Project Report

Driver Drowsiness Detection

Table of Contents

1. Introduction 2
2. Features 3
3. Technologies Used 4
4. Project Structure 5
5. Data Processing 6
    5.1. Data Paths & Setup 6
    5.2. Preprocessing Pipeline Explained 6
    5.3. Data Pipeline Diagram 7
6. Model Architecture 8
    6.1. Embedding Network (Shared Weights) 8
    6.2. Siamese Structure (Comparison) 8
7. Training Configuration 10
8. Current Status & Potential Issues 11

Document Version: 1.1 **Date:** 2025-04-25 '''

Chapter 1. Introduction

Road safety is significantly impacted by driver fatigue, making drowsiness detection a critical area of research and development. This project implements a **Driver Drowsiness Detection** system designed to mitigate this risk. The core of the system leverages **Siamese Neural Networks**, a specialized deep learning architecture adept at similarity learning. By comparing real-time facial features against baseline or known states, the network can identify subtle signs of drowsiness.

The system analyzes multiple indicators of fatigue, including:

- **Eye State:** Detecting prolonged eye closures or rapid blinking patterns often associated with falling asleep.
- **Head Pose:** Identifying head nodding or slumping, indicative of loss of alertness.
- **Mouth Movements:** Specifically monitoring for yawning, a common physiological response to tiredness.

Through integrated analysis of these features using computer vision techniques and the Siamese network, the system aims to provide timely warnings, thereby enhancing driver safety.

Chapter 2. Features

The system incorporates several key functionalities:

- **Real-Time Drowsiness Detection:** Provides continuous monitoring of the driver's face via a camera feed, processing frames in real-time to detect emergent signs of fatigue.
- **Eye Aspect Ratio (EAR) Analysis:** While the Siamese network handles the core comparison, the README mentions the potential use of EAR. This metric calculates the ratio of eye height to width, providing a quantitative measure of eye openness. Prolonged periods of low EAR typically indicate drowsiness. *(Note: EAR implementation details are not present in the provided notebook, suggesting it might be a planned feature or implemented elsewhere).*
- **Head Pose Estimation:** The system is designed to detect abnormal head movements like nodding off or significant head tilting, which are strong indicators of reduced alertness. *(Note: Specific implementation details for head pose estimation are not found in the Siamese network notebook).*
- **Mouth and Yawning Detection:** Leverages computer vision to analyze the mouth region, specifically identifying the characteristic shape and duration of a yawn, another key fatigue indicator.
- **Siamese Neural Network Core:** The central innovation is the use of a Siamese network. Instead of classifying a single image, this network compares two images (e.g., a current frame vs. a baseline frame, or an open eye vs. a closed eye) to determine their similarity or dissimilarity. This approach is particularly effective for detecting state changes indicative of drowsiness.

Chapter 3. Technologies Used

The project relies on a combination of Python libraries standard in computer vision and deep learning domains:

- **Python:** Version 3.8 or higher is required, serving as the primary programming language.
- **Tensorflow & Keras:** The core deep learning frameworks used for building, training, and potentially deploying the Siamese Neural Network. The notebook confirms versions **Tensorflow 2.17.0** and **Keras 3.6.0**. GPU memory growth is configured to avoid out-of-memory errors during training.
- **OpenCV:** A fundamental computer vision library, likely used for tasks like accessing the camera feed, basic image manipulations, and potentially feature extraction (though not explicitly imported in the initial notebook cells).
- **Dlib:** Another powerful library often used for facial landmark detection, crucial for isolating regions like eyes and mouth for analysis (inferred from README).
- **Imutils:** Provides convenience functions for common image processing tasks like resizing, rotation, and skeletonization (inferred from README).
- **Numpy:** Essential for numerical operations, particularly array manipulations involved in image processing and deep learning.
- **Matplotlib:** Used for plotting and visualizing data, such as displaying sample images from the dataset as seen in the notebook.
- **OS module:** Utilized for interacting with the operating system, primarily for path manipulations and directory creation as shown in the notebook's data setup cells.

Chapter 4. Project Structure

The project repository is organized as follows, containing configuration files, source code, documentation placeholders, and pre-trained models:

```
.
├── .editorconfig            # Editor configuration settings
├── .envrc                   # Environment variables configuration (direnv)
├── .gitattributes           # Git attributes file
├── .gitignore               # Specifies intentionally untracked files for Git
├── deep learning report.docx  # Original report document (likely Word format)
├── flake.lock               # Nix flake lock file (dependency management)
├── flake.nix                # Nix flake definition (dependency management)
├── LICENSE                  # Project license file
├── pyproject.toml           # Python project metadata and build configuration (PEP
518)
├── README.md                # High-level project description and usage guide
├── siamese_network.ipynb    # Jupyter notebook containing the core Siamese network
implementation
├── uv.lock                  # Lock file for 'uv' Python package manager
├── docs/                    # Directory intended for project documentation
(currently empty)
└── haarcascade/             # Contains pre-trained Haar cascade classifiers
    ├── haarcascade_eye.xml  # Classifier for detecting eyes
    └── haarcascade_frontalface_default.xml # Classifier for detecting frontal
faces
```

Note: The `dataset` directory mentioned in the notebook is absent, indicating data needs external sourcing or generation.

Chapter 5. Data Processing

Training a Siamese network requires carefully structured data, typically involving pairs of images labeled as similar or dissimilar. This project prepares datasets for eye state and yawning detection using anchor, positive (similar), and negative (dissimilar) image samples.

5.1. Data Paths & Setup

The notebook outlines the creation of necessary directory structures for storing the datasets under a main `dataset` folder.

- **Eyes Dataset:**
  - `dataset/eyes/anchor/` (Baseline eye images)
  - `dataset/eyes/positive/` (Images similar to anchor, e.g., open eyes if anchor is open)
  - `dataset/eyes/negative/` (Images dissimilar to anchor, e.g., closed eyes if anchor is open)
- **Yawn Dataset:**
  - `dataset/yawn/anchor/` (Baseline mouth images, likely non-yawning)
  - `dataset/yawn/positive/` (Images similar to anchor, e.g., non-yawning mouths)
  - `dataset/yawn/negative/` (Images dissimilar to anchor, e.g., yawning mouths)

Important Note: As mentioned, the `dataset` directory itself is not included in the provided file structure. The data would need to be populated in these paths before the notebook code for data loading can execute successfully.

5.2. Preprocessing Pipeline Explained

To handle data loading and preparation efficiently, a TensorFlow (`tf.data`) pipeline is defined in the notebook. This pipeline automates the steps required to transform raw image files into batches suitable for training the neural network:

1. **File Listing:** `tf.data.Dataset.list_files` identifies all `.jpg` files within the specified anchor, positive, and negative directories. A subset (`.take(300)`) is selected, likely for managing dataset size during development or testing.
2. **Image Reading:** `tf.io.read_file` reads the raw byte content of each image file.
3. **Decoding:** `tf.io.decode_jpeg` decodes the raw bytes into image tensors.
4. **Resizing:** `tf.image.resize` standardizes all images to a fixed input size of 105x105 pixels, as required by the embedding network.
5. **Normalization:** Pixel values (typically 0-255) are scaled to the range [0, 1] by dividing by 255.0. This is a standard practice that helps stabilize training.
6. **Pairing and Labeling:** `tf.data.Dataset.zip` combines anchor images with positive images (labeled `1.0`) and anchor images with negative images (labeled `0.0`) to create the input pairs for the Siamese network.

7. **Concatenation:** The positive and negative pair datasets are concatenated into a single dataset.
8. **Caching:** `.cache()` stores the preprocessed data in memory (or on disk) after the first epoch, speeding up subsequent epochs by avoiding repeated preprocessing.
9. **Shuffling:** `.shuffle(buffer_size=1024)` randomizes the order of data samples to improve training generalization.
10. **Train/Test Split:** The dataset is split into training (70%) and testing (30%) partitions using `.take()` and `.skip()`.
11. **Batching:** `.batch(16)` groups samples into batches of size 16 for stochastic gradient descent.
12. **Prefetching:** `.prefetch(8)` allows the pipeline to prepare the next batches of data while the current batch is being processed by the GPU, optimizing throughput.

5.3. Data Pipeline Diagram

This diagram illustrates the flow of data from image files to batched training and testing datasets.

Chapter 6. Model Architecture

The system utilizes a Siamese Neural Network architecture. This design is well-suited for tasks involving similarity comparison, such as determining if two facial images represent the same state (e.g., yawning vs. non-yawning).

6.1. Embedding Network (Shared Weights)

At the heart of the Siamese network is a Convolutional Neural Network (CNN) that acts as an **embedding generator**. This network processes an input image (105x105 pixels, 3 color channels) and transforms it into a dense, lower-dimensional feature vector (4096 dimensions in this case), known as an embedding. The key characteristic is that the **same embedding network with shared weights** is used to process both images in an input pair. This ensures that similar input images are mapped to nearby points in the embedding space.

The architecture of the embedding network defined in the notebook is as follows:

- **Input Layer:** Accepts images of shape (105, 105, 3).
- **Convolutional Block 1:**
  - Conv2D layer with 64 filters, a 10x10 kernel size, ReLU activation, and valid padding. Extracts initial low-level features.
  - MaxPooling2D layer with a 2x2 pool size to reduce dimensionality and provide spatial invariance.
- **Convolutional Block 2:**
  - Conv2D layer with 128 filters, a 7x7 kernel, and ReLU activation. Captures more complex features.
  - MaxPooling2D layer (2x2).
- **Convolutional Block 3:**
  - Conv2D layer with 128 filters, a 4x4 kernel, and ReLU activation. Further feature extraction.
  - MaxPooling2D layer (2x2).
- **Convolutional Block 4:**
  - Conv2D layer with 256 filters, a 4x4 kernel, and ReLU activation. Extracts higher-level features.
- **Flatten Layer:** Converts the 2D feature maps from the convolutional layers into a 1D vector.
- **Dense Output Layer:** A fully connected layer with 4096 units and a Sigmoid activation function, producing the final embedding vector.

6.2. Siamese Structure (Comparison)

The complete Siamese network integrates two instances of the shared embedding network:

1. **Inputs:** Takes two images simultaneously: an `input_image` and a `validation_image`.

2. **Embedding Generation:** Both images are independently passed through the **same** embedding network (defined above) to generate their respective 4096-dimensional embedding vectors.
3. **Distance Calculation:** A custom Keras layer, `L1Dist`, calculates the absolute element-wise difference (L1 or Manhattan distance) between the two embedding vectors. This distance metric quantifies the dissimilarity between the features extracted from the two input images.
4. **Classifier:** A final Dense layer with a single output unit and a Sigmoid activation function takes the distance vector as input. It outputs a probability score between 0 and 1, indicating the likelihood that the two input images belong to the same class (similar state). A score close to 1 suggests similarity (positive pair), while a score close to 0 suggests dissimilarity (negative pair).

Chapter 7. Training Configuration

The notebook sets up the training process as follows:

- **Optimizer:** The Adam optimizer is chosen, a popular adaptive learning rate optimization algorithm, configured with a learning rate of `1e-4` (0.0001).
- **Loss Function:** `tf.losses.BinaryCrossentropy` is used, suitable for the binary classification task (similar/dissimilar pair).
- **Training Loop:** A custom training loop is defined using `@tf.function` for potential performance optimization. The `train_step` function calculates the loss for a batch, computes gradients using `tf.GradientTape`, and applies updates to the model's trainable variables via the optimizer.
- **Epochs:** The `train` function iterates over the specified number of epochs, printing progress using `tf.keras.utils.Progbar`.
- **Checkpoints:** `tf.train.Checkpoint` is configured to save the model's state (optimizer and Siamese model weights) every 4 epochs to the `./training_checkpoints` directory, allowing training to be resumed later.

Chapter 8. Current Status & Potential Issues

While the notebook provides a clear structure for the Siamese network and its training, the execution results embedded within the `.ipynb` file highlight critical issues that prevent successful execution in its current state:

1. **`TypeError` in `L1Dist` Layer:** During the instantiation of the Siamese model (`make_siamese()` function), a `TypeError` is raised within the `L1Dist.call()` method. The error message "unsupported operand type(s) for -: 'list' and 'list'" indicates that the inputs received by the L1 distance layer (`input_embedding` and `validation_embedding`) are lists instead of the expected TensorFlow tensors. This likely stems from an issue in how the outputs of the shared `embedding` network are being passed or handled within the Siamese structure definition. This needs correction for the model to be built.
2. **`NameError` for `siam` Variable:** Consequently, because the `make_siamese()` function failed to execute successfully due to the `TypeError`, the variable `siam` (intended to hold the compiled Siamese model) was never defined. This leads to a `NameError` in the subsequent cell where `tf.train.Checkpoint` attempts to reference `siam`.
3. **Missing Dataset:** The data loading cells rely on a `dataset` directory containing `eyes` and `yawn` subdirectories with `anchor`, `positive`, and `negative` images. This directory is not present in the project's file listing provided. The data needs to be acquired and placed in the expected structure for the data pipeline to function.

Conclusion on Status: The notebook provides the architectural blueprint but is currently non-functional due to the errors in the model definition and the absence of the required dataset. Debugging the `L1Dist` layer interaction and ensuring the dataset is correctly located are necessary next steps to enable model training.