UNIVERSITY OF LONDON

INTERNATIONAL PROGRAMMES

BSc Computer Science (Machine Learning and Artificial Intelligence)

CM3070 Final Project

Facial Emotion Recognition with Attention-Enhanced ResNet18 for Emotion-Based Playlist Mapping

Author: Goh Yin Xuan
Student Number: 220218711

Date Of Submission: 11/03/2025
Supervisor: Liu Jun Hua

Table Of Contents

CHAPTER 1: INTRODUCTION....3
1.1 Background....3
1.2 Introduction....3
1.3 Aims and Objectives....5
1.4 Project Scope....5
CHAPTER 2: LITERATURE REVIEW....7
2.1 Dataset Analysis....7
2.2 Related Work....8
2.3 Emotion Selection....9
2.4 Methods and Techniques....11
2.4.1 Deep Learning Approaches....11
2.4.2 Data Preprocessing, Augmentation, and Training Strategies....14
2.4.3 Evaluation Metrics....15
2.5 Tools and Technologies....16
2.5.1 FER Tools and Libraries....16
2.5.2 Music Streaming Service APIs....16
CHAPTER 3: DESIGN....18
3.1 Design Foundations....18
3.1.1 Dataset Considerations and Challenges....18
3.1.2 Model Selection: ResNet18 with CBAM Enhancement....18
3.1.3 Hypotheses....19
3.2 Model Design....20
3.2.1 Baseline Model: Convolutional Neural Network Model....20
3.2.2 Proposed Model: ResNet18-CBAM....21
2.2.1 ResNet18 Architecture....21
2.2.2 CBAM Architecture....21
2.3.3 ResNet18-CBAM....22
3.2.3 Evaluation Plan....23
3.3 Tools and Technologies Justification....24
3.4 Project Timeline and Management....24
CHAPTER 4: IMPLEMENTATION....27
4.1 Dataset Preparation....27
4.2 Model Implementation....30
4.2.1 Baseline CNN Model....30
4.2.2 ResNet18-CBAM model....34
4.3 Music Streaming Service API Integration....40
CHAPTER 5: EVALUATION....45
5.1 Model Performance Evaluation....45

5.1.1 Baseline CNN Model..... 45
5.1.2 ResNet18-CBAM Model.....46
5.2 Emotion-Aligned Music Playlist Mapping Evaluation..... 49
5.2.1 Webcam Real-Time Capture and Emotion Prediction..... 49
5.2.2 Spotify API Integration for Emotion-Based Playlist.....51
5.3 Hypotheses Validation.....52
5.4 Limitations and Future Work.....53
CHAPTER 6: CONCLUSION.....54
References..... 55
Appendix A: Code Repository (Google Colab Notebook).....60

CHAPTER 1: INTRODUCTION

1.1 Background

Emotions play a central role in daily life, shaping our behaviors, decisions, and interactions. Among various external stimuli, facial expressions are a key indicator of emotional states. Accurately recognizing emotions from facial expression has application in human-computer interaction, healthcare, and entertainment [20].

One area where emotion recognition could be valuable is in music-related applications. Music has been shown to affect our emotions and regulate our mood almost instantly [1]. For instance, a cheerful song can uplift one's mood, while a sad song can evoke nostalgia or sadness, illustrating the deep emotional connection between music and human experiences. Research in Nature Neuroscience reports that music triggers dopamine release, a neurotransmitter associated with emotional processing, explaining why music can evoke emotions such as happiness, sadness, and excitement [2].

Given music's profound impact on emotions, technological advancements have transformed the way people engage with music, shifting from traditional physical media to AI-driven streaming services [3]. This transition has led to a new era of personalized music recommendations, where algorithms analyze listening patterns to enhance user engagement.

With advancement in deep learning and computer vision, facial emotion recognition (FER) models have become increasingly effective in identifying emotions from facial expressions. By leveraging large datasets and deep learning techniques, these models can classify facial expressions into specific emotional categories, enabling potential integration into various applications, including adaptive user experiences in entertainment and digital services.

1.2 Introduction

Music has become an essential part of daily life, accompanying activities such as commuting, exercising, studying, and relaxation. According to the International Federation of the Phonographic Industry (IFPI), people spend an average of 20.7 hours per week listening to music, demonstrating its deep integration into everyday life [4].

With the shift from physical media, such as CD, cassettes, radios, to AI-driven music streaming platforms, services like Spotify, Apple Music, YouTube Music, and TIDAL provide users with instant access to millions of songs [3]. These platforms employ machine learning (ML) algorithms to provide personalized music choices based on historical listening habits, frequently played tracks, and genre preferences [6]. However, these recommendations

are based on past interactions rather than real-time emotional states, limiting its ability to deliver emotionally resonant music experiences [5,7].

Contextual awareness is increasingly recognized as an important factor in enhancing music personalization [8]. To enhance personalization, some platforms provide contextual feature playlists, such as mood-, activity-, location, and time-of-day-based playlists, allowing users to manually select songs that align with their emotions. Table 1.1 summarizes four major services, highlighting their contextual playlist features and the absence of real-time emotion recognition music recommendation.

Table 1. 1 Comparison of Contextual Playlist Features Across Music Streaming Platforms

Platform	Available Contextual Features	Real-Time Emotion Recognition Recommendation
Spotify	Diverse playlists for mood (Chill Hits, Mood Booster), genre (R&B, Hip-Hop), activity (Workout), and time (Discover Weekly, Daily Mixes, Daylist).	No
Apple Music	Curated playlists for moods, activity, seasonal, and time-of-day (Feel Good, Workout Mix, Morning Coffee).	No
YouTube Music	Limited playlists for mood (Feel Good, Sad, Relax), activity (Focus, Workout, Commute), and situational (Party, Romance).	No
TIDAL	Primarily genres-based playlist, fewer moods/activity-based options.	No

Despite these contextual features, no major streaming platforms incorporate real-time FER-based music selection. A user experiencing stress may not find their usual energetic playlists helpful but might benefit from calming or reflective music. Current streaming services lack the ability to detect a listener's emotions dynamically, making their recommendations reactive rather than proactive [7]. In this context, FER could provide an additional layer of personalization to detect users emotions in real-time. FER uses deep learning techniques to classify facial expressions into distinct emotional states and recommend songs accordingly, enabling application in the music streaming industry.

This project focuses on developing a FER model using the FER-2013 dataset to classify facial expressions into emotional states. The model will be experimentally tested with a music streaming service to explore emotion-based playlist mapping for a more responsive listening experience.

While this project does not generate individualized recommendations based on listening history, it suggests playlists aligned with detected emotions. This approach bridges the gap between traditional behavioral-based and real-time contextual recommendations, laying a groundwork for integrating facial emotion recognition into recommendation-driven applications.

1.3 Aims and Objectives

The aim of this project is to develop a FER model that identifies and classifies emotional states based on facial expression. The key objectives includes:

1. **Deep Learning Model Selection and Benchmarking:** Research and compare different deep learning architectures to identify the most suitable model for emotion recognition.
2. **Dataset Preparation and Preprocessing:** Use a publicly available facial emotion dataset, applying preprocessing techniques to optimize data quality and model performance.
3. **Model Training, Validation, Performance Assessment:** Train and evaluate the model using key performance metrics
4. **Experimental Integration with a Music Streaming API:** Test the feasibility of integrating the model with an existing music streaming service API as a proof of concept, showing its potential for emotion-aligned music recommendation.

1.4 Project Scope

This project focuses on developing a facial emotion recognition model as a foundation step toward emotion-aligned music recommendations. The scope is defined across three key stages:

1. **Facial Emotion Recognition Model Development:** Train and evaluate a deep learning model for facial emotion classification using a publicly available dataset.
2. **Model Performance Evaluation:** Validate the trained model using classification metrics to assess its classification effectiveness.
3. **Music Playlist Mapping (Experimental Phase):** Conduct a proof-of-concept integration with a music streaming service API to map detected emotions to predefined playlists, exploring how facial emotions can be linked to music choices.

The project follows the "Machine Learning and Neural Network Project Idea 1 - Deep Learning on a Public Dataset" template. The structure aligns with this template by utilizing a publicly available dataset (FER-2013), deep learning techniques, and standard classification metrics. The experimental API integration phase serves to test real-world feasibility rather than to develop an official deployment.

CHAPTER 2: LITERATURE REVIEW

2.1 Dataset Analysis

Selecting an appropriate dataset is critical for FER as it directly impacts model accuracy and generalizability. Table 2.1 compares three widely used datasets with their strengths and limitations.

Table 2. 1 Comparison of FER datasets

Dataset	Description	Strengths	Challenges
AffectNet [18]	Around 0.4 million labeled images across 8 emotions, including arousal and valence.	Diverse, large-scale	Computationally demanding due to its size and complexity.
FER-2013 [9]	Around 32,000 grayscale images from Kaggle challenge, categorized into 7 emotions.	Popular for academic projects, large datasets.	Low resolution, class imbalance, geographic bias.
Extended Cohn-Kanade Dataset (CK+) [19]	593 facial video sequences labeled into 7 emotions.	High-quality images, includes temporal data.	Limited dataset size, mostly posed expressions rather than real-world emotional expression.

AffectNet provides greater diversity and scale but requires high computational demands make it challenging for projects with limited hardware and time. CK+ is ideal for temporal emotion studies, which primarily contain posed expressions but lack real-world variability. FER-2013 is a practical choice due to its balance of accessibility and diversity, making it suitable for developing deep learning-based FER models despite its challenges like low resolution and class imbalance.

FER-2013 contains 32,298 grayscale facial images of 48x48 pixels, categorized into seven emotion: Angry (0), Disgust (1), Fear (2), Happy (3), Sad (4), Surprise (5), and Neutral (6), with 28,709 images for training and 3,589 for testing. Despite its advantages, issues like imbalance emotion distribution may affect model performance, requiring preprocessing techniques discussed in the later section.

2.2 Related Work

Facial emotion recognition (FER) has gained significant attention due to its applications in human-computer interaction and personalized systems. FER typically involves detecting facial expressions, extracting facial features, and mapping them to emotional states [20]. This section reviews six studies in FER, highlighting different methodologies and their respective insights and challenges which are summarized in Table 2.2.

Table 2. 2 Comparative analysis of reviewed studies

Study	**Dataset**	**Model Used**	**Insights/ Challenges**
A Lightweight Convolutional Neural Network for Real-time Facial Expression Detection [12].	FER-2013, KDEF	Lightweight CNN using MTCNN with depth-wise separable convolution and global average pooling.	Efficient for real-time applications but lacks robustness in real-world scenarios with lighting, occlusion.
Hybrid Facial Expression Recognition (FER2013) Model for Real-Time Emotion Classification and Prediction [13].	FER-2013	Hybrid DCNN + Haar Cascade with data augmentation.	Struggles with recognizing certain emotions due to the class imbalance and underrepresentation in the dataset.
Facial Emotion Recognition Based On Sobel-Resnet [14].	CK+, FER-2013	Pretrained ResNet-18 combined with the Sobel operator for preprocessing.	Uses Sobel operator for feature enhancement, but suffers from low recognition speed and struggles with distinguishing visually similar emotions.
Three-dimensional DenseNet self-attention neural network(DenseAttNet) for automatic detection of student's engagement [15].	DAiSEE	DenseNet-121 combined with a 3D self-attention module for temporal and spatial feature extraction.	Incorporates 3D self-attention for temporal and spatial features, but shows suboptimal performance on specific emotion classes.
Occlusion Aware Facial Expression Recognition Using CNN With Attention Mechanism [16].	FED-RO	Attention-based CNN (pACNN, gACNN) built on ResNet-18 and VGG16.	Applies attention mechanisms and achieves improved accuracy on diverse dataset but remains computationally intensive.
A Brief Review of Facial Emotion Recognition Based on Visual Information [17].	MMI	CNN, Hybrid CNN-LSTM approach for FER.	Computationally intensive on the model and it is more suitable for sequence temporal dynamic datasets.

Notes:

- FER-2013: Facial Expression Recognition 2013 dataset
- KDEF: Karolinska Directed Emotional Faces dataset
- CK+: Extended Cohn-Kanade dataset
- FED-RO: Facial Expression Dataset with Occlusion
- CNN: Convolutional Neural Network
- MTCNN: Multi-task Cascaded Convolutional Neural Network
- DenseNet: Dense Convolutional Network
- ResNet: Residual Network
- DCNN: Deep Convolutional Neural Network
- pACNN: Patch-based Attention Convolutional Neural Network
- gACNN: Global-based Attention Convolutional Neural Network
- LSTM: Long Short-Term Memory

The insights and challenges derived from these studies display trends and gaps in FER research. The reviewed studies indicate a strong reliance on CNN-based architectures for feature extraction and emotion classification. Advanced CNN models, such as ResNet and DenseNet have demonstrated effectiveness in handling FER tasks by improving feature learning and classification accuracy. Attention mechanisms further optimize CNNs, improving robustness against occluded inputs while introducing computational trade-offs. Hybrid approaches (CNN-LSTM) capture temporal dynamics but are more applicable to sequence-based datasets rather than static datasets like FER-2013.

Although some reviewed studies examine different datasets, they provide valuable insights applicable to FER-2013-specific challenges, including low resolution and real-world variability. Accordingly, this project focuses on leveraging CNN-based architectures for developing a robust FER model.

2.3 Emotion Selection

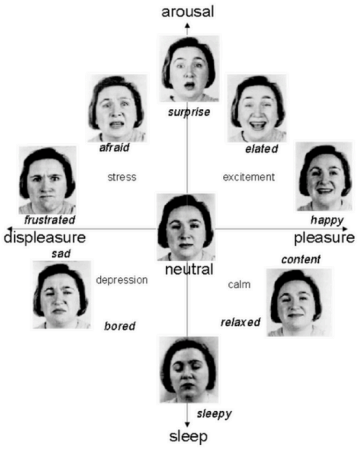
The project adopts Russell's Circumplex Model, which classifies emotions based on arousal (level of activation) and valence (emotional value) [10]. As shown in Figure 2.1, emotions are distributed across four quadrants:

- High Arousal, Pleasure: Happy and elated.
- High Arousal, Displeasure: Frustrated and afraid.
- Low Arousal, Displeasure: Sad and bored.
- Low Arousal, Pleasure: Content and relaxed.

Some emotions are positioned along the axes:

- Surprise has high arousal but neutral valence, representing a reaction rather than a sustained emotion.
- Sleep is classified as low arousal but is a physical state not an emotion.
- Neutral lies at the center, representing the absence of strong emotional stimuli.

Figure 2. 1 Russell's Circumplex Model [11]



For practical relevance, the project focuses on four emotions: happy, sad, angry, and neutral, aligning with FER-2013 and music recommendation relevance [10]:

- **Happy**: High-arousal positive emotions, replacing nuanced states like elated for easier classification, suited for fast-tempo playlists .
- **Sad**: Low-arousal negative emotions, associated with slow-tempo playlists.
- **Angry**: High-arousal negative emotions, suitable for intensity playlists.
- **Neutral**: A baseline category for ambiguous expressions.

Other FER-2013 emotions like disgust and fear are excluded due to their underrepresentation and limited relevance to music recommendation. Similarly, content and relaxed overlap with neutral in arousal/valence terms, making them redundant.

2.4 Methods and Techniques

2.4.1 Deep Learning Approaches

Deep learning algorithms are the core of FER, enabling models to identify complex facial patterns. Table 2.3 compares the performances of various deep learning models across datasets in the reviewed studies.

Table 2. 3 FER models performance Comparison

Dataset	**Model**	**Accuracy**
FER-2013	CNN [12]	58.9%
	Lightweight CNN + MTCNN [12]	67%
	Hybrid DCNN + Haar Cascade [13]	70.04%
	ResNet-18 (Pretrained) [14]	71.89%
FED-RO	ResNet-18 + Attention [16]	pACNN: 64.25%, gACNN: 66.50%
	VGG16 + Attention [16]	pACNN: 64.22%, gACNN: 65.27%
KDEF	Lightweight CNN + MTCNN [12]	87.71%
CK+	ResNet-18 [14]	95.96%
DAiSEE	DenseNet [4]	Does not report an overall accuracy, categories-specified accuracies: ● boredom: 81.17% ● engagement: 94.85% ● confusion: 90.96% ● frustration: 95.85%
MMI	Hybrid CNN-LSTM [17]	78.61%

Models trained on FER-2013 typically achieve lower accuracy than those trained on CK+ due to lower image quality and real-world variations. However, insights from models trained on other dataset remain valuable for FER research. For instance, attention mechanisms integrated into CNNs, as demonstrated in FED-RO, enhance recognition by focusing on key facial regions, mitigating occlusion challenges. Similarly, ResNet and DenseNet architectures enhance feature extraction, enabling better accuracy in FER tasks. These findings highlight the importance of advanced CNN models and advanced optimization techniques, forming a foundation for this project's selection and refinement.

1. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are fundamental to FER due to their ability to extract hierarchical spatial features from facial images. THey consist of three main layers [17]:

- Convolutional Layer: Extract low-level features like edges and textures using filters applied to local image regions.
- Pooling Layer: Downsample dimensionality using average or max pooling while retaining essential information.
- Fully Connected Layer: Combines extracted features for final classification.

Deeper CNN layers learn more abstract patterns such as facial textures and shapes, however, it faces challenges such as vanishing gradients, overfitting on small datasets, and vulnerability to pose and lighting variations [21]. These challenges highlight the importance of preprocessing techniques and architectural enhancements in the FER workflow.
For instance, a standalone CNN achieved 58.9% accuracy on FER-2013, but integrating MTCNN for face detection improved accuracy to 67%, demonstrating the importance of preprocessing and architectural enhancements in the FER workflow [12].

2. Advanced CNN Architectures - ResNet, DenseNet, VGG

Advancements in CNN architectures have introduced structural improvements that optimize feature learning and computational efficiency to address CNN limitations:

1) **ResNet (Residual Networks)**
   ResNet introduces residual connections, which allow information to bypass multiple layers, facilitating that gradients can flow smoothly during backpropagation, enabling deeper networks without performance degradation and mitigating the vanishing gradient issue [22]. Its depth is indicated by numbers (e.g., ResNet-18, ResNet-50). ResNet-18, with 8 residual blocks, is efficient for computationally constrained tasks. In “Facial Emotion Recognition Based on Sobel-ResNet”, ResNet-18 achieved 71.89% accuracy on FER-2013, demonstrating its effectiveness in FER applications [14].

2) **DenseNet (Dense Convolutional Networks)**
   DenseNet improves feature propagation by directly connecting all layers, ensuring efficient feature reuse and improved information flow [22]. It reduces redundancy and overfitting while requiring fewer parameters to retain information compared to traditional deep CNNs, which is particularly beneficial for training low resolution datasets like FER-2013. Based on the reviewed study, DesNet achieved 94.85% accuracy for engagement and 81.71% for boredom, demonstrating its strength in multi-label classification [15]. However, its dense connection increases computational demands, affecting real-time feasibility.

3) **VGG (Visual Geometry Group)**
VGG uses stacked sequential 3x3 convolutional layers for hierarchical feature extraction [22]. While it provides strong features extraction capabilities, VGG requires high parameters and computational power compared to other architectures, making it less ideal for resource-constrained tasks [23]. Variants like VGG16 and VGG19 are widely used in FER but are impractical for real-time applications due to their inefficiency. In "Occlusion Aware Facial Expression Recognition Using CNN with Attention Mechanism", VGG16 with attention achieved 65% accuracy on FED-RO, demonstrating its effectiveness for FER tasks requiring hierarchical feature extraction [16].

3. Attention Mechanisms

FER models often struggle with misclassification due to variations in facial features and expression intensity. Attention mechanisms mitigate these challenges by directly prioritizing key facial regions while ignoring irrelevant information [24]. Several attention-based approaches have been explored in the study.

The study on "Occlusion-Aware Facial Expression Recognition Using CNN with Attention Mechanism" demonstrated that pACNN and gACNN enhance classification performance, achieving 64.5% and 66.5% accuracy respectively [16].

- **Patch-based Attention (pACNN):** A form of spatial attention that assigns weights to specific facial landmarks such as eyes, nose, mouth to focus on key regions via patch-gated units.
- **Global Attention (gACNN):** A form of channel attention that applies weights across the entire image to capture broader feature relationships via global-gated units.

Both methods process spatial and channel attention separately, limiting their ability to capture local and global features simultaneously. To address this, **Convolutional Block Attention Module (CBAM)** combines both spatial and channel attention, further refining feature extraction while maintaining computational efficiency. Its dual attention mechanism enhances FER model performance while maintaining computational efficiency due to its lightweight structure [26]. It dynamically adjusts attention across dimensions, improving adaptability to facial structures and expression variations.

The study "Emotion Recognition with Facial Attention and Objective Activation Functions" demonstrates that CBAM-enhanced models consistently outperformed non-attention models on FER-2013 [26] as shown in Table 2.4.

Table 2. 4 Impact of CBAM on advanced CNN model accuracy (FER-2013) [26]

Model	**Accuracy**	**Model with CBAM**	**Accuracy**
VGG-16	60.66%	VGG-16 + CBAM	63.46%
VGG-19	60.92%	VGG-19 + CBAM	64.07%
ResNet-50	58.61%	ResNet-50 + CBAM	59.90%
ResNet-101	58.67%	ResNet-101 + CBAM	60.92%

Despites its benefit, attention mechanisms may overemphasize certain facial regions, leading to misclassifications of emotions that share subtle facial cues [25].

2.4.2 Data Preprocessing, Augmentation, and Training Strategies

Efficient data preparation is essential for improving the input data quality and ensuring balanced evaluation of the dataset. This section explores some strategies that are commonly used to mitigate challenges in FER datasets.

1. Haar Cascade
Haar Cascade is a machine learning-based algorithm used for detecting and cropping facial regions by analyzing Haar-like features and using a cascade of classifiers [27]. It improves FER by focusing on critical facial features and removing background noise. In "Hybrid Facial Expression Recognition (FER2013) Model" [13], Haar Cascade standardized input data, improving emotion recognition. Though its performance can be impacted by poor lighting and occlusions, this study proved its effectiveness by refining input quality, ensuring the model focuses on facial expressions for better emotion recognition.

2. Data Augmentation
Data augmentation is typically used techniques to artificially increase the size and variety of the dataset, improving generalization by introducing variations in pose, lighting, and occlusions [28]:

- **Random Horizontal Flip:** Mirrors images to increase dataset size.
- **Random Rotation:** Rotates images by a specific degree to increase pose diversity.
- **Brightness and Contrast Adjustment:** Simulates lighting variations to handle inconsistencies in image exposure and contrast effectively.
- **Random Erasing (Occlusion Simulation):** Obscures facial regions to simulate real-world occlusions like glasses and masks.
- **Elastic Transformation:** Introduces natural distortions in facial expressions.
- **Affine Transformations:** Applying scaling, translation, and shearing to improve generalization.

Studies show that combining multiple augmentation techniques significantly enhances FER performance, particularly for datasets with limited samples for certain emotions.

3. Loss Functions

FER datasets, including FER-2013, often exhibit class imbalance, where some classes are underrepresented. Adjusting loss functions helps mitigate bias:

- **Weighted Cross-Entropy Loss:** Penalizes misclassification of underrepresented classes to counteract class imbalance, guiding the model to learn patterns from minority classes more effectively [18].
- **Balanced Cross-Entropy Loss:** Automatically adjusts class contributions to enable each class contributes equally to the training process, leading to a more balanced classification performance [38].
- **Focal Loss:** Focuses on hard-to-classify instances while reducing emphasis on easy cases, making it effective for imbalanced data [38].

2.4.3 Evaluation Metrics

To evaluate the performance of FER models, the following metrics are commonly adopted in the reviewed FER studies [37]:

1. **Accuracy**: Measures the proportion of correctly classified instances among all instances.
   $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
2. **Precision**: Indicates the proportion of predicted positive instances that are correct, highlighting the reliability of positive predictions.
   $$Precision = \frac{TP}{TP + FP}$$
3. **Recall**: Indicates the proportion of actual positive instances that were correctly identified by the model.
   $$Recall = \frac{TP}{TP + FN}$$
4. **F1-Score**: Balances precision and recall, particularly useful for evaluating models with imbalanced class.
   $$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$
5. **Confusion Matrix**: Provide a visual summary of classification results, displaying the counts of true positives, false positives, true negatives, and false negatives, which helps to identify misclassification patterns across emotion class.

2.5 Tools and Technologies

2.5.1 FER Tools and Libraries

This project utilizes standard deep learning frameworks, preprocessing tools, and evaluation libraries for model training, ensuring efficiency in FER pipeline development. Table 2.5 summarizes key tools.

Table 2. 5 Summary of tools and libraries for FER model training

Category	**Tools/ Libraries**	**Use Case**
Deep Learning Frameworks	TensorFlow, Keras, PyTorch.	Training CNNs, integrating attention mechanisms, and fine tuning models. - TensorFlow and Keras: Simple and easy to use. - PyTorch: More flexible, suitable for custom model implementation.
Data Preprocessing	NumPy, Pandas, OpenCV	Standard image manipulation, resizing, and augmentation.
Visualization	Matplotlib, Seaborn, TensorBoard	Visualizing datasets, performance metrics.
Evaluation	Scikit-learn	Computing model performance metrics such as classification reports, confusion matrices.
Hardware Acceleration	Google Colab, Kaggle Kernels, CUDA/cuDNN	Accelerating training with GPUs/TPUs support: - Cloud resources: Google Colab, Kaggle Kernels offer cost-free access resources. - Local Resources: CUDA/cuDNN accelerates training on local GPUs.
Webcam Integration	OpenCV, PIL, Jupyter Notebook Widgets, IPython.display, Google Colab Output (eval_js),	Enabling real-time webcam image capture. - OpenCV, PIL: Processing live images. - IPython.display, eval.js: Activating webcam in Google Colab. - Jupyter Notebook Widgets: Activating webcam in local Jupyter Notebook environments.

2.5.2 Music Streaming Service APIs

Music streaming services such as Spotify, Tidal and Apple Music provide public APIs that allow developers to access extensive music libraries and implement recommendation systems within their own designed applications. These APIs enable personalized music experiences by offering features such as playlist management, metadata access, and user interaction insights.

Each API has distinct strengths and limitations, as summarized in Table 2.6. Among the reviewed APIs, Spotify stands out as a versatile and widely accessible option, making it particularly suitable for academic projects and prototypes. In contrast, Apple Music and TIDAL impose access restrictions, either through paid developer accounts or limited public API availability, which can affect their usability in research applications.

Table 2. 6 Comparison of music streaming service APIs

API	**Strengths**	**Limitations**
Spotify API [29]	Free to access, extensive music library	Free-tier restrictions on user specific data and higher-rate limits.
Apple Music API [30]	Able to integrate with Apple ecosystems, extensive music library	Require a paid Apple developer account for access.
TIDAL API [31]	High-quality audio, exclusive content access	Limited public access, minimal documentation and community support.

CHAPTER 3: DESIGN

3.1 Design Foundations

Building on insights from the literature review, this section defines the design principles for the FER model. Chapter 2 reviewed key methodologies, including CNN-based architectures, attention mechanisms, and data preprocessing techniques, which serves as the foundation for optimizing FER performance. This design will address the limitations of the FER-2013 dataset, select a suitable model architecture and refining techniques to improve emotion classification accuracy.

3.1.1 Dataset Considerations and Challenges

The FER-2013 dataset presents specific challenges, including low image resolution and class imbalance. These factors contribute to suboptimal model performance, necessitating advanced deep learning techniques. While alternative datasets like CK+ and Affect offer higher resolution and more diverse annotations, FER-2013 is still recognized as a robust benchmark in this academic project.
Addressing FER-2013's challenges requires:

- A robust CNN-based architecture capable of extracting key facial features despite low resolution.
- Preprocessing and augmentation strategies to enhance image quality and reduce the effects of class imbalance.
- Validation techniques to ensure model generalizability across different facial expressions

3.1.2 Model Selection: ResNet18 with CBAM Enhancement

Among the reviewed deep learning architecture, ResNet18 has demonstrated its strong feature extraction capabilities while maintaining computational efficiency in FER tasks, which achieved 71.89% accuracy on FER-2013 in prior studies [16], making an ideal starting point for this project. Unlike traditional CNNs, ResNet18 employs residual connections, mitigating the vanishing gradient problem and allowing deeper feature learning.

To further improve accuracy, the Convolutional Block Attention Module (CBAM) complements ResNet18. CBAM enhances the model's ability to focus on critical regions while minimizing computational overhead by introducing spatial and channel attention mechanisms, thereby improving emotion classification. Prior research has shown that CBAM integration refines feature learning, compensating for FER-2013's low-resolution constraint [26].

3.1.3 Hypotheses

Based on these findings, two hypotheses are formulated:

1. **Adding CBAM to ResNet18 will improve facial emotion recognition performance on FER-2013 compared to a baseline CNN.**
   This will be validated by comparing the final test evaluation metrics of both models. The training and validation loss curves will also be monitored to confirm that the proposed model maintains stable generalization, avoiding overfitting during training.

2. **A higher accuracy FER model (ResNet18 + CBAM) will enable more relevant emotion-aligned music playlists.**
   Emotion recognition accuracy directly affects the reliability of music playlists mapping when integrated with a music streaming API. This will be validated by assessing the accuracy of real-time facial emotion predictions and reviewing whether the mapped playlist aligns appropriately with the detected emotion.

These hypotheses align with the project's objectives and link improved emotion recognition with enhanced personalized music experience.

3.2 Model Design

3.2.1 Baseline Model: Convolutional Neural Network Model

The baseline CNN model is designed as a common convolutional neural network structure, following a simple and effective architecture commonly used for image classification tasks. Its primary goal is to establish a foundational benchmark for evaluating the effectiveness of the proposed ResNet18-CBAM model. The baseline follows the conventional design principles of CNN architecture:

1. **Input Layer**: The model accepts grayscale images of size 48×48 as input. Grayscale images are processed without the need for color channels, simplifying the data pipeline and reducing computational overhead.
2. **Convolutional Layers**:
    - The architecture comprises three convolutional layers. Each layer employs 3×3 filters, which are effective for capturing local spatial features like edges, textures, and simple patterns.
    - The first convolutional layer uses 32 filters to extract low-level features. These features progressively increase in complexity as the number of filters doubles in subsequent layers: 64 filters in the second layer and 128 filters in the third. This design ensures that the network can learn hierarchical feature representations, from basic contours to abstract emotional cues.
3. **Activation Function:** After each convolutional operation, ReLU activation is applied to enable the model to learn complex patterns more effectively.
4. **Pooling Layer**: Each convolutional layer is followed by a 2×2 max pooling layer, which reduces the spatial dimensions for feature maps by half, useful for handling variations in facial expression positioning.
5. **Flattening and Dense Layers**:
    - After the final pooling layer, the feature maps (6x6x128) are flattened into a 1D feature vector. This vector serves as input to the fully connected (dense) layers.
    - The first dense layer consists of 128 neurons, which combine and learn abstract representations of the extracted features. The final dense layer outputs probabilities for the four emotion classes (happy, sad, angry, neutral) using the softmax activation function
6. **Output**: The model will output a probability distribution over four emotion classes, and the emotion with the highest probability is selected as the predicted emotion.

3.2.2 Proposed Model: ResNet18-CBAM

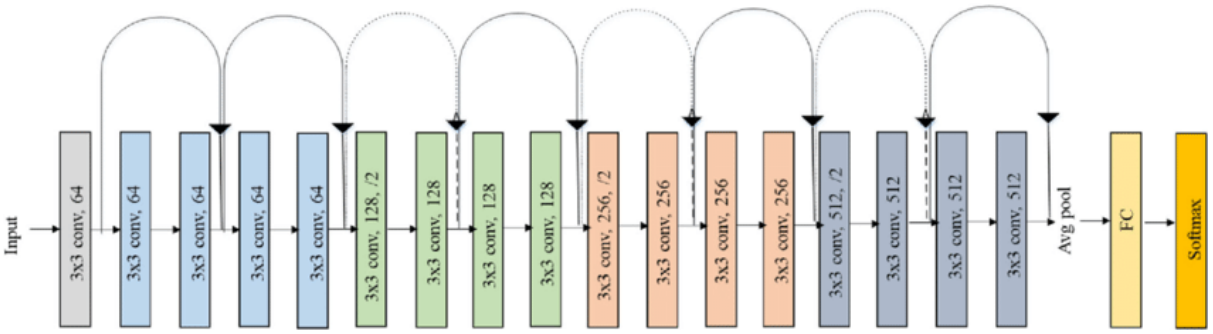
2.2.1 ResNet18 Architecture

ResNet-18 is used as the backbone of the proposed model, chosen for its lightweight design and hierarchical feature extraction capability. Figure 3.1 illustrates the architecture of ResNet18 which consists of 18 layers, organized into five main stages, where each stages extracts increasing abstract features for effective image classification:

1. **Conv1**: A single 3×3 convolutional layer with 64 filters, extracting low-level features like edges and textures from the input.
2. **Conv2_x**: Two residual blocks, each consisting of two 3×3 convolutional layers with 64 filters, processing low-level features.
3. **Conv3_x**: Two residual blocks with 128 filters, capturing mid-level features like shapes and patterns.
4. **Conv4_x**: Two residual blocks with 256 filters, focusing on high-level features such as facial regions.
5. **Conv5_x**: Two residual blocks with 512 filters, extracting abstract features such as emotional expressions.

After these stages, global average pooling (GAP) reduces the spatial dimensions of the feature map to 1 x 1x 512. A fully connected (FC) layer then maps the feature vector to class probabilities(e.g. happy, sad, angry, neutral) via softmax activation function.

Figure 3. 1 ResNet18 architecture [32]



2.2.2 CBAM Architecture

CBAM enhances feature refinement by focusing the most relevant spatial and channel-wise information, improving the model's ability to classify emotions accurately. As illustrated in Figure 3.2, it operates in two stages:

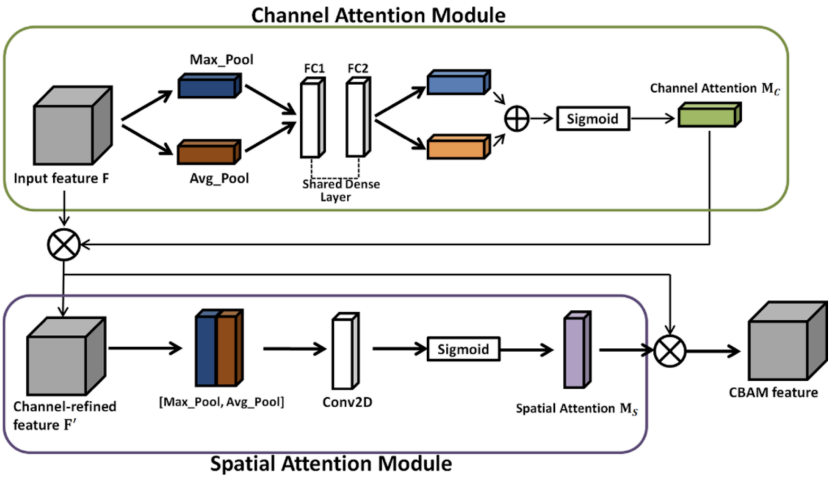
1. Channel Attention Module: Computes attention weights for the input feature map (F) by applying average pooling and max pooling, followed by a shared multi-layer

perceptron (MLP) and sigmoid activation function to generate channel-wise refined features.

2. Spatial Attention Module: Applies spatial attention weights to the channel-refined feature (F') by pooling along the channel dimension, preprocessing through a convolution layer, and generating the final refined features (F'').

These attention mechanisms prioritize critical facial landmarks, helpful in addressing FER-2013 challenges for FER.

Figure 3. 2 CBAM architecture [33]



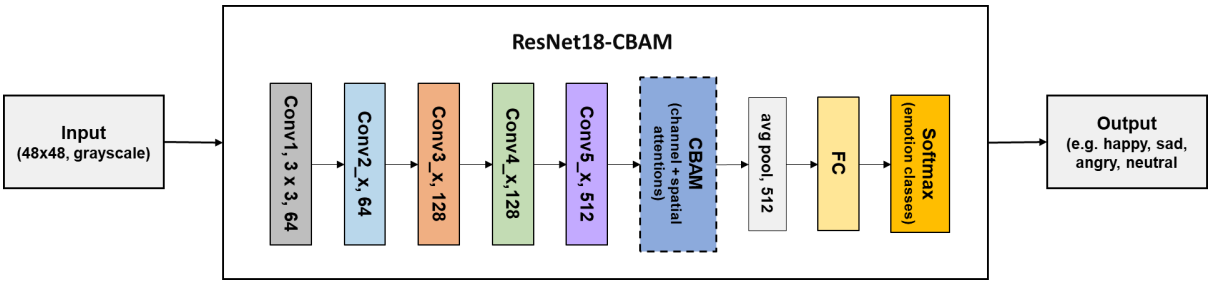
2.3.3 ResNet18-CBAM

In the proposed model, **ResNET18-CBAM**, integrates CBAM into ResNet18 after the Conv5_x stage. The decision is based on the role of Conv5_x in extracting high-level and abstract features critical for FER tasks, such as distinguishing subtle differences in facial expressions. Integrating CBAM at this stage refines these features by focusing on important regions and channels while deemphasizing irrelevant ones.

This strategic placement ensures a balance between performance improvement and computational efficiency. Adding CBAM at an earlier stage or every stage such as from Conv2_x to Conv5_x would significantly increase computational costs and model complexity unnecessarily, as earlier stages focus on low- to mid-level feature extraction. Focusing refinement at Conv5_x avoids redundant computation and maximizing the impact of CBAM on the final representation before emotion classification.

The workflow of the proposed ResNet18-CBAM architecture is illustrated in Figure 3.3 , highlighting the sequence of stages from input to emotion classification:

Figure 3. 3 The proposed ResNet18-CBAM architecture



3.2.3 Evaluation Plan

The evaluation of the proposed ResNet18-CBAM model involves two key aspects:

1. Model Performance Evaluation

The ResNet18-CBAM model will be evaluated against a baseline CNN and the three reviewed studies using the FER-2013 dataset. The evaluation metrics include:

- **Accuracy**: Measures the overall correctness of emotion classification.
- **Precision, Recall, and F-Score**: Evaluate the model's performance in handling class imbalance and identifying rare emotions.
- **Confusion Matrix**: Provides insights into class-specific misclassifications, assessing whether the proposed ResNet18-CBAM model improves classification for challenging emotions like "sad" and "neutral".

To ensure fair comparison, the baseline CNN and ResNet18-CBAM will be trained under identical conditions. Additionally, their performance will be analyzed in relation to prior FER-2013 reviewed studies, which reported accuracies of 67%, 70.04%, 71.89% [12,13,14]. This comparison will help quantify the impact of CBAM integration on classification accuracy, robustness, and the class imbalance handling.

2. Emotion-Aligned Music Playlist Mapping Evaluation

To validate the feasibility of integrating the model for emotion-aligned music playlist mapping, the following experiments will be conducted:

- **Webcam Capture and Preprocessing:** The system will capture facial expressions in real time through a webcam. Then, the captured image will be processed and normalized for emotion classification.
- **Emotion Prediction:** The ResNet18-CBAM model will classify the captured facial image into one of four predefined emotions categories: happy, sad, angry, neutral.
- **Music Streaming Service API Integration for Emotion-Based Playlist:** The detected emotions will be mapped to predefined music categories in the music

streaming service, which will fetch tracks corresponding to the emotional state. For example:

- Happy → Upbeat pop or dance music.
- Sad → Acoustic or mellow tracks.
- Angry → High-energy genres like rock or metal.
- Neutral → Ambient or classical music.

3.3 Tools and Technologies Justification

The technologies and methods chosen for this project address the challenges of FER-2013 and support the objective of emotion-aligned playlist mapping:

- **Data Inspection and Preprocessing**: Standard libraries such as Numpy and Pandas are used for handling and preprocessing data. Visualization tools like Matplotlib and Seaborn are used for visualizing dataset distribution, image samples, and model performance metrics, ensuring interpretability of results.
- **Deep Learning Frameworks**: PyTorch is preferred over Tensorflow and Keras for its flexibility and ease of debugging and customization, which simplifies the integration of custom components like CBAM in the ResNet18 architecture.
- **Music Streaming Service API**: The Spotify API is chosen due to its extensive catalog and availability of free access, which facilitates easy integration with the model to generate emotion-based playlists. Alternative APIs like Apple Music and Spotify require paid subscriptions, making Spotify the more accessible choice for research and prototyping.
- **Development Environment:** Google Colab is chosen for its free GPU/TPU resources, and built-in support for Python libraries, enabling efficient model training and testing while facilitating integration with tools like OpenCV and Spotify API without the need for local infrastructure. However, limitations such as session timeouts and memory constraints are acknowledged.

3.4 Project Timeline and Management

The project adopts an agile software development methodology, following a structured timeline defined milestones and deliverables, while retaining the flexibility to adapt to challenges and refinements as the project evolves. The timeline is divided into four key phases, each with specific objectives, deliverables, and review points to ensure steady progress. Figure 3.4 provides a visual overview of these phases, highlighting key milestones and enabling tracking of progress, while allowing adjustments to accommodate iterative improvements when necessary.

Phase1: Project Idea and Proposal (15 Oct - 11 Nov 2024)
This phase lasts approximately 4 weeks, focusing on topic selection, background research, and defining project objectives. Key activities include formulating the project idea, conducting preliminary literature research, and doing the video-based project presentation. The outcome of this phase is reviewed by the supervisor reviewing submitted to Coursera as part of the initial proposal.

Phase2: Theoretical Foundations and Model Design (11 Nov - 8 Dec 2024)
Starting after the video proposal submission, this phase lasts four weeks and focuses on detailed literature review, dataset analysis, and method and tools selection. The FER model architecture is designed in this phase to establish its technical feasibility. The work is aligned with the preliminary report submission, ensuring a structured foundation before implementation begins.

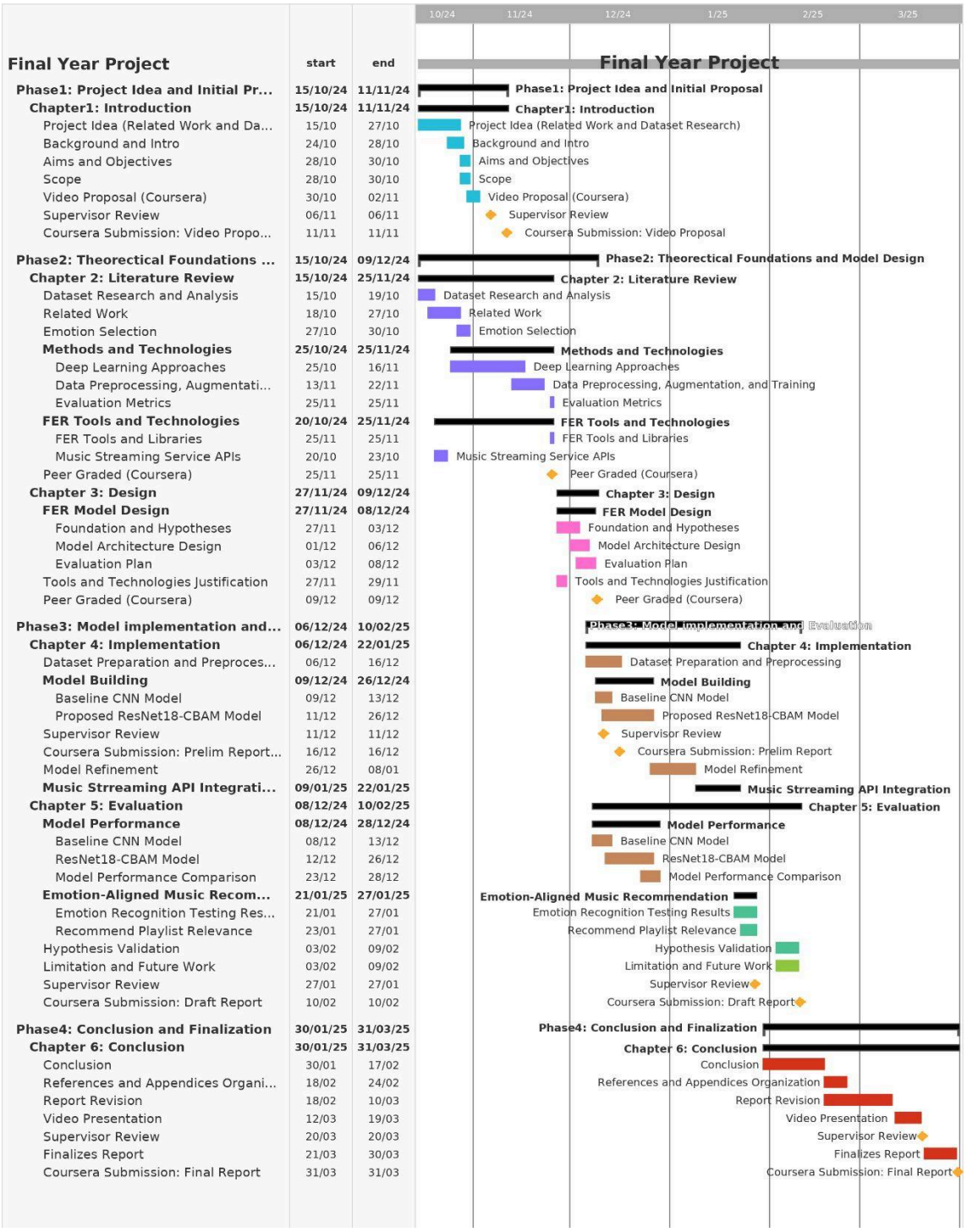
Phase3: Model Implementation and Evaluation (6 Dec - 9 Feb 2025)
This phase, spanning nine weeks, involves model development, evaluation, and integration with the music streaming API. Major tasks include:

- Dataset preprocessing and augmentation (6 - 12 Dec)
- Baseline and proposed model training and their evaluation (9 - 31 Dec)
- Model refinement and models performance comparison (1 - 18 Jan)
- Music API integration and testing (9 Jan - 9 Feb)

Given its complexity, this phase has the largest time allocation, allowing for risk management and performance optimization before the draft report submission.

Phase4: Conclusion and Finalization (18 Feb - 30 Mar 2025)
This final phase, lasting for five weeks, focuses on report writing, final review, and all submission. It includes summarizing key findings, ensuring clear documentation, and preparing for final Coursera submission and presentation.

Figure 3. 4 Gantt chart



CHAPTER 4: IMPLEMENTATION

4.1 Dataset Preparation

The FER-2013 dataset, imported via Kaggle API, is widely used for facial emotion recognition tasks. Each dataset folder contains images categorized into seven emotion classes: happy, sad, angry, neutral, surprise, disgust, and fear. To ensure its compatibility with the objectives of this project, the following preprocessing steps are taken:

1. **Dataset Reorganization:**
   The FER-2013 dataset is structured into train and test sets. The train set is used to train the model, while the test set is utilized for evaluation. The test set further contains two types of data:
   - **Public test**, which is commonly used for validation during model training.
   - **Private test** which is reserved for final evaluation.

   To facilitate easier implementation, the dataset is reorganized. The test directory is separated into two distinct subdirectories: *`publictest`* for validation and *`privatetest`* for final testing in order to ensure a clear evaluation pipeline.

Figure 4.1 Dataset reorganization

```
# Function to categorize privatetest and publictest in the test directory
# into subdirectories based on their filenames
def categorize_images(test_dir, privatetest_dir, publictest_dir):
    # Get all emotion subdirectories in the test directory
    emotion_folders = [f for f in os.listdir(test_dir) if os.path.isdir(os.path.join(test_dir, f))]

    for emotion in emotion_folders:
        emotion_path = os.path.join(test_dir, emotion)
        privatetest_emotion_path = os.path.join(privatetest_dir, emotion)
        publictest_emotion_path = os.path.join(publictest_dir, emotion)

        # Create subdirectories in privatetest and publictest
        os.makedirs(privatetest_emotion_path, exist_ok=True)
        os.makedirs(publictest_emotion_path, exist_ok=True)

        # Loop through files in the emotion folder
        for file_name in os.listdir(emotion_path):
            if file_name.startswith("PrivateTest_") and file_name.endswith(".jpg"):
                shutil.move(os.path.join(emotion_path, file_name), os.path.join(privatetest_emotion_path, file_name))
            elif file_name.startswith("PublicTest_") and file_name.endswith(".jpg"):
                shutil.move(os.path.join(emotion_path, file_name), os.path.join(publictest_emotion_path, file_name))
    print("PublicTest and PrivateTest have been categorized successfully.")


# Function to remove empty directories
def remove_empty_dirs(directory):
  # Check if include empty directories
  for folder in os.listdir(directory):
      folder_path = os.path.join(directory, folder)
      if os.path.isdir(folder_path):
          remove_empty_dirs(folder_path)
          # Remove the empty folder
          if not os.listdir(folder_path):
              os.rmdir(folder_path)
```

```
# Define paths for new subdirectories
privatetest_dir = os.path.join(test_dir, "privatetest")
publictest_dir = os.path.join(test_dir, "publictest")

# Create subdirectories for public and private test data
os.makedirs(privatetest_dir, exist_ok=True)
os.makedirs(publictest_dir, exist_ok=True)

# Distinguish privatetest and publictest from test dir
categorize_images(test_dir, privatetest_dir, publictest_dir)

# Remove empty directories
remove_empty_dirs(test_dir)
```

```
PublicTest and PrivateTest have been categorized successfully.
```

After executing the code, the dataset structure is updated as follows:

Initial Structure	After Reorganization
Figure 4.2 Initial dataset structure	*Figure 4.3 Reorganized dataset structure*

```
Folder: fer2013/train
  Folder: fer2013/train/neutral
  Folder: fer2013/train/sad
  Folder: fer2013/train/fear
  Folder: fer2013/train/disgust
  Folder: fer2013/train/surprise
  Folder: fer2013/train/happy
  Folder: fer2013/train/angry
Folder: fer2013/test
  Folder: fer2013/test/neutral
  Folder: fer2013/test/sad
  Folder: fer2013/test/fear
  Folder: fer2013/test/disgust
  Folder: fer2013/test/surprise
  Folder: fer2013/test/happy
  Folder: fer2013/test/angry
```

```
Folder: fer2013/train
  Folder: fer2013/train/surprise
  Folder: fer2013/train/neutral
  Folder: fer2013/train/sad
  Folder: fer2013/train/fear
  Folder: fer2013/train/angry
  Folder: fer2013/train/disgust
  Folder: fer2013/train/happy
Folder: fer2013/test
  Folder: fer2013/test/publictest
    Folder: fer2013/test/publictest/surprise
    Folder: fer2013/test/publictest/neutral
    Folder: fer2013/test/publictest/sad
    Folder: fer2013/test/publictest/fear
    Folder: fer2013/test/publictest/angry
    Folder: fer2013/test/publictest/disgust
    Folder: fer2013/test/publictest/happy
  Folder: fer2013/test/privatetest
    Folder: fer2013/test/privatetest/surprise
    Folder: fer2013/test/privatetest/neutral
    Folder: fer2013/test/privatetest/sad
    Folder: fer2013/test/privatetest/fear
    Folder: fer2013/test/privatetest/angry
    Folder: fer2013/test/privatetest/disgust
    Folder: fer2013/test/privatetest/happy
```

2. **Filtering Unrelated Classes:** To focus on the project's goals, unrelated emotion classes: surprise, fear, disgust are removed from the dataset.

Figure 4. 4 Code for fIltering unrelated classes

```
# Function to remove unrelated classes
def filter_classes(directory, relevant_classes):
    for class_name in os.listdir(directory):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path) and class_name not in relevant_classes:
            # Safely remove the folder that is not relevant
            print(f"Removing unrelated folder: {class_path}")
            shutil.rmtree(class_path)
```

```
# Retain the four relevant classes
relevant_classes = ['happy','sad','angry','neutral']
# Filter unrelevant dataset folder
filter_classes(train_dir, relevant_classes)
filter_classes(publictest_dir, relevant_classes)
filter_classes(privatetest_dir, relevant_classes)
```

The final structure of the dataset is well-structured and optimized for model building, with only the relevant emotions retained.

Figure 4. 5 Final dataset structure after fIltering

```
Folder: fer2013/train
  Folder: fer2013/train/neutral
  Folder: fer2013/train/sad
  Folder: fer2013/train/angry
  Folder: fer2013/train/happy
Folder: fer2013/test
  Folder: fer2013/test/publictest
    Folder: fer2013/test/publictest/neutral
    Folder: fer2013/test/publictest/sad
    Folder: fer2013/test/publictest/angry
    Folder: fer2013/test/publictest/happy
  Folder: fer2013/test/privatetest
    Folder: fer2013/test/privatetest/neutral
    Folder: fer2013/test/privatetest/sad
    Folder: fer2013/test/privatetest/angry
    Folder: fer2013/test/privatetest/happy
```

3. **Dataset Visualization:** To confirm the correctness of the reorganization and filtering process, a subset of images from the dataset is visualized.

Figure 4. 6 Dataset visualization

4.2 Model Implementation

4.2.1 Baseline CNN Model

The baseline CNN model serves as a foundation for evaluating the effectiveness of the proposed advanced architectures.

1. **Dataset Preprocessing:** The preprocessing steps to prepare FER-2013 for model training includes:

    - Normalization: Image pixel values are scaled to the range [0,1] to improve training efficiency and convergence.
    - Tensor Conversion: Images and labels are converted into PyTorch tensors with an additional channel dimension to align with CNN input requirements.
    - Batch Loading: Data loaders are created with a batch size of 32 and shuffling enabled for training and validation.

    The following function is used for preprocessing:

    Figure 4. 7 Dataset preprocessing for baseline CNN

```python
# Function to preprocess dataset
# by loading images from directory, resizes, normalization, and converts to numpy
def preprocess_dataset(directory, target_size, normalize=True):
    images = []
    labels = []
    class_names = sorted(os.listdir(directory))

    # Iterate through class folders
    for label, class_name in enumerate(class_names):
        class_path = os.path.join(directory, class_name)
        if os.path.isdir(class_path):
            for file_name in os.listdir(class_path):
                file_path = os.path.join(class_path, file_name)
                # Load image in grayscale
                img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
                if img is not None:
                    # Resize to target size
                    img = cv2.resize(img, target_size)
                    images.append(img)
                    labels.append(label)

    # Convert to NumPy arrays
    images = np.array(images, dtype='float32')
    labels = np.array(labels, dtype='int')

    # Normalize pixel values
    if normalize:
        images = images / 255.0  # Scale to [0, 1]

    # Add channel dimension and convert to PyTorch tensors
    images_tensor = torch.tensor(images, dtype=torch.float32).unsqueeze(1)  # (N, 1, H, W)
    labels_tensor = torch.tensor(labels, dtype=torch.int64)

    # Create TensorDataset
    dataset = TensorDataset(images_tensor, labels_tensor)

    return dataset, class_names
```

Figure 4. 8 Dataset preprocessing for baseline CNN

```
train_dataset, class_names_train = preprocess_dataset(train_dir, target_size=(48, 48))
val_dataset, class_names_val = preprocess_dataset(publictest_dir, target_size=(48, 48))
test_dataset, class_names_test = preprocess_dataset(privatetest_dir, target_size=(48, 48))

assert class_names_train == class_names_val == class_names_test, "Class names do not match across datasets!"

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

2. **Model Architecture:** The baseline CNN model is a simple and effective architecture designed for FER-2013's grayscale nature. It consists of three convolutional layers, each followed by ReLU activation and max pooling to progressively extract features and reduce spatial dimensions. The final layers include a fully connected dense layer and an output softmax layer for emotion classification across predefined categories. The model's architecture is as follows:

Figure 4. 9 Code for baseline CNN architecture

```
class CNNBaseline(nn.Module):
    def __init__(self, num_classes):
        super(CNNBaseline, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)  # First convolutional layer
        self.relu1 = nn.ReLU()                                   # ReLU activation
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)       # Max pooling
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1) # Second convolutional layer
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1) # Third convolutional layer
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)      # Final pooling
        self.flatten = nn.Flatten()                              # Flatten layer
        self.fc1 = nn.Linear(128 * 6 * 6, 128)                  # Fully connected layer
        self.fc2 = nn.Linear(128, num_classes)                  # Output layer for classification

    def forward(self, x):
        x = self.conv1(x)   # Apply first convolution
        x = self.relu1(x)   # Apply ReLU
        x = self.pool1(x)   # Apply Max Pooling
        x = self.conv2(x)   # Apply second convolution
        x = self.relu2(x)
        x = self.pool2(x)
        x = self.conv3(x)   # Apply third convolution
        x = self.relu3(x)
        x = self.pool3(x)   # Final pooling
        x = self.flatten(x) # Flatten for fully connected layers
        x = self.fc1(x)     # Apply first fully connected layer
        x = self.fc2(x)     # Apply output layer
        return x
```

Model summary:

Figure 4. 10 Baseline CNN model summary

```
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 48, 48]             320
              ReLU-2          [-1, 32, 48, 48]               0
         MaxPool2d-3          [-1, 32, 24, 24]               0
            Conv2d-4          [-1, 64, 24, 24]          18,496
              ReLU-5          [-1, 64, 24, 24]               0
         MaxPool2d-6          [-1, 64, 12, 12]               0
            Conv2d-7         [-1, 128, 12, 12]          73,856
              ReLU-8         [-1, 128, 12, 12]               0
         MaxPool2d-9           [-1, 128, 6, 6]               0
           Flatten-10                [-1, 4608]               0
           Linear-11                 [-1, 128]         589,952
           Linear-12                   [-1, 4]             516
================================================================
Total params: 683,140
Trainable params: 683,140
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.25
Params size (MB): 2.61
Estimated Total Size (MB): 4.87
----------------------------------------------------------------
```

3. **Training Process:** The training loop iterates over batches of data, performing forward passes, loss computation, backpropagation, and weight updates. Validation loss and accuracy are tracked after each epoch. The model is trained using the following setup:

    - Loss function: Weighted cross-entropy loss is used to mitigate class imbalance.
    - Optimizer: Adam optimizer with a learning rate of 0.001 is selected for efficient convergence.
    - Early Stopping: Training is halted after three consecutive epochs without improvement in validation loss to prevent overfitting.

Figure 4. 11 Baseline CNN training setup

```
# Compute class weights o handle imbalanced classes
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.arange(num_classes),
    y=y_train_np
)
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Class weights: {class_weights}")

# Configure the optimizer and loss function
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam optimizer
criterion = nn.CrossEntropyLoss(weight=class_weights_tensor)

# Set up early stopping params
early_stopping_patience = 3
best_val_loss = float('inf')  # Best validation loss
early_stopping_counter = 0  # Track consecutive epochs with no improvement
```

Figure 4. 12 Code for baseline CNN training process

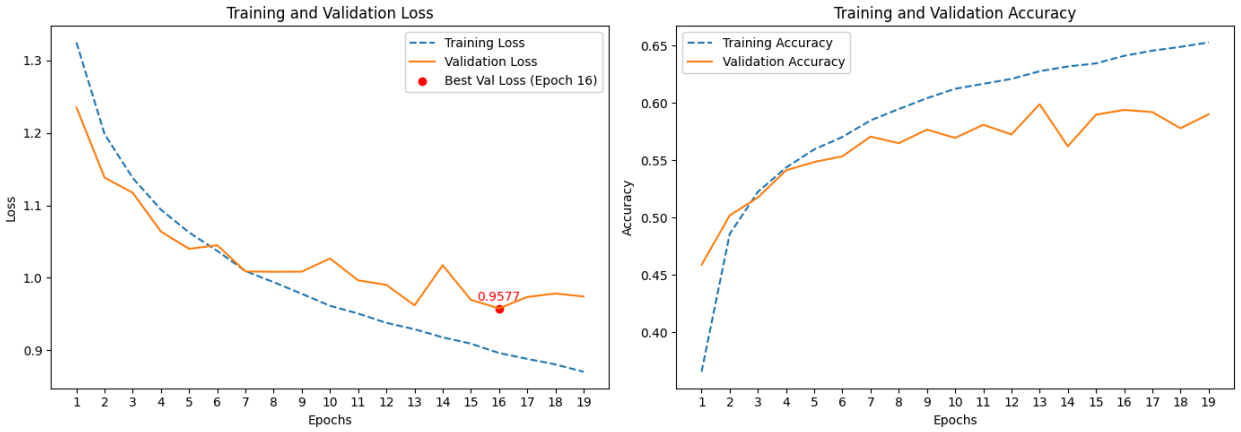
```
# Lists to store training and validation metrics
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Define the num of epochs for training
epochs = 20

# Training and Validation loop
for epoch in range(epochs):
  # ---------- Training ---------
  model.train()
  train_loss = 0.0
  correct_train = 0
  total_train = 0

  # Iterate over the training data in batches
  for images, labels in train_loader:
      images, labels = images.to(device), labels.to(device)

      optimizer.zero_grad()              # Clear gradients
      outputs = model(images)            # Forward pass
      loss = criterion(outputs, labels)  # Compute loss
      loss.backward()                    # Backward pass
      optimizer.step()                   # Update weights
      train_loss += loss.item()          # Accumultae batch loss

      # Compute training accuracy
      _, predicted_train = torch.max(outputs, 1)
      total_train += labels.size(0)
      correct_train += (predicted_train == labels).sum().item()

  # Calculate average training loss and accuracy for epoch
  train_loss /= len(train_loader)
  train_accuracy = correct_train / total_train
  train_losses.append(train_loss)
  train_accuracies.append(train_accuracy)
```

Figure 4. 13 Code for baseline CNN training process

```
# ---------- Validation ---------
model.eval()
val_loss = 0.0
correct = 0
total = 0

# Disable gradient computation
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)

        outputs = model(inputs)            # Forward pass
        loss = criterion(outputs, labels)  # Compute validation loss
        val_loss += loss.item()            # Accumulate batch loss

        # Compute validation accuracy
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

# Calculate average validation loss and accuracy for epoch
val_loss /= len(val_loader)
val_losses.append(val_loss)
val_accuracy = correct / total
val_accuracies.append(val_accuracy)

print(f"Epoch {epoch + 1}/{epochs}, "
      f"Training Loss: {train_loss:.4f},"
      f"Validation Loss: {val_loss:.4f},"
      f"Validation Accuracy: {val_accuracy:.4f}")

# --------- Early stopping ---------
if val_loss < best_val_loss:
    best_val_loss = val_loss    # Update the best validation loss
    early_stopping_counter = 0  # Reset the counter for early stopping
    torch.save(model.state_dict(), "best_baseline.pth")  # Save the best model
else:
    early_stopping_counter += 1
    if early_stopping_counter >= early_stopping_patience:
        print("Early stopping triggered.")
        break
```

3) Validation Process: The public test set of FER-2013 is used to monitor validation loss and accuracy during training. Throughout training, the training accuracy steadily increases, while validation accuracy plateaus around 60%. Validation loss fluctuates slightly after an initial decline, indicating overfitting despite the simplicity of the architecture. Early stopping is applied after 19 epochs when no further validation improvement is observed. This validation process highlighted the baseline CNN's limited generalization capacity, especially in the later training stages, motivating the exploration of more advanced models to improve robustness.

Figure 4. 14 Baseline CNN training and validation loss/accuracy



4.2.2 ResNet18-CBAM model

The proposed model builds upon the ResNet-18 architecture by integrating CBAM to enhance feature refinement.

1. **Dataset Preprocessing:** To optimize the data for ResNet-based architectures, images are resized to a target size of 224×224 pixels to ensure compatibility with the ResNet input requirements. Several data augmentation techniques are applied to improve diversity and generalization, including:

    - Random horizontal flipping: Simulate mirrored expressions.
    - Random rotations: Introduces small variations in orientation.
    - Affine transformations: Adds slight positional and scale variations.
    - Random erasing: simulate occlusions, such as partial face coverage.

    Normalization is applied using the dataset's mean and standard deviation:

Figure 4. 15 Data augmentation pipeline

```
# Augmentation pipeline for training dataset
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.RandomAffine(degrees=10, translate=(0.05, 0.05), scale=(0.95, 1.05)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4922], std=[0.2486]),
    transforms.RandomErasing(p=0.3, scale=(0.02, 0.15), ratio=(0.3, 3.3), value=0),
])

# No augmentation pipeline for validation and testing dataset
val_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4922], std=[0.2486])
])
```

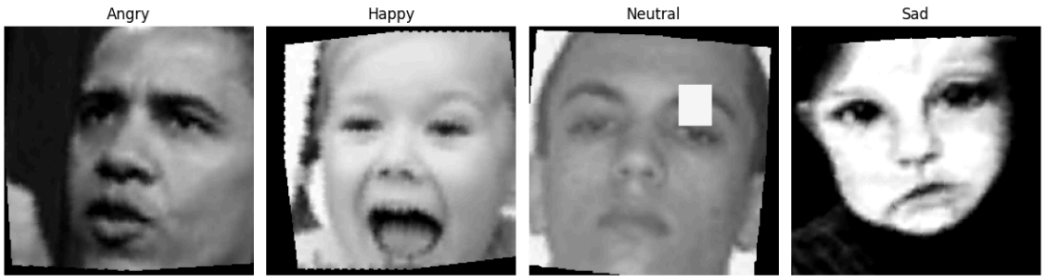
All transformations are handled through a CustomAugmentedDataset class, ensuring augmented data is directly accessible through PyTorch DataLoaders.

Figure 4. 16 Code for data augmentation preprocessing

```
class CustomAugmentedDataset(Dataset):
    def __init__(self, images, labels, augmentations):
        """
        Args:
            images (numpy.ndarray): Array of preprocessed images (N, H, W).
            labels (numpy.ndarray): Array of labels (N,).
            augmentations (dict): A dictionary mapping class indices to augmentation pipelines.
        """
        self.images = images
        self.labels = labels
        self.augmentations = augmentations

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]

        # Convert grayscale image (NumPy) to PIL Image for PyTorch transforms
        image = Image.fromarray(image.astype('uint8'))

        # Apply class-specific augmentation
        augmentation = self.augmentations[label]
        image = augmentation(image)

        return image, label
```

A few images from the dataset are shown after the data preprocessing:

Figure 4. 17 Augmented dataset visualization



2. **Model Architecture:** The ResNet18-CBAM model builds on the ResNet18 backbone and integrates Convolutional Block Attention Module (CBAM), This module is inserted after the final convolutional block (Conv5_x) to refine the extracted feature maps before global pooling and classification. Additionally, the first convolutional layer is modified to accept single-channel grayscale images, aligning with FER-2013's input format. The final fully connected layers map the extracted and refined features to the four emotion classes: Happy, Sad, Angry, Neutral.

Figure 4. 18 Code for CBAM architecture

```python
# Channel Attention Module
class ChannelAttention(nn.Module):
    def __init__(self, in_channels, reduction=16):
        super(ChannelAttention, self).__init__()
        self.fc1 = nn.Linear(in_channels, in_channels // reduction, bias=False)
        self.fc2 = nn.Linear(in_channels // reduction, in_channels, bias=False)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        b, c, _, _ = x.size()
        avg_pool = torch.mean(x, dim=(2, 3))
        max_pool = torch.amax(x, dim=(2, 3))
        avg_out = self.fc2(self.relu(self.fc1(avg_pool)))
        max_out = self.fc2(self.relu(self.fc1(max_pool)))
        out = self.sigmoid(avg_out + max_out).unsqueeze(2).unsqueeze(3)
        return x * out


# Spatial Attention Module
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()
        self.conv = nn.Conv2d(2, 1, kernel_size=kernel_size, padding=kernel_size // 2, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_pool = torch.mean(x, dim=1, keepdim=True)
        max_pool, _ = torch.max(x, dim=1, keepdim=True)
        concat = torch.cat([avg_pool, max_pool], dim=1)
        out = self.sigmoid(self.conv(concat))
        return x * out


# CBAM Block
class CBAM(nn.Module):
    def __init__(self, in_channels, reduction=16, kernel_size=7):
        super(CBAM, self).__init__()
        self.channel_attention = ChannelAttention(in_channels, reduction)
        self.spatial_attention = SpatialAttention(kernel_size)

    def forward(self, x):
        x = self.channel_attention(x)
        x = self.spatial_attention(x)
        return x
```

Figure 4. 19 Code for ResNet18-CBAM architecture

```
# ResNet18 with CBAM for Grayscale Input
class ResNet18_CBAM(nn.Module):
    def __init__(self, num_classes):
        super(ResNet18_CBAM, self).__init__()
        self.base_model = models.resnet18(pretrained=True)  # Load pre-trained ResNet18

        # Modify the first convolutional layer to accept 1-channel input (grayscale)
        self.base_model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)

        # CBAM applied after Conv5_x (512 channels)
        self.cbam = CBAM(512)

        # Replace the final fully connected layer to match the number of classes
        self.base_model.fc = nn.Linear(512, num_classes)

    def forward(self, x):
        # Forward pass through ResNet18's convolutional blocks
        x = self.base_model.conv1(x)
        x = self.base_model.bn1(x)
        x = self.base_model.relu(x)
        x = self.base_model.maxpool(x)

        x = self.base_model.layer1(x)
        x = self.base_model.layer2(x)
        x = self.base_model.layer3(x)
        x = self.base_model.layer4(x)

        # Apply CBAM after Conv5_x
        x = self.cbam(x)

        # Global average pooling and final classification
        x = self.base_model.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.base_model.fc(x)
        return x
```

Model summary:

Figure 4. 20 ResNet18-CBAM Model Summary

```
Total params: 11,237,926
Trainable params: 11,237,926
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 63.37
Params size (MB): 42.87
Estimated Total Size (MB): 106.43
----------------------------------------------------------------
```

3. **Training Process:** The training process includes the initial training and refined training:

1) Initial Training

The initial training uses baseline hyperparameters to assess general behavior. Key details include:

- **Optimizer:** Adam optimizer with a learning rate of 0.001 is used for parameter updates.
- **Learning Rate:** A StepLR scheduler reduced the learning rate by a factor of 0.1 every 5 epochs to ensure smoother convergence.
- **Loss Function:**
    - Class weights is computed using scikit-learn's compute_class_weight function to address class imbalance

- Focal loss with class weights is employed to handle imbalance classes and improve performance on underrepresented classes.

- **Early Stopping** is configured to monitor validation loss and halt training after five consecutive epochs with no improvement.

Training workflow:

Figure 4. 21 ResNet18-CBAM Training Setup

```
# Compute class weights to handle imbalanced classes in the training dataset
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(device)

# Function to handle class imbalance and improve performance on underrepresented classes
class FocalLoss(nn.Module):
    def __init__(self, alpha=None, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma

    def forward(self, inputs, targets):
        ce_loss = F.cross_entropy(inputs, targets, reduction='none', weight=self.alpha)
        p_t = torch.exp(-ce_loss)  # Compute the probabilities of correct class
        focal_loss = (1 - p_t) ** self.gamma * ce_loss
        return focal_loss.mean()


# Configure the optimizer, learning rate scheduler, and loss function
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  # Adam optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
criterion = FocalLoss(alpha=class_weights_tensor, gamma=2)  # focal loss

# Set up early stopping params
early_stopping_patience = 5
best_val_loss = float('inf')  # Best validation loss
early_stopping_counter = 0  # Track consecutive epochs with no improvement
```

2) Refined Training Process

Based on the result of the initial training, several refinement are made:

- **Dropout Regularization:** Dropout layer with a probability of 0.7 to reduce overfitting.
- **Learning Rate Adjustments:** The learning rate is reduced to 5e-5 with cosine annealing scheduler to stabilize training and avoid overshooting
- **Weight Decay:** The weight decay of 1e-2 is added to penalize overly complex models.
- **Loss Function Adjustment:** Switched to cross-entropy loss with label smoothing 0.1 for better generalization and handling of noisy labels.

Refine training workflow:

Figure 4. 22 Refined Training Setup

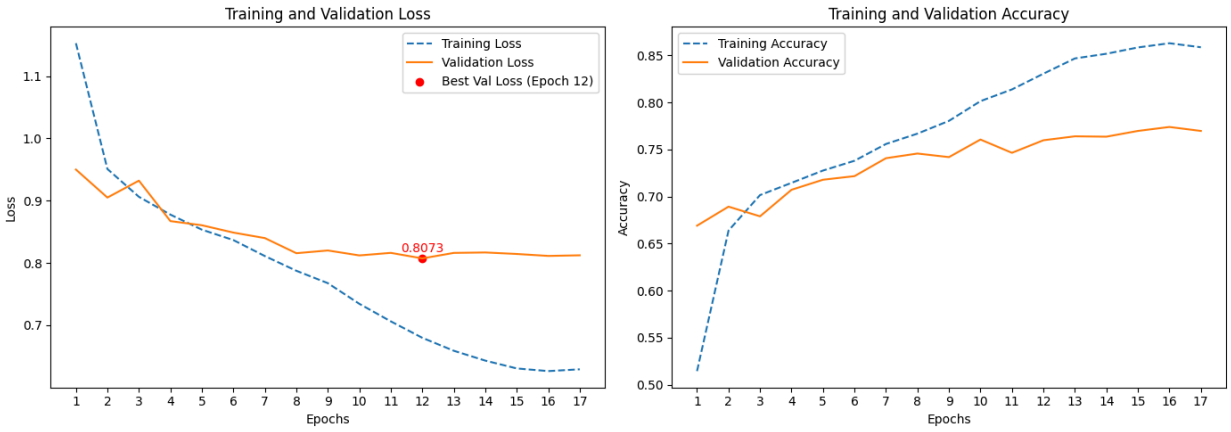
```
# Initialize optimizer, scheduler and criterion
optimizer = optim.Adam(model.parameters(), lr=5e-5, weight_decay=1e-2)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=15)
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
```

3) Validation Process:

Throughout training, validation loss and accuracy are closely monitored after each epoch to guide hyperparameter tuning and prevent overfitting. Multiple hyperparameter combinations are tested, with the final configuration: learning rate (5e-5), dropout (0.7), weight decay (1e-2), and label smoothing (0.1), selected based on its stable validation accuracy around 78% while maintaining low validation loss.

The training and validation loss curves show clear and consistent improvement in both metrics during the initial training phases. Early stopping is triggered at epoch 17 when validation loss stops improving for several consecutive epochs, ensuring the final model is selected at the optimal between training performance and generalization.

Figure 4. 23 ResNet18-CBAM training and validation loss/accuracy



Overall, the refined training process achieves more stable and reliable generalization compared to the initial training runs.

4.3 Music Streaming Service API Integration

The integration contains three main components:

1. **Webcam Setup and Preprocessing**
   To capture a real-time facial image, the function`activate_camera` is developed using JavaScript embedded within Python. This function, adapted from an open-source implementation on GitHub [34], performs the steps involving activating the webcam to display a live video feed, allows the user to capture an image by clicking a "capture" button, then saves the captured image as a JPEG file for further processing.

   Figure 4. 23 Code for webcam setup

```
# Function to activate the webcam and display the video feed
def activate_camera():
    js = Javascript('''
        async function activateCamera() {
            const div = document.createElement('div');
            const video = document.createElement('video');
            video.style.display = 'block';
            div.appendChild(video);
            document.body.appendChild(div);

            const stream = await navigator.mediaDevices.getUserMedia({video: true});
            video.srcObject = stream;
            await video.play();

            // Add a button for capturing the photo
            const capture = document.createElement('button');
            capture.textContent = 'Capture';
            div.appendChild(capture);

            // Wait for the Capture button to be clicked
            await new Promise((resolve) => capture.onclick = resolve);

            const canvas = document.createElement('canvas');
            canvas.width = video.videoWidth;
            canvas.height = video.videoHeight;
            canvas.getContext('2d').drawImage(video, 0, 0);
            stream.getTracks().forEach(track => track.stop());
            div.remove();

            return canvas.toDataURL('image/jpeg', 0.8);
        }
    ''')
    return js
```

   The captured images are preprocessed via the *`process_captured_image`* function which involves focusing the face using the Haar Cascade Classifier, crop, convert to grayscale, resize to 224×224, normalize the pixel values and convert the image into Pytorch tensor to make it able to fit as an input to the ResNet18-CBAM model.

Figure 4. 24 Code for preprocessing captured image

```python
# Function to process the captured image for facial recognition tasks
# The process includes detect face, convert to grayscale, and resize it.
def process_captured_image(image_path, target_size=(224, 224)):
    # Load Haar Cascade for face detection
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

    # Read the image
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Input file {image_path} not found.")

    # Convert the image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(50, 50))
    if len(faces) == 0:
        raise ValueError("No face detected in the image.")

    # Crop the first detected face
    x, y, w, h = faces[0]
    face = gray[y:y+h, x:x+w]

    # Resize the face to the target size
    resized_face = cv2.resize(face, target_size)

    # Normalize pixel values to [0, 1]
    normalized_face = resized_face / 255.0

    # Add channel and batch dimensions for PyTorch (1, 1, 224, 224)
    face_tensor = torch.tensor(normalized_face, dtype=torch.float32).unsqueeze(0).unsqueeze(0)

    # Move the tensor to the specified device
    face_tensor = face_tensor.to(device)

    return face_tensor
```

2. **Emotion Prediction**

   The preprocessed image is passed to the `predict_emotion` to predict the user's emotional state. The model outputs a class index, which is mapped to one to the predefined emotion classes: Angry, Happy, Neutral, Sad.

   Figure 4. 25 Code for predict emotion

```python
# Function to predict emotion
def predict_emotion(image_path):
    # Preprocess the image
    img_tensor = process_captured_image(image_path)

    # Make the prediction
    with torch.no_grad():
        output = model(img_tensor) # Forward pass
        predicted_class = torch.argmax(output, dim=1).item() # Get the predicted class index

    # Map the predicted class index to the corresponding emotion label
    emotion = class_names[predicted_class]

    return emotion
```

3. **Spotify API Integration**

   To integrate Spotify API with the model, the Spotify authentication setup and fetching track from Spotify is referenced from GitHub [35] https://github.com/SyedsPortfolio/Spotify-Recommendation-System/blob/main/Spotify_Recommendation_System.ipynb.

1) **Spotify Authentication Setup**:
The workflow implements Spotify's OAuth 2.0 process, where the application requests user authorization, retrieves an authorization code, exchanges it for access and refresh tokens, and uses the access token to make authenticated API requests. The refresh token ensures continued secure access by renewing expired tokens. The following diagram and code show how it works:

Figure 4. 26 OAuth authorization flow for accessing Spotify Web API [36]

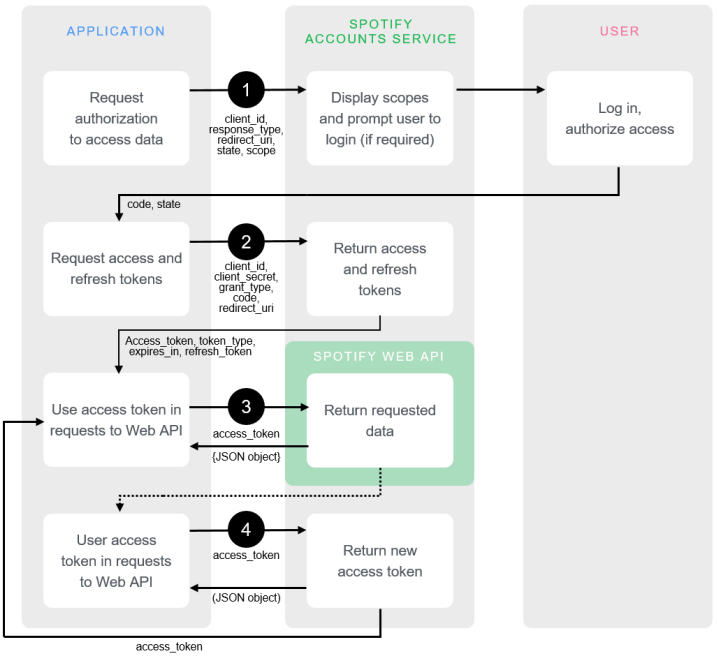


Figure 4. 27 Code for Spotify authentication setup

```
# Set up SpotifyOAuth
auth_manager = SpotifyOAuth(
    client_id=SPOTIPY_CLIENT_ID,
    client_secret=SPOTIPY_CLIENT_SECRET,
    redirect_uri=SPOTIPY_REDIRECT_URI,
    scope=scope,
    open_browser=False
)

# Generate the authentication URL
auth_url = auth_manager.get_authorize_url()
print("Click the link below to authenticate:")
print(auth_url)

# Paste the redirected URL after logging in
redirected_url = input("Paste the redirected URL here: ")

# Extract the authorization code
auth_code = auth_manager.parse_response_code(redirected_url)

# Fetch the access token using the authorization code
token_info = auth_manager.get_access_token(code=auth_code)

# Use the token to create a Spotify client
sp = spotipy.Spotify(auth=token_info['access_token'])

# Test the connection
current_user = sp.current_user()
print(f"Spotify authenticated successfully! Logged in as {current_user['display_name']}")
```

2. Fetch Tracks from Spotify: To fetch emotion-aligned tracks from Spotify, the *`fetch_tracks_from_playlist`* function is created to search Spotify playlist based on the predefined emotion keywords and retrieved track URIs from the top matching playlist. These keywords correspond to culturally intuitive associations with moods, for example

- "Happy" might map to playlists with terms like "Joy" or "Pop".
- "Sad" could be associated with "Tear" or "Depression".

Figure 4. 28 Code for fetch tacks from Spotify

```
# Function to search playlists by emotion keywords and fetches tracks from the playlist.
def fetch_tracks_from_playlists(emotion, limit=10):

  # Predefined emotion-aligned playlist keywords
  # these keywords are culturally intuitive, people commonly associate specifci terms with mood
  emotion_keywords = {
      "Happy": ["Happy", "Good", "Joy", "Pop"],
      "Sad": ["Sad", "Cry", "Tear", "Depression"],
      "Angry": ["Angry", "Annoyed", "Rage", "Agressive"],
      "Neutral": ["Calm", "Chilling", "Relaxed", "Ambiance"]
  }

  # Check if emotion keyword can fetch the related playlist
  keywords = emotion_keywords.get(emotion, [])
  if not keywords:
      print(f"No keywords found for emotion: {emotion}")
      return []

  # Define a list to store fetched tracks
  tracks = []

  # Search for playlists by emotion keywords
  for keyword in keywords:
      try:
          print(f"Searching playlists with keyword: {keyword}")
          search_results = sp.search(q=keyword, type="playlist", limit=1)  # Limit to 1 playlist per keyword

          # Extract the first playlist's tracks
          if search_results['playlists']['items']:
              playlist = search_results['playlists']['items'][0]
              playlist_id = playlist['id']
              print(f"Found playlist: {playlist['name']} (ID: {playlist_id})")

              # Get tracks info from the playlist
              playlist_tracks = sp.playlist_tracks(playlist_id, limit=limit)
              for item in playlist_tracks['items']:
                  track = item['track']
                  print(f"Track found: {track['name']} by {track['artists'][0]['name']}")
                  tracks.append(track['uri'])
          else:
              print(f"No playlists found with keyword: {keyword}")

      except Exception as e:
          print(f"Error searching playlists with keyword '{keyword}': {e}")

  # Return a list of unique tracks (Remove duplicates)
  unique_tracks = list(set(tracks))
  print(f"Fetched {len(unique_tracks)} unique tracks for emotion: {emotion}")
  return unique_tracks
```

3. Create Emotion-Based Playlist: The *`create_emotion_playlist`* function creates a private playlist based on the predicted emotion and adds the retrieved tracks. When a playlist is successfully created, a playlist link is generated as an output, enabling direct access to the personalized playlist in Spotify.

Figure 4. 29 Code creating emotion-based playlist

```
# Function to create a new playlist for a specific emotion and adds tracks to it.
def create_emotion_playlist(user_id, emotion, track_uris):
  # Create a playlist name based on the emotion
  playlist_name = f"{emotion} Vibes Playlist"
  try:
    # Create a new playlist
    playlist = sp.user_playlist_create(user=user_id, name=playlist_name, public=False)
    playlist_id = playlist['id']
    # Adds the given tracks 'track_uris' into the playlist
    sp.playlist_add_items(playlist_id, track_uris)
    # Return the playlist link to access
    playlist_link = playlist['external_urls']['spotify']
    print(f"`{playlist_name}` is created successfully!!")
    print(f"Access it here: {playlist_link}")
    return playlist_link
  except Exception as e:
      print(f"Error creating playlist: {e}")
      return None
```

```
# Fetch tracks for predicted emotion
track_uris = fetch_tracks_from_playlists(predicted_emotion)
print(f"Fetched track URIs: {track_uris}")

# Generate a playlist based on the predicted emotion with the fetched tracks
if track_uris:
  user_id = sp.current_user()["id"]
  playlist_link = create_emotion_playlist(user_id, predicted_emotion, track_uris)
```

CHAPTER 5: EVALUATION

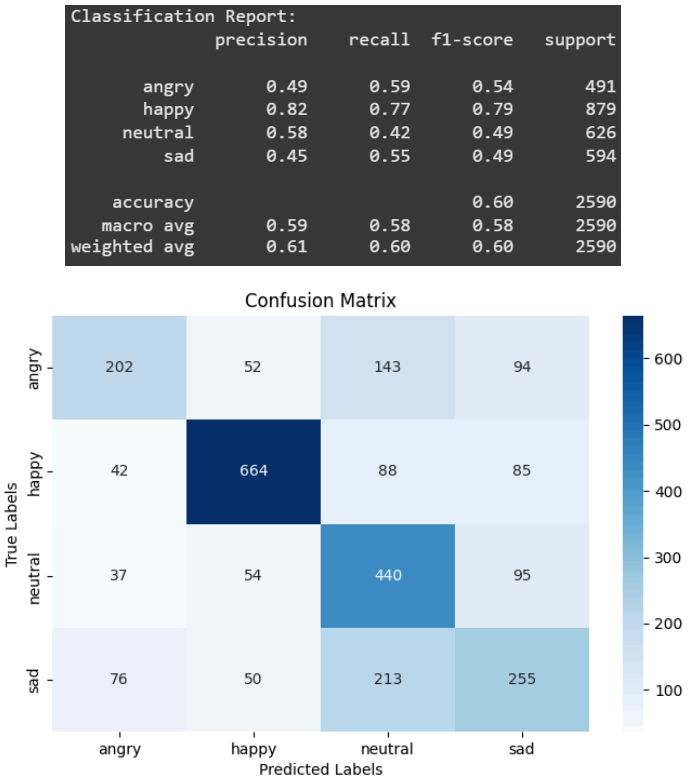
5.1 Model Performance Evaluation

This section evaluates the classification performance of the baseline CNN model and the ResNet18-CBAM model, comparing their test accuracy, precision, recall, F1-score, and confusion matrices. To ensure a comprehensive evaluation, the results of the proposed ResNet18-CBAM are also compared with prior FER-2013 reviewed studies to assess its relative effectiveness..

5.1.1 Baseline CNN Model

- **Test Accuracy: 60%**
- **Evaluation Metrics:**

Figure 5. 1 Evaluation performance of baseline CNN



- **Performance Analysis of Baseline CNN:**
  The baseline CNN model demonstrates varied performance across different emotions, with significant challenges in distinguishing emotions with overlapping features.
  - **Angry class:** Achieves moderate recall (0.59) but has low precision (0.49), showing confusion with neutral and sad expressions due to overlapping serious facial expressions.

- **Happy class:** Performs the best, with high precision (0.82) and recall (0.77), effectively identifying distinctive features like smiles, though minor classifications occurred with neutral (88) and sad (85) classes.
- **Neutral class:** Has the lowest recall (0.42) with many neutral expressions misclassified as sad (213) or angry (143), reflecting difficulty in distinguishing neutral from negative emotions with subtle variations.
- **Sad class:** Shows moderate recall (0.55) and low precision (0.45), with heavy confusion between neutral (213) and angry (76) expressions.

Overall, the model struggled with distinguishing negative and neutral motions, emphasizing the need for more robust features extraction and class separations.

5.1.2 ResNet18-CBAM Model

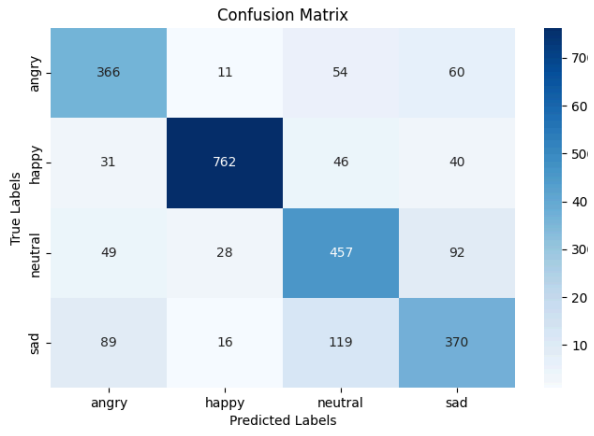
The ResNet18-CBAM model is trained in two phases:

1. **Initial Training -** Before hyperparameter tuning.
2. **Refined Training -** After fine-tuning for improved performance.

<u>Initial Training Performance</u>
Test Accuracy: 75%

Figure 5. 2 Evaluation initial performance of ResNet18-CBAM

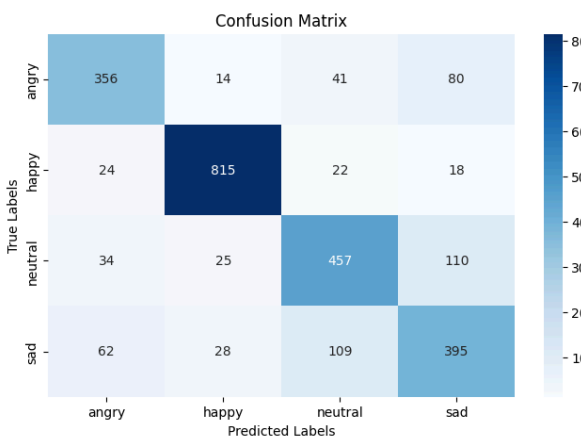
Classification Report:	precision	recall	f1-score	support
angry	0.68	0.75	0.71	491
happy	0.93	0.87	0.90	879
neutral	0.68	0.73	0.70	626
sad	0.66	0.62	0.64	594
accuracy			0.75	2590
macro avg	0.74	0.74	0.74	2590
weighted avg	0.76	0.75	0.76	2590



<u>Refined Training Performance</u>
Test Accuracy: 78%

Figure 5. 3 Evaluation refined performance of ResNet18-CBAM

Classification Report:	precision	recall	f1-score	support
angry	0.75	0.73	0.74	491
happy	0.92	0.93	0.93	879
neutral	0.73	0.73	0.73	626
sad	0.66	0.66	0.66	594
accuracy			0.78	2590
macro avg	0.76	0.76	0.76	2590
weighted avg	0.78	0.78	0.78	2590



- **Performance Improvement from Refinement**
  The refined training process resulted in an increase in test accuracy from 75% to 78% due to improvements discussed in Chapter 4 Implementation, including dropout regularization, learning rate adjustments, label smoothing, and loss function changes, which collectively led to better generalization and stability.

  The ResNet18-CBAM model demonstrates overall improvements after fine-tuning, with better generalization and class balance.
  - **Angry class:** Shows a significant precision improvement from 0.68 to 0.75, while recall decreases slightly from 0.75 to 0.73. This results in stable F1-scores, with fewer misclassifications. The model correctly predicts 356 samples as angry, with reduced confusion compared to initial training: 41 misclassified as neutral and 80 as sad.
  - **Happy class:** Achieves consistent precision at 0.93, while recall improves from 0.87 to 0.93. It correctly classifies 815 samples as happy, with minimal misclassifications, including 24 as angry and 22 as neutral.
  - **Neutral class:** Benefits from improved precision (0.68 to 0.73), while recall remains steady at 0.73. The model correctly identifies 457 neutral samples, reducing confusion with 110 misclassified as sad and 34 as angry.
  - **Sad class:** Precision remains stable at 0.66, while recall increases from 0.62 to 0.66, improving the F1-score from 0.64 to 0.66. The model correctly predicts 385 sad samples, though confusion persists, with 109 misclassified as neutral and 62 as angry.

Performance Comparison with Baseline CNN Model
The following table summarizes the performance improvements of ResNet18-CBAM over the baseline CNN model:

Table 5. 1 Comparative between baseline and ResNet18-CBAM

Class	Metric	Baseline CNN	ResNet18-CBAM (refined)	Improvement
	Overall Accuracy	60%	78%	+18%
Angry	Precision	0.49	0.75	+26%
	Recall	0.59	0.73	+14%
	F1-Score	0.54	0.74	+20%
Happy	Precision	0.82	0.92	+10%
	Recall	0.77	0.93	+16%
	F1-Score	0.79	0.93	+14%

Neutral	Precision	0.58	0.73	+10%
	Recall	0.42	0.73	+31%
	F1-Score	0.49	0.73	+26%
Sad	Precision	0.45	0.66	+21%
	Recall	0.55	0.66	+10%
	F1-Score	0.49	0.66	+17%

- **Performance Analysis:**
  While the baseline CNN provides a strong benchmark, the ResNet18-CBAM model significantly improves performance, achieving 78% overall accuracy.
  - **Angry Class:** Significantly improvements in recall (+14%) and precision (+26%), indicating better detection of angry emotions.
  - **Happy Class:** Achieved the best results, with precision at 92% and recall at 93%.
  - **Neutral Class:** Largest improvement in recall (+31%), showcasing better detection of subtle features.
  - **Sad Class:** Moderate improvement, but still challenges in classified with neutral and angry classes.

  The confusion matrix highlights reduced misclassifications across all classes compared to the baseline CNN, Most notably, there is less confusion between neutral and sad and between angry and neutral, leading to clearer distinctions in predictions. These refinements demonstrate the effectiveness of fine-tuning the ResNet18-CBAM model, particularly for challenging negative emotions.

Performance Comparison with Prior FER-2013 Reviewed Studies

To contextualize the performance of ResNet18-CBAM, its result is compared with prior reviewed studies on FER-2013.

Table 5. 2:Performance comparison with prior FER-2013 reviewed studies

Model	**Accuracy**
CNN [12]	58.9%
Lightweight CNN + MTCNN [12]	67%
Hybrid DCNN + Haar Cascade [13]	70.04%
ResNet-18 (Pretrained) [14]	71.89%
ResNet18_CBAM (Proposed Model)	**78%**

ResNet18-CBAM outperforms previously reviewed FER-2013 models, achieving an accuracy of 78%, while prior models ranged between 58.9% and 71.89%. Notably, compared to a standard ResNet-18 model (71.89%), the integration of CBAM increased a +6.11% improvement, highlighting the effectiveness of attention mechanisms in enhancing feature extraction and classification performance. These results suggest that CBAM's spatial and channel attention mechanisms contribute to better generalization and more precise facial expression recognition

5.2 Emotion-Aligned Music Playlist Mapping Evaluation

This section evaluates the feasibility of integrating the ResNet18-CBAM emotion recognition model with the Spotify API to map detected emotions to predefined playlists.

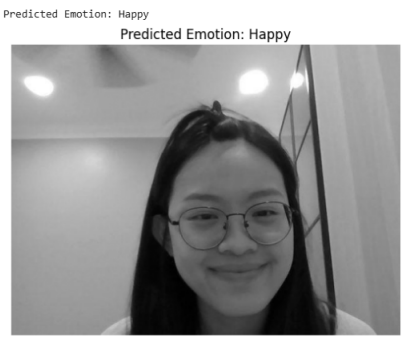
5.2.1 Webcam Real-Time Capture and Emotion Prediction

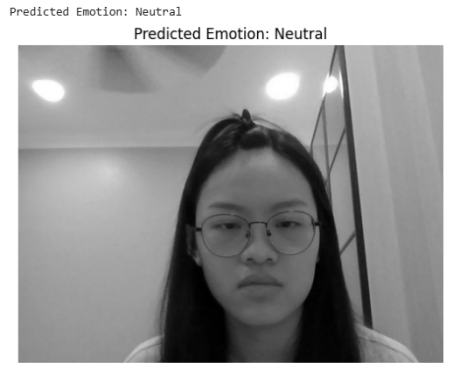
The webcam captures a real-time facial image, which is processed and fed into the trained ResNet18-CBAM model to predict the corresponding emotion. The predicted emotion is displayed as a title on the captured image, allowing users to see the model's interpretation.

Emotion Recognition Testing Results

The table below presents sample test cases, comparing the expected emotion, the model's actual prediction, and whether the output was correct.

Table 5. 2:Emotion Recognition Testing Results

Captured Image	**Expected Result**	**Actual Result**	**Remark (Pass/ Failed)**
	Happy	Happy	Pass

	Angry	Angry	Pass
	Sad	Sad	Pass
	Neutral	Neutral	Pass
	Angry	Neutral	Failed

Based on the observations, the model correctly classified emotions in most cases, indicating an overall well performance. However, misclassifications still occur due to the following factors:

- Minor variations in expressions can make classification difficult. For example, a slight smirk may be misclassified between neutral and happy.
- Poor lighting or noise in the image can affect facial feature extraction. For example, dim lighting may cause the model to miss fine-grained expressions.
- Some emotions, such as angry and neutral, have similar facial expressions, leading to misclassifications. For example, sad and neutral share similar tense expressions which may cause confusion.

This misclassification highlights the need for further improvements, such as enhancing feature extraction through additional data augmentation.

5.2.2 Spotify API Integration for Emotion-Based Playlist

Once the user's facial emotion is recognized, the system integrates with Spotify API to retrieve and generate an emotion-aligned playlist. The system relies on a set of predefined mappings between recognized emotions and suitable music genres, enabling it to query Spotify for tracks that match the detected emotional state.

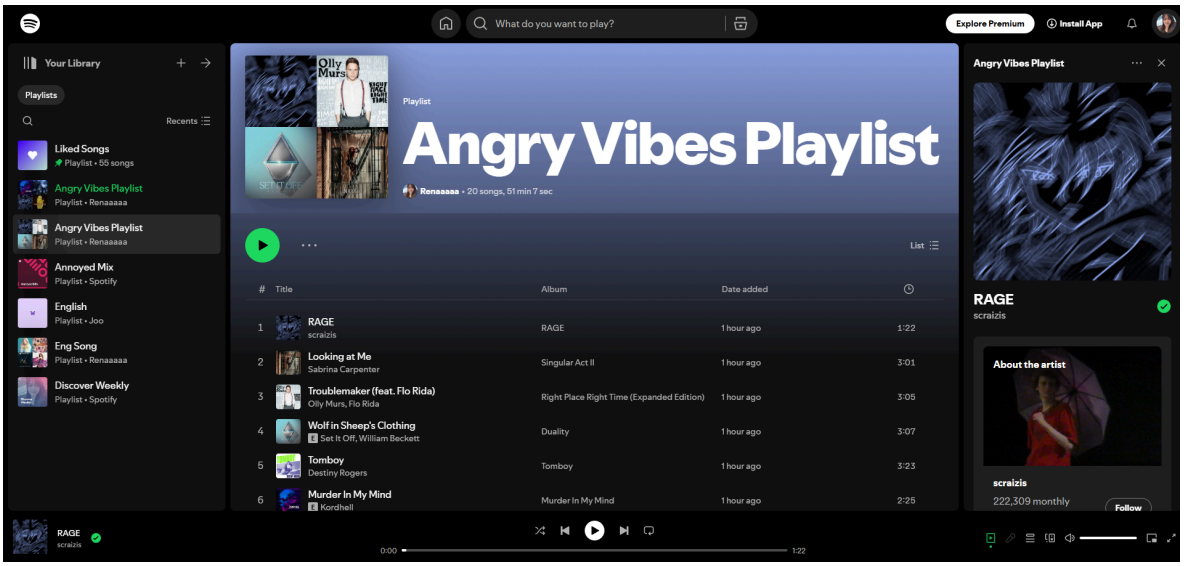
For example, when an angry emotion is detected, the system retrieves tracks aligned with this emotion, as shown in the console output below.
The below console output shows the tracks retrieved for the Angry emotion:

Figure 5. 4 Console output showing playlist creation process for the Angry emotion using Spotify API.

```
Searching playlists with keyword: Angry
Found playlist: Gacha Life Songs 2017-2020 (ID: 27awmW5TzyLWKejZ4VT8ob)
Track found: Acapella by Karmin
Track found: Guys Dont Like Me by It Boys!
Track found: Tomboy by Destiny Rogers
Track found: Looking at Me by Sabrina Carpenter
Track found: She's Crazy but She's Mine by Alex Sparrow
Track found: Just My Type by The Vamps
Track found: Say My Name by David Guetta
Track found: Troublemaker (feat. Flo Rida) by Olly Murs
Track found: How to Be a Heartbreaker by MARINA
Track found: Wolf in Sheep's Clothing by Set It Off
Searching playlists with keyword: Annoyed
Error searching playlists with keyword 'Annoyed': 'NoneType' object is not subscriptable
Searching playlists with keyword: Rage
Error searching playlists with keyword 'Rage': 'NoneType' object is not subscriptable
Searching playlists with keyword: Agressive
Found playlist: TIKTOK PHONK SONGS 2024 / 2025 HITS 💀 (ID: 7jucyRFk66ag3ALFZZ8pAt)
Track found: Murder In My Mind by Kordhell
Track found: FUNK DO BOUNCE (Slowed) by Ariis
Track found: SLAY! by Eternxlkz
Track found: LOVELY BASTARDS by ZWE1HVNDXR
Track found: 9mm by Memphis Cult
Track found: Bero 00 by Anar
Track found: I LOVE YOU HOE (w/ 9lives) by Odetari
Track found: READY TO ROLL by MoonDeity
Track found: CUTE DEPRESSED by Dyan Dxddy
Track found: RAGE by scraizis
Fetched 20 unique tracks for emotion: Angry
Fetched track URIs: ['spotify:track:52OWRufMrpxRcv3Sy97A42', 'spotify:track:59tskctgqUmjCWAwhzYAFm', 'spotify:track:6s8nHXTJVqFjXE4yVZPDHR', 'spotify:track:55Fpeuuc2sbQiy74eA1gTt', 'spotify:track:5Ti3fqGTfjUs
`Angry Vibes Playlist` is created successfully!!
Access it here: https://open.spotify.com/playlist/7fdIygW9ChtjRYxe2nEMI1
```

The generated playlist is then automatically created on the user's Spotify account and is accessible through a direct link, allowing users to listen to the recommended tracks on Spotify.

Figure 5. 5 Spotify-generated playlist



Overall, the successful integration of the ResNet18-CBAM emotion recognition model with the Spotify API validates the feasibility of using facial emotion recognition to trigger emotion-driven playlist generation.

5.3 Hypotheses Validation

Hypothesis 1 Validation

The hypothesis “Adding CBAM to ResNet18 will improve facial emotion recognition performance on FER-2013 compared to a baseline CNN” is validated through both quantitative performance metrics and the validation process.
During training, the ResNet18-CBAM model demonstrates more stable validation accuracy and lower validation loss compared to the baseline CNN, indicating better generalization. The final test results further confirms this trend, with ResNet18-CBAM achieving approximately 78% accuracy, representing an 18% improvement over the baseline. The confusion matrix analysis also shows that the proposed model is more effective at distinguishing subtle facial expressions, particularly between Sad and Neutral, which are frequently misclassified by the baseline model. These findings collectively support the hypothesis that incorporating CBAM into ResNet18 enhances facial emotion recognition performance.

Hypothesis 2 Validation

The hypothesis “A higher accuracy FER model (ResNet18 + CBAM) will enable more relevant emotion-aligned music playlists” is validated through real-time emotion recognition tests using webcam input, where the predicted emotions are used to map predefined playlist

recommendations through the Spotify API. During testing, the ResNet18-CBAM model demonstrates reliable emotion classification, resulting in mapped playlists that are generally aligned with the detected emotions. However, as highlighted in section 5.2.1, occasional misclassification occurs, which could affect the accuracy of playlist recommendations. These findings support the hypothesis while also reinforcing the need for further refinement to improve the model robustness.

5.4 Limitations and Future Work

Despite the promising results of this project, several limitations remain.

One major limitation is the FER-2013 dataset, which contains low-resolution images and class imbalances, impacting model generalization. This issue leads to misclassification errors, particularly in similar expressions such as neutral and sad. Future work can address this by training on larger and more diverse datasets, such as the AffectNet which have been analyzed in the data analysis section of chapter 2. This expansion would improve emotion variability and classification accuracy, enabling the model to better distinguish between visually similar emotions.

Another key limitation is that the model runs in a Google Colab environment and is not deployed as a standalone application, limiting real-time deployment and large-scale user evaluations for assessing playlist mapping effectiveness. Since the current evaluation does not include user studies, future work should explore deploying the system as a mobile or web-based application, allowing for direct user interaction and real-world validation. This would provide quantitative insights into user experience and recommendation relevance accuracy, strengthening the system’s practical application.

Furthermore, user studies and real-world implementation would allow for an adaptive approach to emotion-based music playlist mapping. The current system relies on a static emotion-to-music mapping, which may not always align with individual user preferences. To enhance personalization, reinforcement learning could be integrated, allowing the model to learn from user feedback and refine its playlists recommendations over time. By dynamically adjusting emotion-to-music associations based on listening behavior, the system could become more context-aware and user-centric.

CHAPTER 6: CONCLUSION

This project focuses on developing a facial emotion recognition (FER) model, where ResNet18 is proposed and enhanced with CBAM to improve classification accuracy. To validate the feasibility of emotion-aligned playlist mapping, the trained FER model integrates with the Spotify API, enabling detected emotions to map to personalized music playlists. The evaluation demonstrates that CBAM-enhanced ResNet18 improves FER accuracy, achieving 78% accuracy, compared to 60% from the baseline CNN. Real-time emotion detection via webcam captures the user's facial expressions, effectively processing them for emotion-driven music retrieval.

This study highlights the potential of AI-driven emotion recognition models for real-world applications, including context-aware media recommendations. While this system does not generate fully personalized music recommendations, emotion-based playlist mapping remains a form of recommendation, as it suggests playlists based on detected emotions. Future developments can further enhance adaptability, enabling more emotionally responsive AI systems.

(9375 words)

References

[1] Nawaz Ahmad and Afsheen Rana. 2015. Impact of music on Mood: Empirical investigation. Retrieved from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2696883

[2] Valorie N Salimpoor, Mitchel Benovoy, Kevin Larcher, Alain Dagher, and Robert J Zatorre. 2011. Anatomically distinct dopamine release during anticipation and experience of peak emotion to music. Nature Neuroscience 14, 2 (January 2011), 257–262. https://doi.org/10.1038/nn.2726

[3] Ulrich Dolata. 2020. The digital transformation of the music industry. The second decade: From download to streaming. Retrieved from https://www.econstor.eu/handle/10419/225509

[4] Laura Childs-Young. 2023. IFPI's global study finds we're listening to more music in more ways than ever - IFPI. IFPI. Retrieved from https://www.ifpi.org/ifpis-global-study-finds-were-listening-to-more-music-in-more-ways-than-ever

[5] Yiwen Ding and Chang Liu. 2015. Exploring drawbacks in music recommender systems : the Spotify case. DIVA. Retrieved from https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A896794&dswid=6681

[6] Markus Schedl, Christine Bauer, Wolfgang Reisinger, Dominik Kowald, and Elisabeth Lex. 2021. Listener Modeling and Context-Aware music recommendation based on country archetypes. Frontiers in Artificial Intelligence 3, (February 2021). https://doi.org/10.3389/frai.2020.508725

[7] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. International Journal of Multimedia Information Retrieval 7, 2 (April 2018), 95–116. https://doi.org/10.1007/s13735-018-0154-2

[8] Na Jiang, Xiaohui Liu, Hefu Liu, Eric Tze Kuan Lim, Chee-Wee Tan, and Jibao Gu. 2022. Beyond AI-powered context-aware services: the role of human–AI collaboration. Industrial

Management & Data Systems 123, 11 (December 2022), 2771–2802. https://doi.org/10.1108/imds-03-2022-0152

[9] Manas Sambare. 2020. FER-2013. Kaggle. Retrieved from https://www.kaggle.com/datasets/msambare/fer2013

[10] Adzira Husain. 2020. Murdoch University. Framework for Visualising Music Mood Using Visual Texture (2020). Retrieved from https://researchportal.murdoch.edu.au/esploro/outputs/doctoral/Framework-for-visualising-music-mood-using/991005543853807891#file-0

[11] Keunwook Kim. 2021. Movement and Emotions. Retrieved from https://kimkeunwook.com/Movement-and-Emotions

[12] Ning Zhou, Renyu Liang, and Wenqian Shi. 2020. A lightweight convolutional neural network for Real-Time facial expression detection. IEEE Access 9, (December 2020), 5573–5584. https://doi.org/10.1109/access.2020.3046715

[13] Ozioma Collins Oguine, Kanyifeechukwu Jane Oguine, Hashim Ibrahim Bisallah, and Daniel Ofuani. 2022. Hybrid Facial Expression Recognition (FER2013) Model for Real-Time Emotion Classification and Prediction. arXiv.org. Retrieved from https://arxiv.org/abs/2206.09509

[14] Xinyi Shen, Xinmin Xu, and Yao Zhuang. 2021. Facial emotion Recognition based on Sobel-Resnet. 2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC) 7, (October 2021), 484–488. https://doi.org/10.1109/itnec52019.2021.9587235

[15] Naval Kishore Mehta, Shyam Sunder Prasad, Sumeet Saurav, Ravi Saini, and Sanjay Singh. 2022. Three-dimensional DenseNet self-attention neural network for automatic detection of student's engagement. Applied Intelligence 52, 12 (March 2022), 13803–13823. https://doi.org/10.1007/s10489-022-03200-4

[16] Yong Li, Jiabei Zeng, Shiguang Shan, and Xilin Chen. 2018. Occlusion aware facial expression recognition using CNN with attention mechanism. IEEE Transactions on Image Processing 28, 5 (December 2018), 2439–2450. https://doi.org/10.1109/tip.2018.2886767

[17] Byoung Ko. 2018. A brief review of facial emotion recognition based on visual information. Sensors 18, 2 (January 2018), 401. https://doi.org/10.3390/s18020401

[18] Ali Mollahosseini, Behzad Hasani, and Mohammad H. Mahoor. 2017. AffectNet: a database for facial expression, valence, and arousal computing in the wild. IEEE Transactions on Affective Computing 10, 1 (August 2017), 18–31. https://doi.org/10.1109/taffc.2017.2740923

[19] Patrick Lucey, Jeffrey F. Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. 2010. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (June 2010). https://doi.org/10.1109/cvprw.2010.5543262

[20] Publications Office of the European Union. 2021. TechDispatch : facial emotion recognition. Issue 1, 2021. Publications Office of the EU. Retrieved from https://op.europa.eu/en/publication-detail/-/publication/709bd47b-be94-11eb-a925-01aa75ed71a1/language-en

[21] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. 2024. A review of convolutional neural networks in computer vision. Artificial Intelligence Review 57, 4 (March 2024). https://doi.org/10.1007/s10462-024-10721-6

[22] M. C. Leong, D. Prasad, Yong Tsui Lee, and Feng Lin. 2019. Semi-CNN architecture for effective Spatio-Temporal Learning in Action recognition. Applied Science (2019). Retrieved from https://www.semanticscholar.org/paper/Semi-CNN-Architecture-for-Effective-Spatio-Temporal-Leong-Prasad/29248ee8f8c7032e6d122390f02973a3e76ac6e4

[23] Aqeel Anwar. 2022. Difference between AlexNet, VGGNet, ResNet, and Inception. Medium. Retrieved from

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

[24] Zhong Hong. 2023. Attention mechanisms in Deep Learning: Enhancing model performance. Medium. Retrieved from https://medium.com/@zhonghong9998/attention-mechanisms-in-deep-learning-enhancing-model-performance-32a91006092a

[25] Yesha Shastri. 2024. Attention Mechanism in LLMs: An Intuitive Explanation. Retrieved from https://www.datacamp.com/blog/attention-mechanism-in-llms-intuition

[26] Andrzej Miskow and Abdulrahman Altahhan. 2024. Emotion Recognition with Facial Attention and Objective Activation Functions. arXiv.org. Retrieved from https://arxiv.org/abs/2410.17740

[27] Poorvi Taunk, Vani Jayasri, Padma Priya J, and Suresh Kumar. 2020. Face Detection using Viola Jones with Haar Cascade. Test Engineering and Management 83, (April 2020), 19146–19149. Retrieved from https://www.researchgate.net/publication/342122717_Face_Detection_using_Viola_Jones_with_Haar_Cascade#fullTextFileContent

[28] Hoang N. Tran, Phuc H. Phan, Khang H. Nguyen, Huy K. Hua, Anh Q. Nguyen, Huynh N.V. Nguyen, and Nghi V. Nguyen. 2024. Augmentation-Enhanced deep learning for face detection and emotion recognition in elderly care robots. Research Square (Research Square) (February 2024). https://doi.org/10.21203/rs.3.rs-3924396/v1

[29] Spotify. Web API | Spotify for Developers. Retrieved from https://developer.spotify.com/documentation/web-api

[30] Apple. Apple Music API | Apple Developer Documentation. *Apple Developer Documentation*. Retrieved from https://developer.apple.com/documentation/applemusicapi/

[31] TIDAL. Overview | TIDAL Developer Portal. *TIDAL Developer Portal*. Retrieved from https://developer.tidal.com/documentation/api-sdk/api-sdk-overview

[32] Farheen Ramzan, Muhammad Usman Ghani Khan, Asim Rehmat, Sajid Iqbal, Tanzila Saba, Amjad Rehman, and Zahid Mehmood. 2019. A deep learning approach for automated diagnosis and Multi-Class classification of Alzheimer's disease stages using Resting-State FMRI and residual neural networks. *Journal of Medical Systems* 44, 2 (December 2019). https://doi.org/10.1007/s10916-019-1475-2

[33] Pendar Alirezazadeh, Michael Schirrmann, and Frieder Stolzenburg. 2022. Improving Deep Learning-based Plant Disease Classification with Attention Mechanism. *Gesunde Pflanzen* 75, 1 (December 2022), 49–59. https://doi.org/10.1007/s10343-022-00796-y

[34] theAIGuysCode. 2020. colab-webcam/colab_webcam.ipynb at main · theAIGuysCode/colab-webcam. *GitHub*. Retrieved from https://github.com/theAIGuysCode/colab-webcam/blob/main/colab_webcam.ipynb

[35] SyedsPortfolio. 2023. Spotify-Recommendation-System/Spotify_Recommendation_System.ipynb at main · SyedsPortfolio/Spotify-Recommendation-System. *GitHub*. Retrieved from https://github.com/SyedsPortfolio/Spotify-Recommendation-System/blob/main/Spotify_Recommendation_System.ipynb

[36] Spotify. Authorization Code Flow. *Spotify for Developers*. Retrieved from https://developer.spotify.com/documentation/web-api/tutorials/code-flow

[37] De Mel. 2023. *Survey of Evaluation Metrics in Facial Recognition Systems*. https://doi.org/10.13140/RG.2.2.10974.20805

[38] Roshan Nayak. 2022. Focal Loss : A better alternative for Cross-Entropy. *Towards Data Science*. Retrieved from https://towardsdatascience.com/focal-loss-a-better-alternative-for-cross-entropy-1d073d92d075/

Appendix A: Code Repository (Google Colab Notebook)

Accessible Link:
https://colab.research.google.com/drive/1w5DGItZnaDf-Qfbf7506j9OLQmJvIDil?usp=sharing

This project is implemented on Google Colab notebook.

Notebook Structure:

1. **Dataset Preparation**
   - Retrieving and preprocessing the FER-2013 dataset.
   - Data splitting, and visualization.

2. **Model Building**

   a) Baseline Model: CNN Model
   - Defining a basic CNN architecture for facial emotion recognition.
   - Training and validating the baseline model.
   - Evaluating accuracy, loss, precision, recall, F1-score.

   b) Proposed Model: ResNet18 + CBAM
   - Implementing an enhanced deep learning model using ResNet18 with CBAM (Convolutional Block Attention Module).
   - Training and fine-tuning the model for improved performance.
   - Comparing evaluation results with the baseline CNN model.

3. **Music Streaming Service API Integration**
   - Webcam Capturing: Real-time facial detection.
   - Facial Emotion Recognition: Classifying emotions from facial expressions.
   - Spotify API Integration: Mapping emotions (Happy, Sad, Angry, Neutral) to predefined Spotify playlists.