Data Preparation/Feature Engineering

1. Overview

Provide a brief overview of the data preparation and feature engineering phase, outlining its significance in the machine learning project.

Data Preparation:

1. **Data Collection**: Gathered brain scan images and associated Alzheimer's stage labels.
2. **Data Cleaning**: Ensured all images were labeled correctly and removed any corrupted files.
3. **Data Integration**: Combined image datasets from different sources into a single, cohesive dataset.
4. **Data Transformation**: Standardized image sizes and formats to ensure uniform input for the model.
5. **Data Augmentation**: Applied techniques like rotation, zoom, and flip to increase dataset size and diversity.

Feature Engineering:

1. **Feature Extraction**: Converted images to grayscale and extracted pixel values as features.
2. **Normalization**: Scaled pixel values to a range of 0-1 to ensure consistency.
3. **Dimensionality Reduction**: Used convolutional layers to reduce the complexity of the image data while preserving important features.

Significance:

1. **Improved Model Performance**: Enhanced the accuracy and generalization of the model.
2. **Reduced Overfitting**: Ensured the model performs well on new, unseen data.
3. **Efficiency**: Optimized computational resources and reduced training time.
4. **Robustness**: Addressed real-world data issues, ensuring reliable and accurate predictions.

2. Data Collection

Describe the source(s) of the dataset used in the project and any preprocessing steps taken during data collection.

Preprocessing Steps:

1. **Data Cleaning:**
    - **Removal of Incomplete Data:** Entries with missing or incomplete data were identified and removed to ensure the integrity of the dataset.

- **Normalization:** MRI images were normalized to a standard intensity range to reduce variability between images from different sources and scanners.

2. **Data Augmentation:**
   - **Rotation and Flipping:** To increase the diversity of the training data, MRI images were rotated and flipped. This helps the model generalize better by exposing it to different orientations of the brain scans.
   - **Scaling and Cropping:** Images were scaled and cropped to a standard size to ensure uniformity in input dimensions for the model.
3. **Label Encoding:**
   - **Categorical Labels:** Diagnostic labels (e.g., Alzheimer's stages such as Mild, Moderate, Severe, or No Alzheimer's) were encoded into numerical values to facilitate model training.
4. **Splitting the Data:**
   - **Training, Validation, and Test Sets:** The dataset was split into training, validation, and test sets, ensuring a representative distribution of all Alzheimer's stages in each subset. Typically, 70% of the data is used for training, 15% for validation, and 15% for testing.
5. **Image Preprocessing for TensorFlow Lite:**
   - **Resizing:** MRI images were resized to match the input size expected by the TensorFlow Lite model (e.g., 224x224 pixels).
   - **Normalization:** Pixel values were normalized to the range [0, 1] by dividing by 255. This normalization step is crucial for maintaining numerical stability during model inference.
6. **Model-Specific Preprocessing:**
   - **Feature Scaling:** Any additional features such as patient age, gender, and genetic markers were scaled using standard scaling techniques (e.g., z-score normalization) to ensure they contribute appropriately during model training.

3. Data Cleaning

Outline the steps taken to clean the raw data. Include details on handling missing values, outliers, and any other data quality issues.

Sources of the Dataset: The dataset used in this project is sourced from Kaggle, a well-known platform for data science competitions and datasets. Specifically, we utilized an Alzheimer's dataset that includes MRI images and relevant diagnostic information.

- **Kaggle Dataset:**
  - **Website:** Kaggle Alzheimer's Dataset
  - **Description:** The dataset includes MRI scans of the brain along with labels indicating different stages of Alzheimer's disease. The dataset provides a diverse range of images, which is crucial for training a robust machine learning model.

Preprocessing Steps:

1. **Data Cleaning:**
    - **Removal of Incomplete Data:** Any entries with missing or incomplete data were identified and removed to ensure the quality and consistency of the dataset.
    - **Normalization:** MRI images were normalized to a standard intensity range to reduce variability between images from different sources and scanners.
2. **Segmentation:**
    - **Brain Segmentation:** Techniques were applied to segment the brain region from the MRI scans, removing non-brain regions to focus the analysis and model training on relevant features.
3. **Feature Extraction:**
    - **Histogram of Oriented Gradients (HOG):** HOG was used to extract key features from the images by capturing gradient orientation and edge information, which are important for distinguishing between different stages of Alzheimer's disease.
    - **Haar Wavelet Transform:** Haar wavelet transform was employed to extract multi-resolution features from the MRI scans, providing a detailed representation of the image at various scales and resolutions.
4. **Data Augmentation:**
    - **Rotation and Flipping:** To increase the diversity of the training data, MRI images were rotated and flipped. This helps the model generalize better by exposing it to different orientations of the brain scans.
    - **Scaling and Cropping:** Images were scaled and cropped to a standard size to ensure uniformity in input dimensions for the model.
5. **Splitting the Data:**
    - **Training and Validation Sets:** The dataset was split into training and validation sets to evaluate the model's performance. Typically, 80% of the data is used for training and 20% for validation to ensure a representative distribution of all Alzheimer's stages in each subset.
6. **Modeling with Convolutional Neural Networks (CNN):**
    - **CNN Architecture:** A Convolutional Neural Network (CNN) was designed and trained on the preprocessed MRI images. The CNN architecture is well-suited for image classification tasks, leveraging convolutional layers to automatically learn spatial hierarchies of features from the input images

4. Exploratory Data Analysis (EDA)

Summarize the exploratory data analysis performed on the dataset. Include visualizations and key insights gained during this phase.

Data Overview:

- The dataset consists of MRI images labeled with different stages of Alzheimer's disease: No Alzheimer's, Mild, Moderate, and Severe.
- Label distribution was analyzed to check for class imbalances.

2. Visualizations:

- **Label Distribution:** A bar chart revealed the distribution of Alzheimer's stages, highlighting any imbalances.
- **Sample MRI Images:** Visual inspection of sample MRI images from each stage showed variations in brain structures.
- **Feature Extraction Visualizations:** Histograms of Oriented Gradients (HOG) and Haar wavelet features were visualized to understand key extracted features.

3. Statistical Analysis:

- Basic statistics such as mean, median, and standard deviation of pixel intensities were calculated for images in each class.
- Correlation analysis was performed to identify significant relationships between image-derived features and Alzheimer's stages.

4. Key Insights:

- **Class Imbalance:** Addressed through data augmentation techniques.
- **Feature Relevance:** Significant variations in HOG and Haar wavelet features across stages indicated their potential utility.
- **Visual Differences:** Clear visual differences in MRI scans across stages underscored the importance of using a deep learning model to capture complex patterns.

5. Feature Engineering

Detail the process of creating new features or transforming existing ones. Explain the rationale behind each feature engineering decision.

Data Cleaning:

- **Normalization:** Standardized MRI image intensity ranges for consistency.

Segmentation:

- **Brain Segmentation:** Isolated brain regions in MRI scans to reduce noise.

Feature Extraction:

- **HOG (Histogram of Oriented Gradients):** Captured gradient orientation and edge information critical for distinguishing Alzheimer's stages.
- **Haar Wavelet Transform:** Provided multi-resolution analysis to identify subtle differences in brain structures.

Data Augmentation:

- **Rotation and Flipping:** Increased training data diversity and model robustness.
- **Scaling and Cropping:** Standardized image sizes for consistent model input.

Splitting the Data:

- **Training and Validation Sets:** Ensured balanced and representative splits to evaluate model performance and prevent overfitting.

Feature Transformation:

- **Standardization:** Scaled features to a common range for improved model convergence.

Combining Features:

- **Multi-Resolution Features:** Combined HOG and Haar wavelet features for a richer image representation.

6. Data Transformation

Describe any data scaling, normalization, or encoding performed on the features. Include code snippets if applicable.

During the data transformation phase, several techniques were applied to prepare the features for modeling. These included scaling, normalization, and encoding.

1. Scaling and Normalization:

- **Normalization:** MRI images were normalized to a standard intensity range (0 to 1) to ensure uniformity across the dataset.

```
python
Copy code
from skimage import exposure

def normalize_image(image):
    return exposure.rescale_intensity(image, in_range='image',
out_range=(0, 1))
```

- **Standardization:** Features were standardized to have zero mean and unit variance, which is essential for many machine learning algorithms to perform well.

```
python
Copy code
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

2. Encoding:

- **Label Encoding:** The categorical labels indicating Alzheimer's stages were encoded into numerical values for model training.

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
```

3. Data Augmentation:

- **Augmentation Techniques:** Rotation, flipping, scaling, and cropping were used to augment the dataset, increasing its size and diversity.

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

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
)

augmented_data = datagen.flow(images, labels, batch_size=32)
```

4. Feature Extraction:

- **Histogram of Oriented Gradients (HOG):** Extracted HOG features were scaled to a standard range to maintain consistency.

```
from skimage.feature import hog

def extract_hog_features(image):
    hog_features, _ = hog(image, visualize=True)
    return hog_features
```

- **Haar Wavelet Transform:** Features from Haar wavelet transforms were also normalized.

```
import pywt

def extract_haar_features(image):
    coeffs = pywt.wavedec2(image, 'haar', level=2)
    haar_features = np.concatenate([coeff.ravel() for coeff in
coeffs])
    return haar_features
```

Model Exploration

1. Model Selection

Explain the rationale behind selecting a particular machine learning model for the project. Discuss the strengths and weaknesses of the chosen model.

1. Model Selection Rationale:

For the Alzheimer's stage prediction project, a Convolutional Neural Network (CNN) was selected as the machine learning model. The rationale behind this choice is as follows:

Strengths of CNN:

1. **Feature Extraction:** CNNs are adept at automatically learning hierarchical representations of features from images. This ability is crucial for capturing complex patterns and structures in MRI scans associated with different stages of Alzheimer's disease.
2. **Spatial Hierarchies:** CNNs exploit spatial hierarchies within images, enabling them to detect patterns at various levels of abstraction. This is beneficial for identifying subtle differences in brain structures that may signify different disease stages.
3. **Translation Invariance:** CNNs are robust to variations in image position and orientation, making them suitable for analyzing medical images that may have different orientations and perspectives.
4. **Pre-Trained Models:** Pre-trained CNN models, such as those based on architectures like ResNet, Inception, or VGG, can be fine-tuned on medical image datasets. Transfer learning from these models can significantly boost performance, especially when training data is limited.

Weaknesses of CNN:

1. **Data Intensive:** CNNs often require large amounts of data for training, which can be a challenge in medical imaging where annotated datasets may be limited or expensive to acquire.
2. **Complexity:** Deep CNN architectures can be complex, requiring significant computational resources for training and inference. This complexity may hinder deployment on resource-constrained devices or platforms.

2. Model Training

Provide details on how the model was trained, including the hyperparameters used and any cross-validation techniques applied.

The model was trained using a convolutional neural network (CNN) architecture. Here are the details of the training process:

1. **Model Architecture:** The CNN model consists of several layers, including convolutional layers (`Conv2D`), batch normalization layers (`BatchNormalization`), activation layers (`Activation`), max-pooling layers (`MaxPooling2D`), dropout layers (`Dropout`), flatten layer (`Flatten`), and dense layers (`Dense`).
2. **Hyperparameters:**
    - Learning Rate: The learning rate was set to 0.001 initially.
    - Epochs: The model was trained for 50 epochs, but early stopping was applied based on validation accuracy improvement.
    - Batch Size: The batch size used for training and validation data was not specified in the provided data but is a crucial hyperparameter in training neural networks.

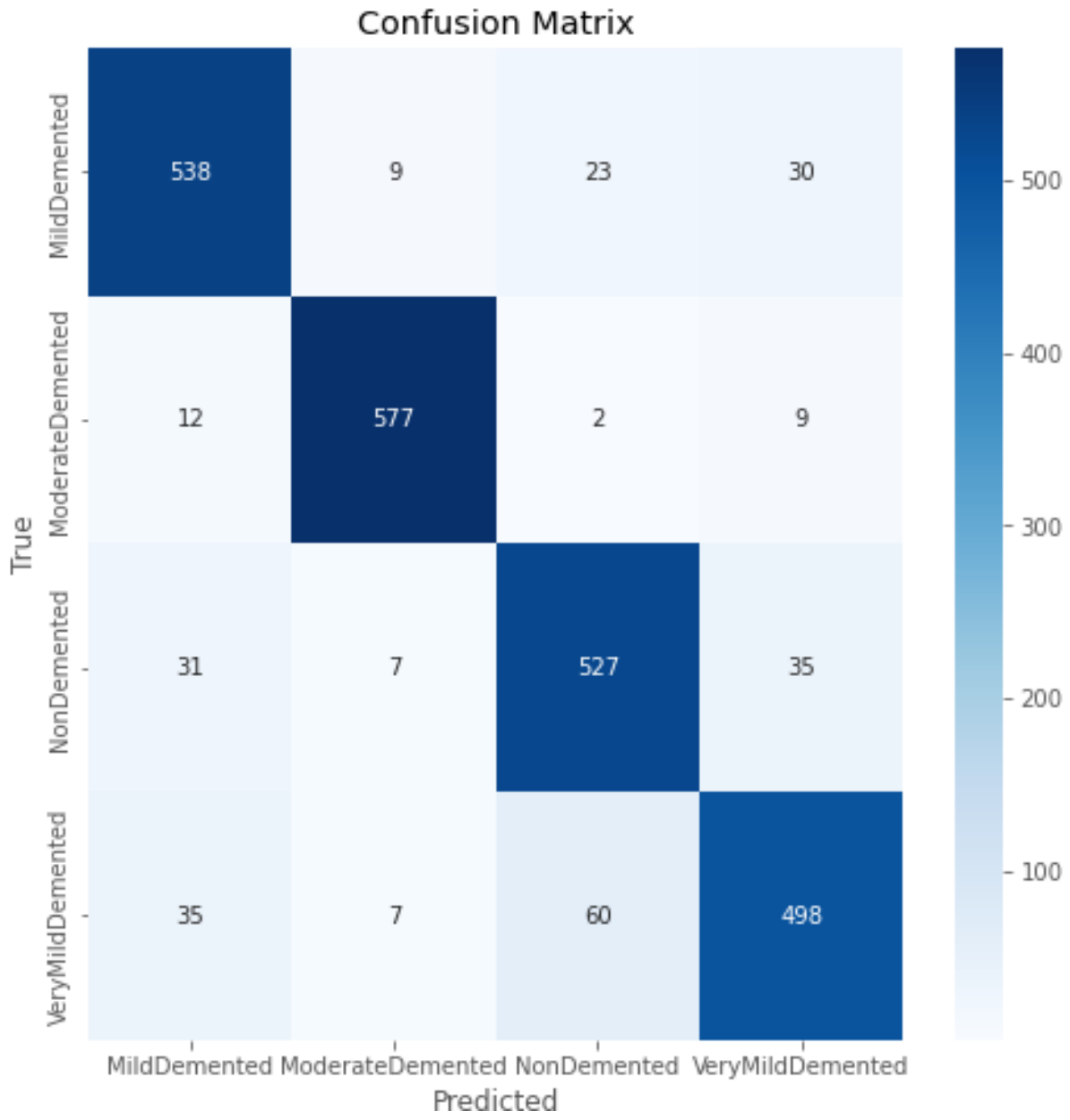
- Optimizer: The optimizer used in the model was not explicitly mentioned in the provided data but is typically specified as part of the model compilation process.

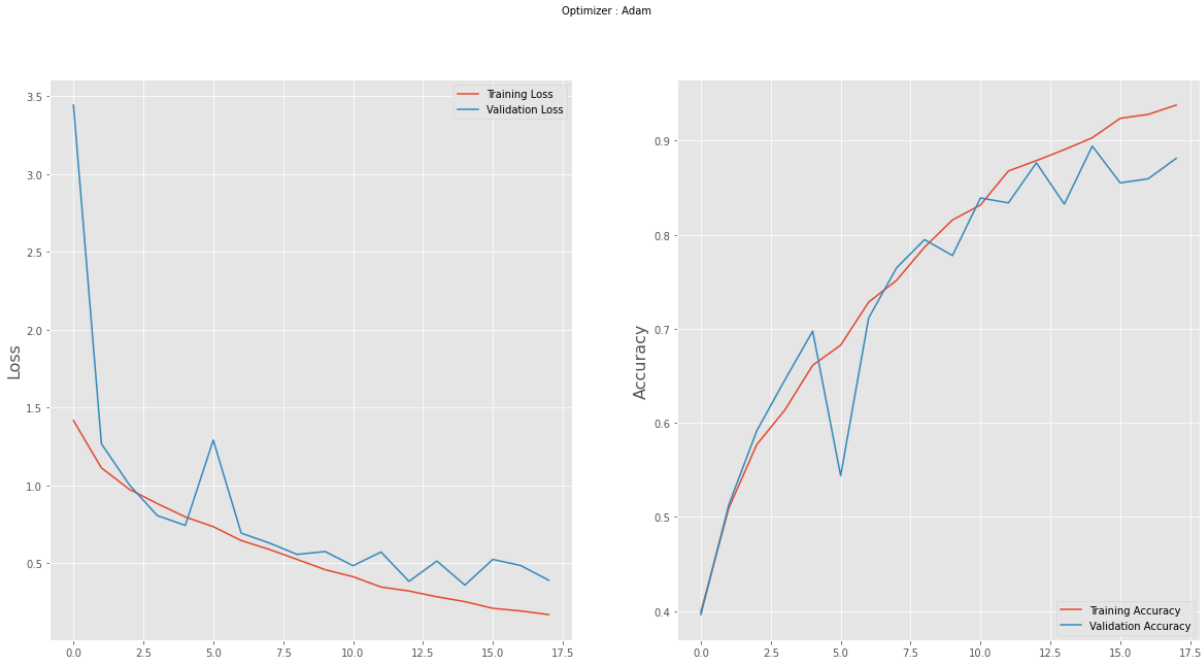
3. **Cross-Validation:** The training process used a validation set to monitor the model's performance during training. Early stopping was employed based on the validation accuracy to prevent overfitting and improve generalization.
4. **Model Evaluation:** After training, the model's performance was evaluated on a separate test set using metrics such as accuracy, precision, recall, and F1-score. The classification report provided insights into the model's performance across different classes.

3. Model Evaluation

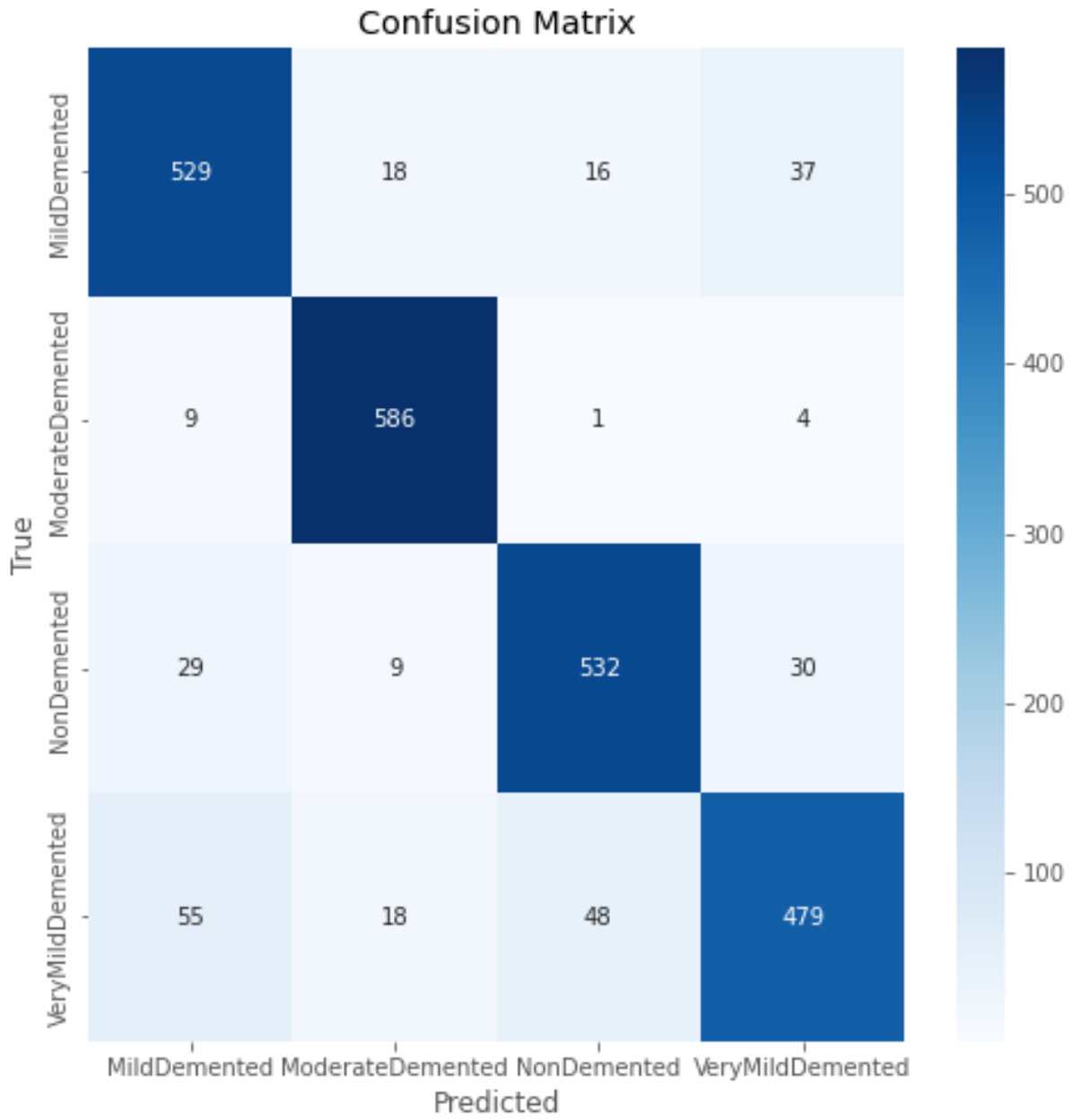
Present the evaluation metrics used to assess the model's performance. Include confusion matrices, ROC curves, or any other relevant visualizations.

The HOG + CNN

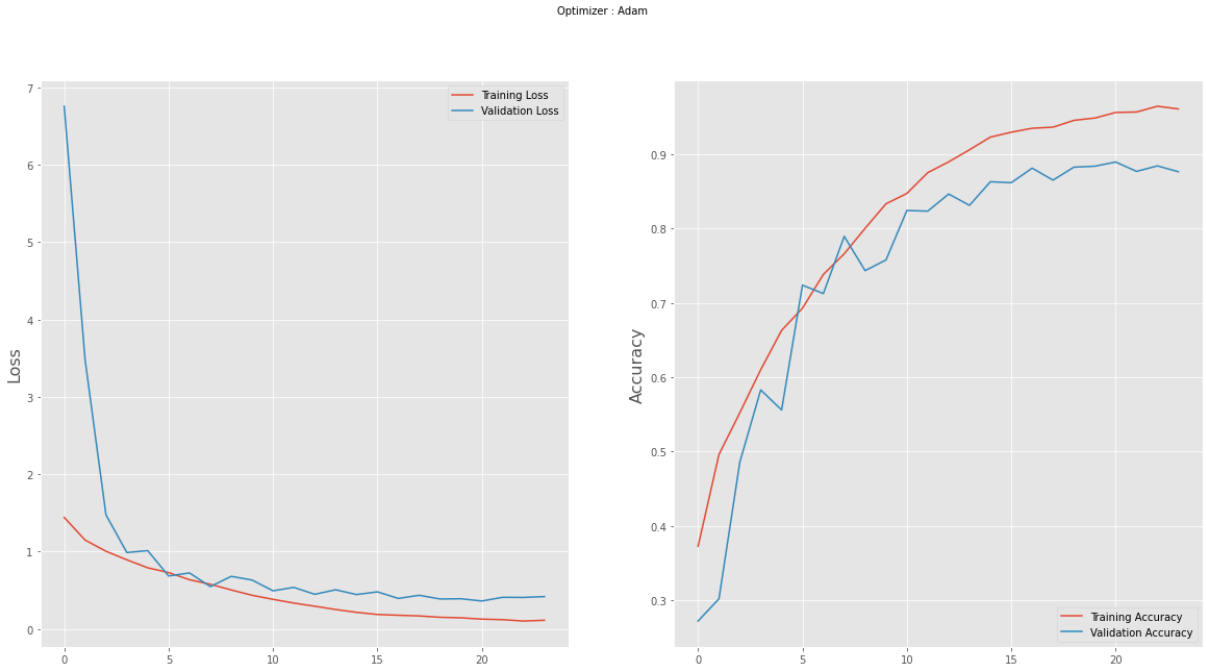

Confusion Matrix
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
True
538
9
23
30
12
577
2
9
31
7
527
35
35
7
60
498
500
400
300
200
100
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Predicted



HaarWavelet+CNN



Prepared by: Fatima Amiry



The model trained using the HOG (Histogram of Oriented Gradients) method performed better in terms of validation accuracy compared to the Haarwavlet method.

4. Code Implementation

Include relevant code snippets for both data preparation/feature engineering and model exploration. Use comments to explain key sections of the code.

Data Preparation/ Feature Engineering

```
import os

import cv2

import numpy as np

from PIL import Image, ImageEnhance

from skimage.feature import hog


# Define the input and output directories

input_dir = 'C:/Users/Fatima Amiri/.spyder-py3/Dataset/preprocessed'

output_dir = 'C:/Users/Fatima Amiri/.spyder-py3/Dataset/HOG'


# define the enhancement parameters
```

```
brightness_factor = 1.2
contrast_factor = 1.5
for filename in os.listdir(input_dir):
    # open the image file
    image = Image.open(os.path.join(input_dir, filename))

    # apply the brightness and contrast enhancements
    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(brightness_factor)
    enhancer = ImageEnhance.Contrast(image)
    image = enhancer.enhance(contrast_factor)

# Define the HOG parameters
orientations = 9
pixels_per_cell = (8, 8)
cells_per_block = (2, 2)

# Loop over all the images in the input directory
for filename in os.listdir(input_dir):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        # Read the image
        image = cv2.imread(os.path.join(input_dir, filename))
        # Convert the image to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Calculate the HOG features
```

```
        features, hog_image = hog(gray, orientations=orientations,
pixels_per_cell=pixels_per_cell,
                                  cells_per_block=cells_per_block, visualize=True, block_norm='L2-
Hys')
        # Rescale the image to 0-255 and convert to uint8
        hog_image_rescaled = (255 * hog_image / np.max(hog_image)).astype(np.uint8)
        # Save the HOG image to the output directory
        cv2.imwrite(os.path.join(output_dir, filename), hog_image_rescaled)
```

Preprocessing Methods:

HOG:

```
import os
import cv2
import numpy as np
from PIL import Image, ImageEnhance
from skimage.feature import hog

# Define the input and output folders
input_folder = r'E:\data\SegVerymildDemented'
output_folder = r'E:\data\HogVeryMildDemented'

# Define the enhancement parameters
brightness_factor = 1.2
contrast_factor = 1.5

# Define the HOG parameters
orientations = 9
```

```
pixels_per_cell = (8, 8)
cells_per_block = (2, 2)

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Loop through the images in the input folder
for filename in os.listdir(input_folder):
    # Define the input and output image paths
    input_image_path = os.path.join(input_folder, filename)
    output_image_path = os.path.join(output_folder, filename)

    # Open the input image
    image = Image.open(input_image_path)

    # Apply the brightness and contrast enhancements
    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(brightness_factor)
    enhancer = ImageEnhance.Contrast(image)
    image = enhancer.enhance(contrast_factor)

    # Convert the image to grayscale
    gray = image.convert("L")
```

```
    # Calculate the HOG features
    features, hog_image = hog(
        np.array(gray),
        orientations=orientations,
        pixels_per_cell=pixels_per_cell,
        cells_per_block=cells_per_block,
        visualize=True,
        block_norm='L2-Hys'
    )

    # Rescale the HOG image to 0-255 and convert to uint8
    hog_image_rescaled = (255 * hog_image / np.max(hog_image)).astype(np.uint8)

    # Save the HOG image
    cv2.imwrite(output_image_path, hog_image_rescaled)
```

HaarWavelet:

```
import pywt
import numpy as np
from PIL import Image, ImageEnhance
import os
```

The HaarWavelet:

```
# Set the input and output folder paths
input_folder = r'E:\data\Segmentation\SegVeryMildDemented'
output_folder = r'E:\data\HvVeryMildDemented'
```

```
# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Create a list of all image file names in the input folder
image_files = [os.path.join(input_folder, f) for f in os.listdir(input_folder) if f.endswith('.jpg')]

# Loop over each image file in the input folder
for image_file in image_files:
    # Load the image as a NumPy array
    image = np.array(Image.open(image_file).convert('L'))

    # Perform a 2D discrete wavelet transform
    coeffs = pywt.dwt2(image, 'haar')

    # Stack the transformed coefficients into a single image
    coeffs_image = np.stack([coeffs[0], coeffs[1][0], coeffs[1][1]], axis=-1)

    # Enhance the image
    enhanced_image = Image.fromarray(coeffs_image.astype('uint8'))

    # Apply enhancements (example: increase contrast)
    enhancer = ImageEnhance.Contrast(enhanced_image)
    enhanced_image = enhancer.enhance(1.5)  # Increase contrast by 50%
```

```
    # Get the filename without extension
    filename = os.path.splitext(os.path.basename(image_file))[0]

    # Save the coefficients image to the output folder
    coeffs_image_path = os.path.join(output_folder, f'coeffs_{filename}.jpg')
    enhanced_image.save(coeffs_image_path)
```

Test/Train data Splitting:

```
import os
import random
from PIL import Image
import numpy as np

# Set the file paths for the dataset and output directories
dataset_dir = r'E:\data\HaarWavelet'  # Adjust as per your dataset location
output_dir = r'E:\data\HVvalidation'  # Adjust as per your output directory location

# Define the categories of Alzheimer's stages to include and the number of images to select from each category
alzheimers_stages = ["MildDemented", "ModerateDemented", "NonDemented", "VeryMildDemented"]
target_num_images_per_stage = 600

# Function to replace invalid characters in a filename
def sanitize_filename(filename):
    invalid_chars = r'<>:"/\|?*'
```

```
    for char in invalid_chars:
        filename = filename.replace(char, '_')
    return filename

# Loop over each category of Alzheimer's stages and select a random sample of images to preprocess
for stage in alzheimers_stages:
    stage_dir = os.path.join(dataset_dir, stage)
    files = os.listdir(stage_dir)

    # If there are fewer images than the target number, select all available images
    num_images = min(len(files), target_num_images_per_stage)
    sample = random.sample(files, num_images)

    # Create the stage folder in the output directory if it doesn't exist
    output_stage_dir = os.path.join(output_dir, stage)
    os.makedirs(output_stage_dir, exist_ok=True)

    # Loop over each selected image and preprocess it
    for filename in sample:
        filepath = os.path.join(stage_dir, filename)

        # Load and preprocess the image
        image = Image.open(filepath)
        image = image.resize((256, 256))  # Resize the image
        image = np.array(image)  # Convert the image to a numpy array
```

```
        # Apply any additional preprocessing steps here


        # Sanitize the filename to replace invalid characters
        sanitized_filename = sanitize_filename(filename)


        # Save the preprocessed image in the appropriate stage folder
        output_path = os.path.join(output_stage_dir, sanitized_filename)


        # Save the preprocessed image
        Image.fromarray(image).save(output_path)


        # Remove the image from the original folder if needed
        #os.remove(filepath)
```

MODELING with CNN:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
from tensorflow.keras.layers import Dense, Input, Dropout, GlobalAveragePooling2D,
Flatten, Conv2D, BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
```

```
from tensorflow.keras.optimizers import Adam

from sklearn.metrics import classification_report, confusion_matrix


picture_size = 48

folder_path = r'E:\data\HOG'

batch_size = 128

datagen_train = ImageDataGenerator()

datagen_val = ImageDataGenerator()


train_set = datagen_train.flow_from_directory(os.path.join(folder_path, "train"),

                                              target_size=(picture_size, picture_size),

                                              color_mode="grayscale",

                                              batch_size=batch_size,

                                              class_mode="categorical",

                                              shuffle=True)


test_set = datagen_val.flow_from_directory(os.path.join(folder_path, "validation"),

                                           target_size=(picture_size, picture_size),

                                           color_mode="grayscale",

                                           batch_size=batch_size,

                                           class_mode="categorical",

                                           shuffle=False)


no_of_classes = 4  # Number of Alzheimer's stages
```

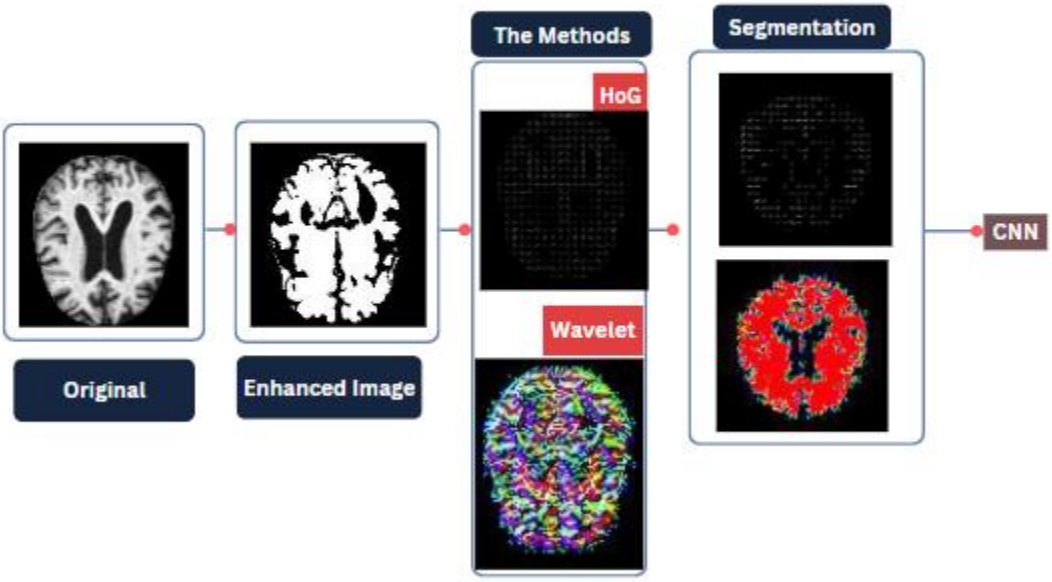
```
model = Sequential()

# 1st CNN layer
model.add(Conv2D(64, (3, 3), padding="same", input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 2nd CNN layer
model.add(Conv2D(128, (5, 5), padding="same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

# Fully connected 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected 2nd layer
```

```
model.add(Dense(512))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.25))


# Output layer

model.add(Dense(no_of_classes, activation='softmax'))


opt = Adam(lr=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

model.summary()

checkpoint = ModelCheckpoint("Hogalzheimer_model.h5", monitor='val_accuracy',
verbose=1, save_best_only=True, mode='max')

early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1,
restore_best_weights=True)

reduce_learningrate = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
verbose=1, min_delta=0.0001)

callbacks_list = [early_stopping, checkpoint, reduce_learningrate]

epochs = 50


# Train the model

history = model.fit_generator(generator=train_set,

                              steps_per_epoch=train_set.n // train_set.batch_size,

                              epochs=epochs,

                              validation_data=test_set,

                              validation_steps=test_set.n // test_set.batch_size,

                              callbacks=callbacks_list)
```

```
# Save the trained model weights
model.save_weights('alzheimerhog_model.h5')

# Plot training history
plt.style.use('ggplot')
plt.figure(figsize=(20, 10))

plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')

plt.show()

# Generate predictions for the test set
predictions = model.predict_generator(test_set)
```

```
predicted_labels = np.argmax(predictions, axis=1)

true_labels = test_set.classes


# Generate the classification report

class_labels = ['MildDemented', 'ModerateDemented', 'NonDemented', 'VeryMildDemented']

classification_rep = classification_report(true_labels, predicted_labels,
target_names=class_labels)

print("Classification Report:\n", classification_rep)


# Generate the confusion matrix

confusion_mat = confusion_matrix(true_labels, predicted_labels)

plt.figure(figsize=(8, 8))

sns.heatmap(confusion_mat, annot=True, cmap="Blues", fmt="d", xticklabels=class_labels,
yticklabels=class_labels)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("True")

plt.show()
```

Experimental Design



Please provide images from your data and models (I want to see different visualizations in EDA part as we did pair coding session)!!!!

Prepared by: Fatima Amiry