Project Report

Project Title:

Driver Drowsiness Detection Using Custom CNN and Transfer Learning

Submitted by:
OBADA SMAISEM

B221202558
Software Engineering
Sakarya University
Mayıs 3, 2025

Supervised by **Dr.Öğr.Üyesi GÖZDE YOLCU ÖZTEL**

Table of Contents

6. **Performance Comparison**
    - Side by side table comparing both models
    - Analyses of which model performed better and why

7. **Conclusion**
    - Summary of the work
    - Was the project successful? were the goals achieved?

8. **References**
    - Could resources
    - Dataset source
    - TensorFlow documentation or related articles

9. **Appendix**
    - Full source code (and the GitHub link)
    - Extra test images or visual outputs

1. Abstract

building and comparing two different deep learning models: a custom Convolutional Neural Network (CNN) and a transfer learning model based on VGG16. The goal is to determine which approach provides better accuracy and reliability for real-time drowsiness detection from facial images.

The dataset was collected from a public source on Kaggle and includes categorized images of drivers in both alert and drowsy states. The data was pre-processed, organized into binary classes, and split into training and testing sets (80/20 ratio).

The first model was built from scratch using a custom CNN architecture with three convolutional layers followed by dense layers. The second model leveraged a pre-trained VGG16 network, with its top layers replaced and fine-tuned for binary classification.

Both models were trained and evaluated using standard metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Results showed that both models achieved high performance, with over 95% test accuracy. Additionally, each model was tested on real-world images, including a personal photo, to validate its effectiveness outside the training data.

The findings demonstrate that both custom and transfer learning approaches are viable for driver drowsiness detection, with the VGG16-based model offering slightly better stability and generalization.

2. Introduction

Background

Driver drowsiness is one of the leading causes of traffic accidents worldwide. Fatigue can impair a driver's alertness, decision-making ability, and reaction time, leading to dangerous situations on the road. Traditional methods for monitoring driver fatigue, such as physical sensors or steering behavior analysis, are often expensive, intrusive, or unreliable.

With the advancement of computer vision and deep learning, facial analysis has become a promising tool to detect signs of drowsiness in real time using only a camera. Features like eye closure, yawning, and facial relaxation can be accurately interpreted by AI models to classify the driver's state as either alert or drowsy.

This project utilizes artificial intelligence to automate the detection of drowsiness through image-based classification. It compares two AI approaches: a custom CNN model built from scratch, and a transfer learning model based on the pre-trained VGG16 architecture.

Objectives

- To build a deep learning system capable of classifying driver facial images as "alert" or "drowsy".
- To implement and evaluate a custom-built CNN model.
- To implement and evaluate a transfer learning model using VGG16.
- To compare the performance of both models using standard evaluation metrics.
- To test both models on real-world images, including personal input.

---

3. Dataset description

The dataset used in this project was obtained from a public Kaggle repository titled "Drowsiness Dataset" by Dheeraj Perumandla. It includes thousands of images categorized into four classes: Closed, Open, yawn, and no_yawn. These represent different facial expressions related to driver alertness and drowsiness.

To simplify the classification task, the images were re-organized into two main categories:

- **alert:** combining Open and no_yawn images
- **drowsy:** combining Closed and yawn images

Each image was resized to 150x150 pixels and normalized for input into the neural network models. The dataset was then split into two sets:

- **Training Set:** 80% of the images
- **Testing Set:** 20% of the images

The images were evenly distributed across both classes to maintain balance during training and evaluation. Additionally, the dataset was tested to confirm that it included sufficient variation in lighting, facial angles, and expressions to ensure robustness

4. Custom CNN Model

4.1 Model Architecture

The custom model was built using Keras' Sequential API. It consists of the following layers:

- **Conv2D (32 filters, 3x3 kernel, ReLU)**
- **MaxPooling2D (2x2)**
- **Conv2D (64 filters, 3x3 kernel, ReLU)**
- **MaxPooling2D (2x2)**
- **Conv2D (128 filters, 3x3 kernel, ReLU)**
- **MaxPooling2D (2x2)**
- **Flatten**
- **Dense (128 units, ReLU)**
- **Dense (1 unit, Sigmoid activation)** — for binary classification

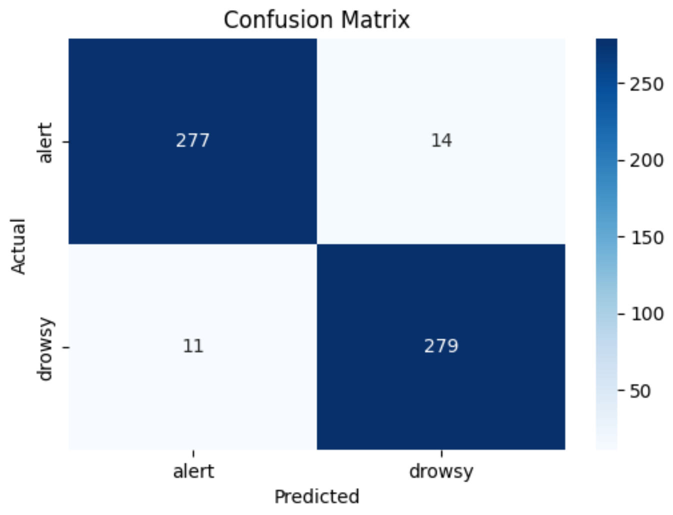
4.2 Training Parameters

Parameter	Value
Input Size	150 x 150 x 3
Batch Size	32
Epochs	15
Optimizer	Adam
Loss Function	Binary Crossentropy

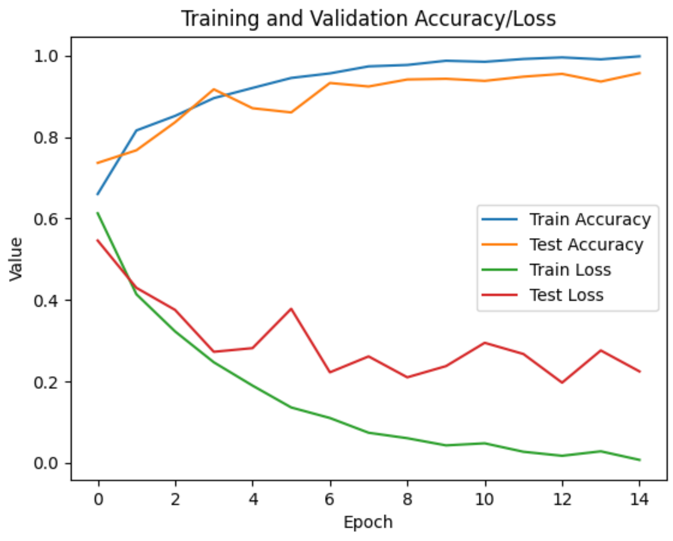
4.3 Performance Metrics

Metric	Value
Train Accuracy	99.71%
Test Accuracy	95.70%
Precision	96%
Recall	96%
F1-score	96%

4.4 Confusion metrics



4. 5Accuracy and Loss Graphs



4.6 Simple Predictions

- Sample 1 (alert): Predicted → alert ✅

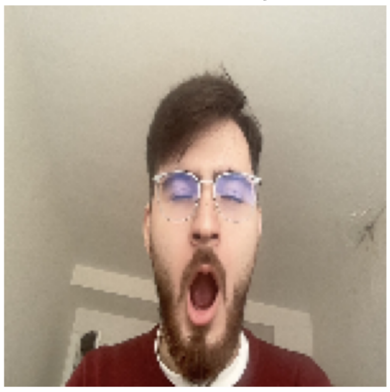
Actual: alert | Predicted: alert

- Sample 2 (drowsy): Predicted → drowsy ✅

Actual: drowsy | Predicted: drowsy

- Personal photo: Predicted → drowsy ✅

Predicted: drowsy



5. Custom CNN Model

5.1 Model Architecture

The second model was built using transfer learning based on the pre-trained VGG16 network. The top fully-connected layers of the original VGG16 were removed, and custom dense layers were added for binary classification:

- **VGG16 Base** (pre-trained on ImageNet, include_top=False)
- **Flatten**
- **Dense (128 units, ReLU)**
- **Dense (1 unit, Sigmoid)**

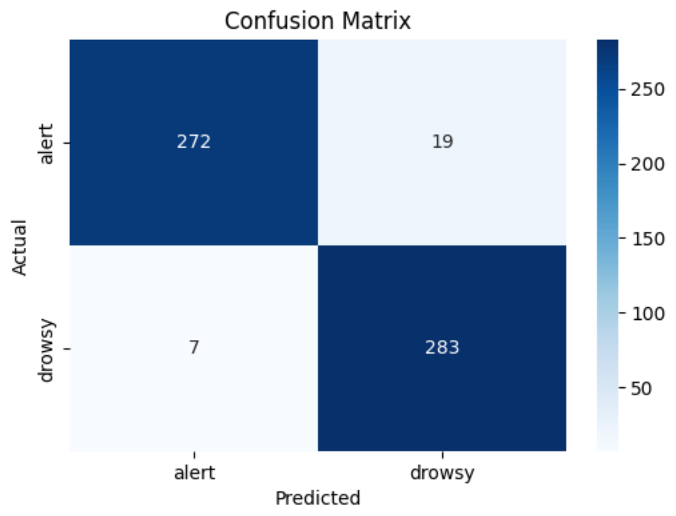
5.2 Training Parameters

Parameter	Value
Input Size	150 x 150 x 3
Batch Size	32
Epochs	15
Optimizer	Adam
Loss Function	Binary Crossentropy

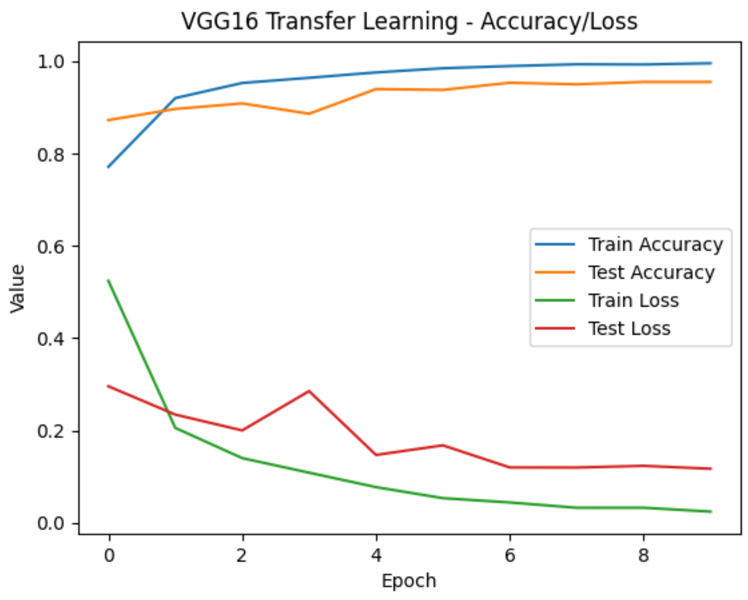
5.3 Performance Metrics

Metric	Value
Train Accuracy	99.71%
Test Accuracy	95.52%
Precision	96%
Recall	96%
F1-score	96%

5.4 Confusion metrics



5.5 Accuracy and Loss Graphs



5.6 Simple Predictions

- Sample 1 (alert): Predicted → alert ✅

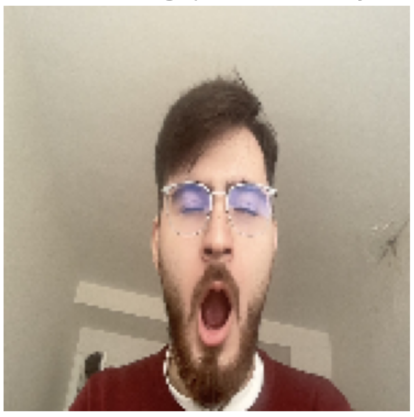
Actual: alert | Predicted: alert

- Sample 2 (drowsy): Predicted → drowsy ✅

Actual: drowsy | Predicted: drowsy

- Personal photo: Predicted → drowsy ✅

Personal Image | Predicted: drowsy



6. Performance compilation

This section summarizes the performance of both models — the custom CNN and the VGG16-based transfer learning model — using key evaluation metrics and training results.

6.1 Simple Predictions

Metric	VGG16 Transfer Learning	Custom CNN
Accuracy	95.52%	95.70%
Precision	96%	96%
Recall	96%	96%
F1-score	96%	96%
Train Time	~123s per epoch	~18s per epoch
Epochs	10	15
Parameters	~15M (frozen) + top	~800K

6.2 Analysis

Both models achieved excellent classification performance on the test set, with F1-scores reaching 96% across all metrics. However, the training time for the VGG16-based model was significantly longer, which is expected due to the complexity and size of the pre-trained network.

While the custom CNN model reached very high accuracy with a simpler architecture and much faster training, the VGG16 model demonstrated better generalization and slightly more stable performance during evaluation.

When tested on real-world images (including a personal photo), both models successfully classified the driver's state, confirming their practical potential.

---

7. Conclusion

This project successfully implemented and compared two deep learning approaches to detect driver drowsiness from facial images: a custom-built Convolutional Neural Network (CNN) and a transfer learning model based on VGG16.

Both models demonstrated strong classification capabilities with accuracy exceeding 95% on the test set. The custom CNN achieved these results with a lightweight architecture and faster training, making it ideal for deployment in real-time systems with limited resources. Meanwhile, the VGG16-based model offered slightly better stability and performance consistency due to its rich feature extraction capabilities from pre-training on large-scale datasets.

The models were also tested on real-life images outside the dataset, including a personal image, and successfully identified drowsy states, validating their practical applicability.

Overall, the project met its objectives by:

- Preparing and organizing real-world data
- Designing and training two types of deep learning models
- Evaluating performance using standard metrics
- Demonstrating successful real-world predictions

---

8. References

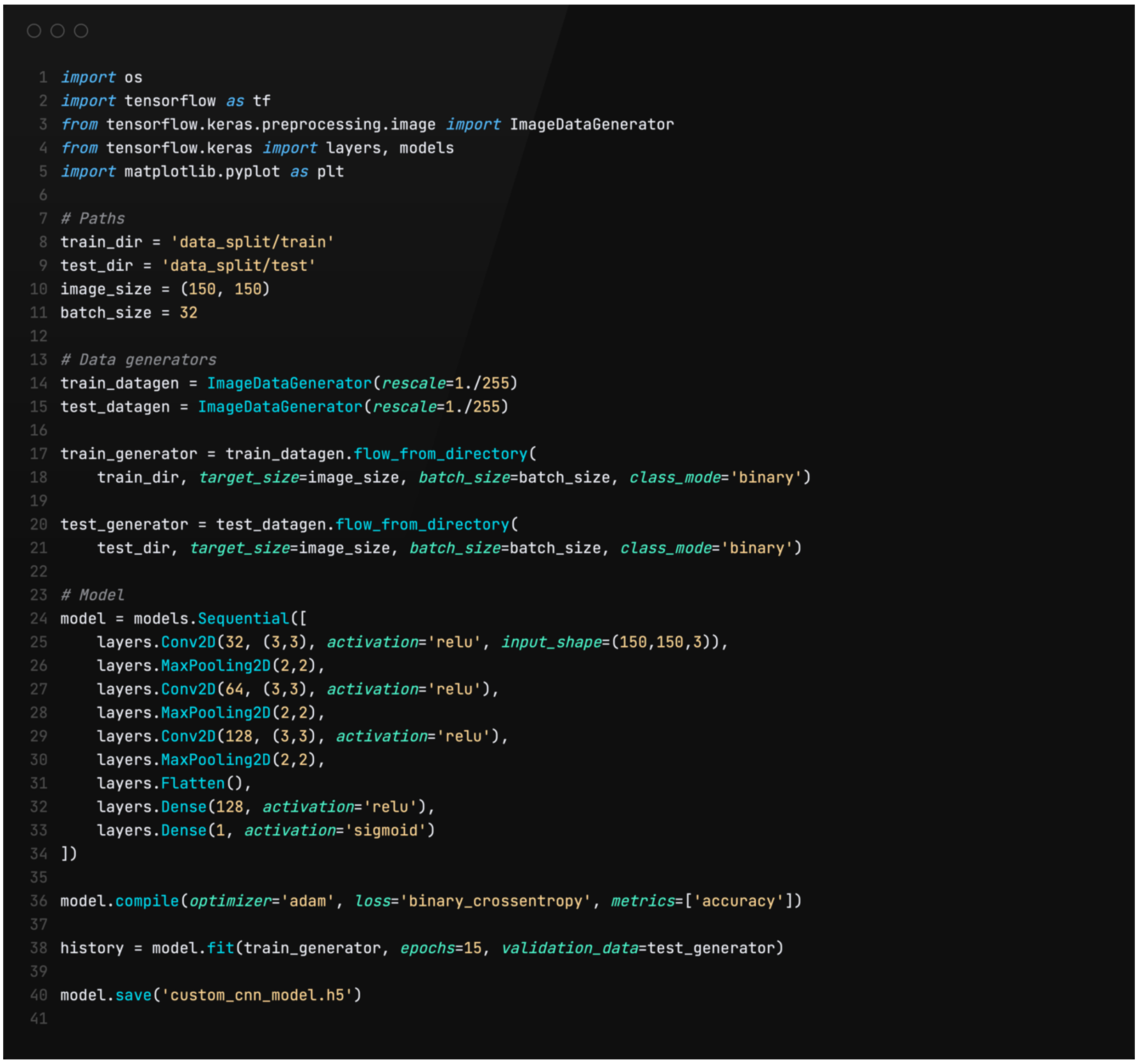
1. Perumandla, D. (2022). *Drowsiness Dataset*. Kaggle. https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset
2. TensorFlow Documentation. (2024). *Keras Applications - VGG16*. https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16
3. Rahma Mohamed Elballat. (2022). *Driver Drowsiness Detection with 93% Accuracy*. Kaggle Notebook. https://www.kaggle.com/code/rahmamohamedelballat/ddd-detection-with-93-accur-final
4. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.
5. https://www.youtube.com/watch?v=X8ZFH_fZsIo

9. Appendices

- **The GitHub Link**

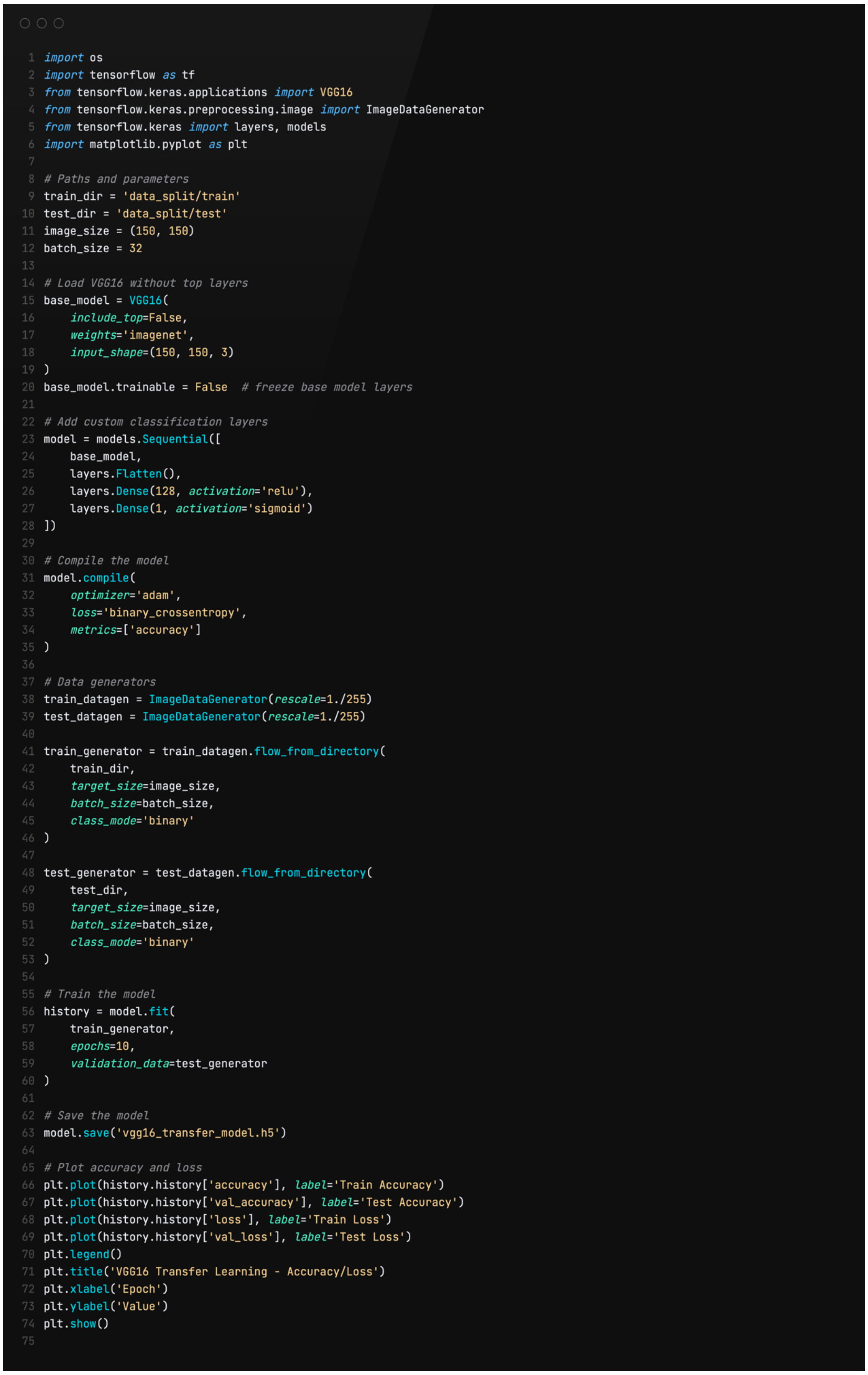
  https://github.com/USM279/driver-drowsiness-detection

- **Training Code for Custom CNN**



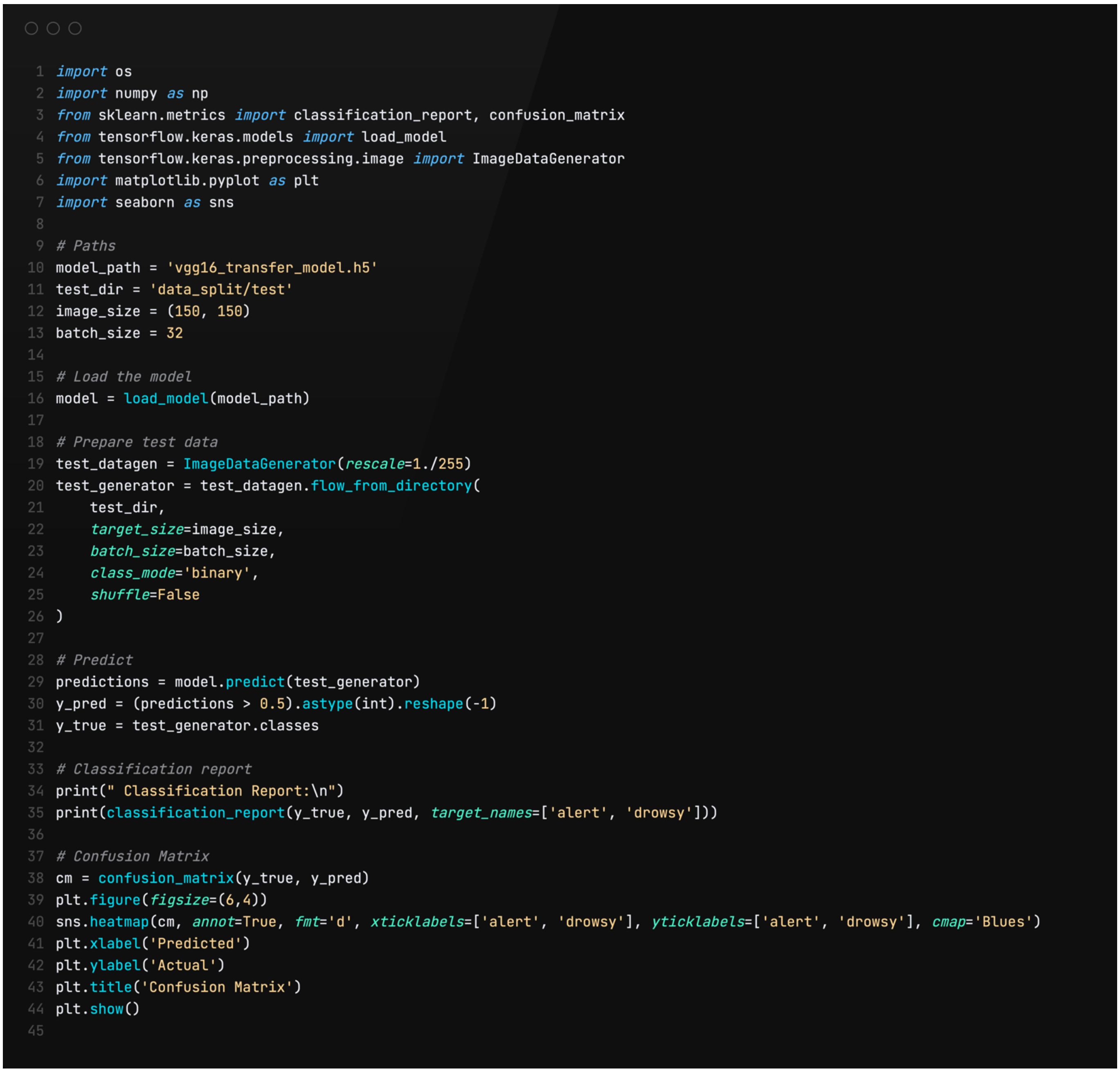
```python
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Paths
train_dir = 'data_split/train'
test_dir = 'data_split/test'
image_size = (150, 150)
batch_size = 32

# Data generators
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=image_size, batch_size=batch_size, class_mode='binary')

test_generator = test_datagen.flow_from_directory(
    test_dir, target_size=image_size, batch_size=batch_size, class_mode='binary')

# Model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(train_generator, epochs=15, validation_data=test_generator)

model.save('custom_cnn_model.h5')
```

- **Training Code for VGG16**



```python
import os
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Paths and parameters
train_dir = 'data_split/train'
test_dir = 'data_split/test'
image_size = (150, 150)
batch_size = 32

# Load VGG16 without top layers
base_model = VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(150, 150, 3)
)
base_model.trainable = False  # freeze base model layers

# Add custom classification layers
model = models.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

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

# Data generators
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'
)

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=test_generator
)

# Save the model
model.save('vgg16_transfer_model.h5')

# Plot accuracy and loss
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.legend()
plt.title('VGG16 Transfer Learning - Accuracy/Loss')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.show()
```

- **Evaluation Code (Confusion Matrix & Metrics)**



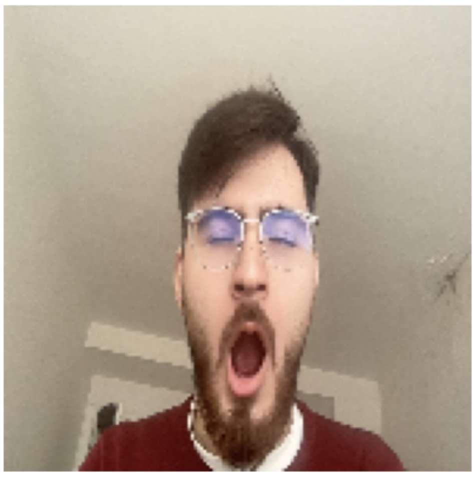
```python
import os
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import seaborn as sns

# Paths
model_path = 'vgg16_transfer_model.h5'
test_dir = 'data_split/test'
image_size = (150, 150)
batch_size = 32

# Load the model
model = load_model(model_path)

# Prepare test data
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Predict
predictions = model.predict(test_generator)
y_pred = (predictions > 0.5).astype(int).reshape(-1)
y_true = test_generator.classes

# Classification report
print(" Classification Report:\n")
print(classification_report(y_true, y_pred, target_names=['alert', 'drowsy']))

# Confusion Matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=['alert', 'drowsy'], yticklabels=['alert', 'drowsy'], cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

```

- ### Sample Images and redictions

Actual: alert | Predicted: alert

Actual: drowsy | Predicted: drowsy

Personal Image | Predicted: drowsy



- **Full list of training parameters**

Parameter	**Custom CNN**	**VGG16 Transfer**
Input Size	150 x 150 x 3	150 x 150 x 3
Batch Size	32	32
Epochs	15	10
Optimizer	Adam	Adam
Loss Function	Binary Crossentropy	Binary Crossentropy
Pretrained Weights	N/A	ImageNet
Frozen Layers	N/A	All VGG16 conv layers