|  |
| --- |
| **DAYANANDA SAGAR UNIVERSITY**  A blue and black text  Description automatically generated  COGNITIVE LOAD DETECTION  *Based On Computer Vision*  Project Implementation Details & Discussion |

*By*

MEENAKSHI

[ENG24CSE0013]

M.TECH 3rd Semester,

Department of Computer Science & Engineering

DAYANANDA SAGAR UNIVERSITY

14-9-2025

CONTENTS

Page No.

[**COGNITIVE LOAD DETECTION** 2](#_Toc208755887)

[1. Project Implementation Summary 2](#_Toc208755889)

[2. Data Preparation 2](#_Toc208755890)

[3. Model Training & Stacking Ensemble 3](#_Toc208755891)

[4. Future Steps 4](#_Toc208755892)

[5. Results Obtained 5](#_Toc208755893)

[6. Installation & Execution Commands: 6](#_Toc208755894)

[7. Handy Commands 7](#_Toc208755895)

[8. Why .h5 File was Created? 8](#_Toc208755896)

[9. Dataset Differences Screenshot 9](#_Toc208755897)

# 

# COGNITIVE LOAD DETECTION

# Based On Computer Vision

## Project Implementation Summary

The goal of this project is to detect cognitive load in students during online examinations by recognizing their facial micro expressions. The implementation follows a structured approach based on the reference paper, which involved two key phases: data preparation & model training.

## Data Preparation

**Datasets Used**: The project uses two publicly available datasets:

* ***FER2013***: Loaded from a folder-based structure with separate train & test directories.
* ***JAFFE***: Loaded from a single folder containing *.tiff* images.

**Data Pipeline**: A dedicated data\_preprocessing.py script was created to handle all data-related tasks, ensuring a clean & modular codebase. It performs the following steps:

* Loads all images & their corresponding labels from the specified directories.
* Converts images to grayscale & resizes them to a uniform 48x48 pixel size.
* Normalizes pixel values from a 0-255 range to a 0-1 range.
* Uses a single LabelEncoder to consistently map all emotion labels (ex: 'angry', 'happy') to integers across all datasets.
* One-hot encodes the integer labels to a total of 9 classes.
* Combines the FER2013 training set & the JAFFE dataset into a single, unified training set, while the FER2013 test set serves as the validation set.

## Model Training & Stacking Ensemble

**Model Architectures:** The implementation follows the paper's recommendation by training five distinct Convolutional Neural Network (CNN) models independently:

* Simple-CNN
* Simpler-CNN
* Tiny-XCEPTION
* Mini-XCEPTION
* Big-XCEPTION

**Stacking Method**: A stacking ensemble method was implemented to combine the outputs of the five individual CNN models into a single meta-model. This approach is expected to significantly improve recognition accuracy by leveraging the unique strengths of each base model.

## Future Steps

The current implementation provides a solid foundation.

*The next steps to advance the project are as follows:*

**Implement Data Augmentation**: The reference paper mentions using data augmentation (such as rotating, scaling, & color-changing images) to improve model robustness. This functionality has to be added to the *data\_preprocessing.py* script.

**Hyperparameter Tuning**: Experiment with different learning rates, batch sizes, & numbers of epochs to further optimize the performance of both the base models & the stacking ensemble.

**Create a Custom Dataset**: The paper emphasizes the importance of a custom-made dataset of blended classroom videos. We need to plan & execute the creation of our own dataset with a setup. This is a critical step to ensure the final model is specifically tailored to our project's context.

**Real-time Inference System**: Develop a system to use the final saved model to predict emotions from a live video feed, which is the ultimate goal of the project.

## 

## Results Obtained

The code executed successfully completed the data preprocessing & model training.

The final results obtained are as follows:

* A combined training set of 28,922 images & a validation set of 7,178 images, all of a consistent (48, 48, 1) shape.
* A unified set of 9 emotion labels identified across the datasets.
* Five trained base models & one final stacking ensemble model saved as .keras files.
* A JSON file containing the accuracy & training history for all models.
* A confusion matrix image that visually represents the final model's performance.

## Installation & Execution Commands:

Here is a list of commands to easily set up & run the project on any new system with Python & Git installed.

1. **Install all required packages.**   
   pip install pandas numpy opencv-python scikit-learn tensorflow matplotlib seaborn
2. **Update the file paths** in the *data\_preprocessing.py* file.
3. **Run the model training script** to process the data, train, & save the models: *python model\_training.py*
4. **Install h5py** if required: *pip install h5py*
5. To see the **tabular summary**, run the *view\_results.py* script **without any arguments**: *python training\_results.py*
6. To see the **detailed, epoch-by-epoch results**, use the --verbose flag: *python training\_results.py –verbose*

## Handy Commands

1. Clone the project repository (if applicable) or navigate to the project directory: *cd /path/to/our/project*
2. Create a new virtual environment to isolate dependencies: *python -m venv venv\_fyp*
   * + Activate the virtual environment.  
       On Windows (Git Bash/WSL):   
        *source venv\_fyp/Scripts/activate*
     + On macOS/Linux:  
        source venv\_fyp/bin/activate

## Why *.h5 File* was Created?

We created the *.h5 files* to save the trained models, because the preferred .keras format was not working properly on the system. The *.h5 file* format is a well-established standard for storing large, hierarchical datasets, & it is particularly well-suited for saving machine learning models in a single, portable file.

What Information the .h5 File Contains ?

An .h5 file generated by Keras contains everything needed to use in our trained model without having to retrain it from scratch. *This includes:*

* **Model Architecture:** The structure of our model, including the number & type of layers (e.g., Conv2D, MaxPooling2D, SeparableConv2D, Dense, etc.)
* **Model Weights:** This is the most important part. The weights are the numerical parameters that the model learned from the training data, essentially the "knowledge" that allows it to make predictions.
* **Optimizer State:** The state of the optimizer (e.g., Adam), which is necessary if we want to resume training the model later on.

Essentially, the .h5 file is a self-contained package that captures the entire state of our trained model at the end of the training process.

Model saving is essential because a trained model only exists in a computer's temporary memory (RAM) while a script is running. Once the script finishes, the model is erased from memory & cannot be recovered. Saving the model to a file, such as an .h5 file, makes it permanent. This is a critical step for a project, as it allows the trained model to be used for making predictions or integrating it into a live system without having to retrain it from scratch every time.

## Dataset Differences Screenshot

The following image highlights the differences between FER2013 & JAFFE datasets:

