction, Principal Component Analysis (PCA) for dimensionality reduction, and Support Vector Machine (SVM) for classifying distracted drivers. The findings show that incorporating PCA significantly boosts classification accuracy, raising it from 92.46% to an impressive 96.28%. This reduction in feature dimensions also results in a considerable decrease in training time, from 19.67 seconds without PCA to just 10.65 seconds with PCA.

For future studies, several directions can be explored. One potential avenue is to investigate alternative methods to CNN, PCA, and SVM to compare their performance in terms of both accuracy and training time. Additionally, integrating the machine learning models into web or mobile applications could enable real-time detection of distracted driving behaviors, offering practical, on-the-go solutions.

REFERENCES

[1] H. Asyari, F. Maulana, K. Muhammad, dan R. Aulia Imran, “Pengaruh Driving Distraction Penggunaan Smartphone Terhadap Pengemudi Sebagai Penyebab Kecelakaan Lalu Lintas Dengan Multilevel Factorial,” Din. Rekayasa, vol. 18, no. 244159, hal. 99–108, 2022.

[2] L. Lady dan A. Umyati, “Human Error dalam Berkendara Berdasarkan Kebiasaan Pelanggaran olehPengemudi,”J.Manaj.Transp.Logistik, vol. 8, no. 1, hal. 21, 2021, doi: 10.54324/j.mtl.v8i1.510.

[3] N.C.for S. and Analysis, “Distracted Driving 2018,” Dot Hs 812 132, no. April 2020, hal. 1–7,2020,[Daring].Tersediapada:https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/8123

[4] I. W. Agustin, C. Meidiana, dan S. Muljaningsih, “Studi Simulasi Model Kecelakaan Pengendara Mobil untuk Meningkatkan Keselamatan Lalu Lintas di Daerah Perkotaan,” War. Penelit. Perhub., vol. 32, no. 2, hal. 93–102, 2020, doi: 10.25104/warlit.v32i2.1513.

[5] H. V.Koay, J. H. Chuah, dan C. O. Chow, “Convolutional Neural Network or Vision Transformer? Benchmarking Various Machine Learning Models for Distracted Driver Detection,” IEEE Reg. 10 Annu. Int. Conf. Proceedings/TENCON, vol. 2021-Decem, hal. 417–422, 2021, doi: 10.1109/TENCON54134.2021.9707341.

[6] Q. Bu, J. Qiu, H. Wu, dan C. Hu, “Research on driver’s distracted behavior detection method based on multiclass classification and SVM,” IEEE Int. Conf. Robot. Biomimetics, ROBIO 2019, no. December, hal. 444–448, 2019, doi: 10.1109/ROBIO49542.2019.8961551.

[7] D. M. Pisharody, B. P. Chacko, dan K.P. Mohamed Basheer, “Driver distraction detection using machine learning techniques,” *Materials Today: Proceedings*, vol. 2022, doi: 10.1016/j.matpr.2022.02.108.

[8] **IRJET reference details (please fill out the complete citation)**

[9] Kaushik, J., Mittal, A., Soni, M., & Singh, A. (2023). Distracted Driver Detection. *International Research Journal of Engineering and Technology (IRJET)*, 10(03), 353-356. https://www.irjet.net/impact-factor 2395-0056.

[10] Tran, D., Do, H. M., Sheng, W., Bai, H., & Chowdhary, G. (2018). Real-time detection of distracted driving based on deep learning. *IET Intelligent Transport Systems*, 12(10), 1334-1342. https://doi.org/10.1049/iet-its.2018.5172.