CS 464

Introduction To Machine Learning

Project Final Report

Group 8:
Kürşad Güzelkaya 21902103
Bahadır Yüzlü 21903504
Hazal Buluş 21903435
Süleyman Gökhan Tekin 21902512
Mustafa Mert Türkmen 21902576

December 25, 2023

1. Introduction and Background Information

1.1. Introduction

In this project, we conducted an in-depth evaluation of four distinct machine learning and deep learning models—k-nearest neighbors (KNN), convolutional neural network (CNN), decision tree, and random forest—using the FER2013 dataset, which is aimed at recognizing human emotions through facial expressions. Our primary objective was to assess and compare how each of these models performs in the context of image classification tasks, with a particular focus on the accuracy of emotion detection. To enhance our understanding of the models' behaviors, we also explored the impact of adjusting various hyperparameters on their performance, ensuring a comprehensive analysis within the scope of this task.

In addition to model tuning, we implemented different data balancing techniques as part of the preprocessing stage to mitigate the effects of imbalanced datasets, which can skew the performance of predictive modeling. The techniques employed included Random OverSampling, Resampling, Synthetic Minority Oversampling Technique (SMOTE), and Balanced Bagging Classification.

KNN demonstrated limited success, achieving an accuracy peak of 34%. The Decision Tree and Random Forest models showed moderate improvements, with the latter slightly outperforming the former. However, the CNN stood out as the most effective, achieving a high accuracy of 58% and showcasing its aptitude for spatial information processing and feature extraction.

1.2 Problem Description

1.2.1. The Purpose of the Project

This project aims to develop a facial expression recognition system using a dataset from Kaggle known as FER-2013 [1]. The system's accuracy in identifying and classifying human emotions from facial images is of paramount importance. Our purpose within this project is to compare and contrast how different machine learning models perform on this dataset and investigate the reasons for the differences between them.

This endeavor utilizes a suite of machine learning algorithms that we have decided to use in our project, which include the k-nearest neighbors algorithm (KNN), convolutional neural network (CNN), decision trees (DT), and the random forest approach. The preliminary phase of implementing these models has been completed, and we are currently delving into the optimization phase, where we are experimenting with various hyperparameters to refine the results and enhance the models' accuracy. We will present an in-depth analysis of each model, exploring the impact of the hyperparameter adjustments and the underlying reasons for their performance in dedicated subsections of this report.

1.2.2. Structure of the Report

This report begins with Section 1, the Introduction, which provides a succinct summary of the entire project. Section 1.2, Problem Description, delves deeply into the problem, including details about the dataset, code repository, and the structure of the report. In Section 2, Methods, we explore the background and implementation specifics of each model in separate subsections. Section 3, Results, showcases empirical findings related to the process in individual subsections, presented without commentary. The Discussion in Section 4 engages with the impact of various hyperparameters and offers a comparative analysis of the models, drawing on insights from Section 3. The report culminates in Section 5 with a conclusion that synthesizes the results and metrics. Additionally, Section 6 outlines the contributions of each team member.

1.2.3. Code Repository

All code developed for this project is available in ipynb and py format and can be accessed through the GitHub repository at the following URL:
https://github.com/kursadguzelkaya0/EmotionDetection

1.2.4 Background Information Regarding The Dataset

The FER-2013 dataset is composed of 48x48 pixel grayscale images of faces, each displaying one of seven emotions: happiness, neutrality, sadness, fear, anger, surprise, and disgust. The training set includes 28,709 examples, while the test set comprises 3,589 examples, setting the stage for a robust analysis and model training process. The distribution of images across these emotions is as follows: 7,221 for happiness, 4,965 for neutrality, 4,830 for sadness, 4,097 for fear, 3,995 for anger, 3,171 for surprise, and 436 for disgust in the training set. The test set contains 1,774 happy, 1,239 neutral, 1,247 sad, 1,024 fearful, 958 angry, 831 surprised, and 111 disgusted expressions. Faces are centrally aligned to ensure uniformity. The dataset is notably imbalanced, with certain emotions like happiness being over-represented, while others, such as disgust, are under-represented.

To tackle the challenge of an imbalanced dataset, several techniques were employed, including Data Augmentation, Random OverSampling, Resampling, Synthetic Minority Oversampling Technique (SMOTE), and Balanced Bagging Classification. Data Augmentation was utilized to artificially enhance the dataset's diversity by modifying existing images through transformations like rotations and color adjustments, which helped in improving the model's generalization capabilities. Random OverSampling involved replicating instances from underrepresented classes to achieve a balance, while Resampling adjusted the dataset to ensure a more equal representation of all emotional categories. SMOTE played a crucial role in creating new, synthetic samples from these minority classes, introducing additional variation and addressing the class imbalance issue more effectively. Furthermore, Balanced Bagging Classification was implemented as a strategy to ensure each subset of the dataset used in the ensemble approach was balanced, further contributing to the integrity and fairness of the emotion classification model.

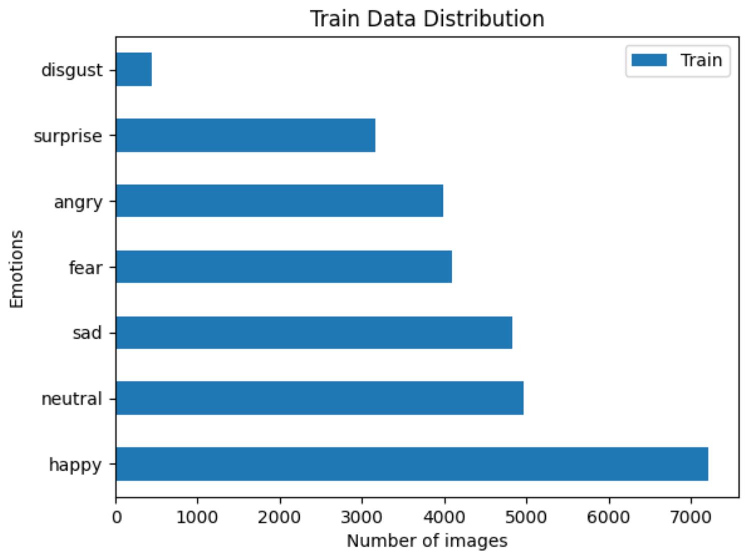


Figure 1. Distribution of train data

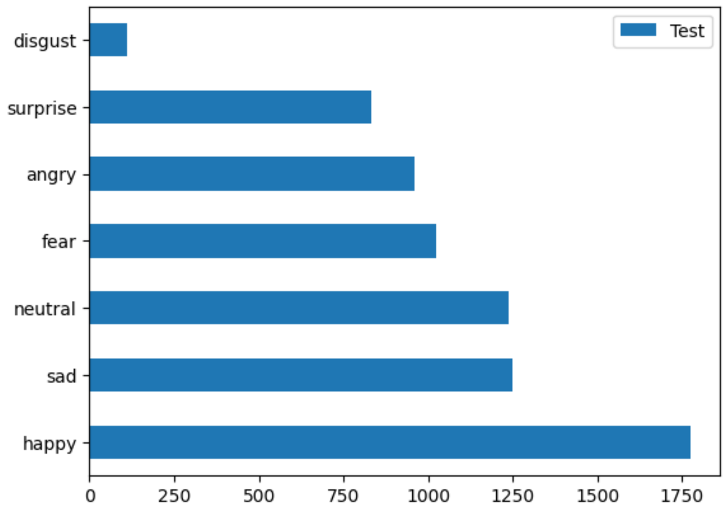


Figure 2. Distribution of test data

We calculated and plotted the Proportion of Variance Explained (PVE) by different numbers of principal components in a Principal Component Analysis (PCA). Specifically, it iterates through 1 to 199 principal components, applies PCA to the training data, and records the cumulative variance explained by each number of components. By plotting the cumulative Proportion of Variance Explained (PVE) against the number of components, one can identify the point of diminishing returns, where adding more components does not significantly increase the explained variance. This helps in dimensionality reduction, retaining most of the information while simplifying the model and potentially improving computational efficiency.

As seen in Figure 3, the graph exhibits a rapid increase in the proportion of variance explained (PVE) as the number of principal components rises, plateauing near 150 components, indicating that most of the dataset's variability can be captured with this number of components.

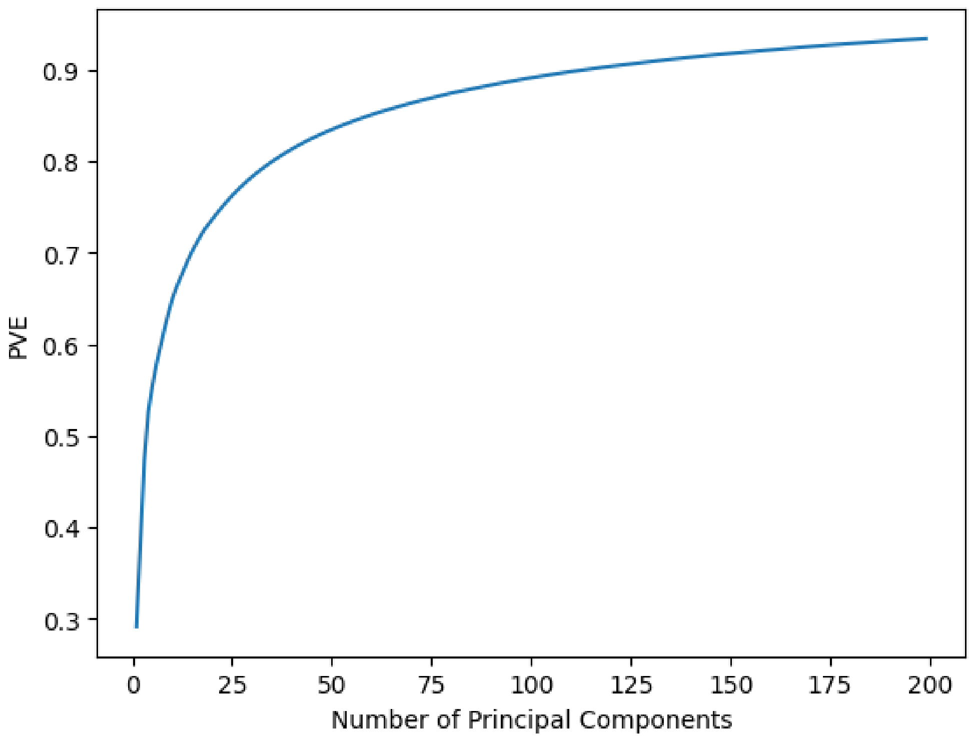


Figure 3. Graph of PVE-PCA

2. Models

2.1 KNN

2.1.1 Background Information Regarding KNN

The k-Nearest Neighbors (KNN) algorithm is a simple yet versatile machine learning algorithm used for both classification and regression tasks, though it is more widely known for classification. Fundamentally, KNN functions based on the idea that related objects are located close to one another. In actual use, it locates the 'k' nearest training data points to a given test point and uses these neighbors to predict the label for this test point. The choice of 'k' is important because a smaller 'k' makes the algorithm more susceptible to noise, whereas a bigger 'k' increases computational cost and may result in less accuracy. The simplicity of KNN lies in its ability to provide predictions without requiring a training phase. Instead, it computes the distance between each training point and the test point using metrics like the Manhattan or Euclidean distances. Even though this method is simple, it can become ineffective when dealing with huge datasets, which can result in high processing expenses. Nevertheless, KNN is an invaluable tool in a data scientist's toolbox due to its intuitive design and versatility.

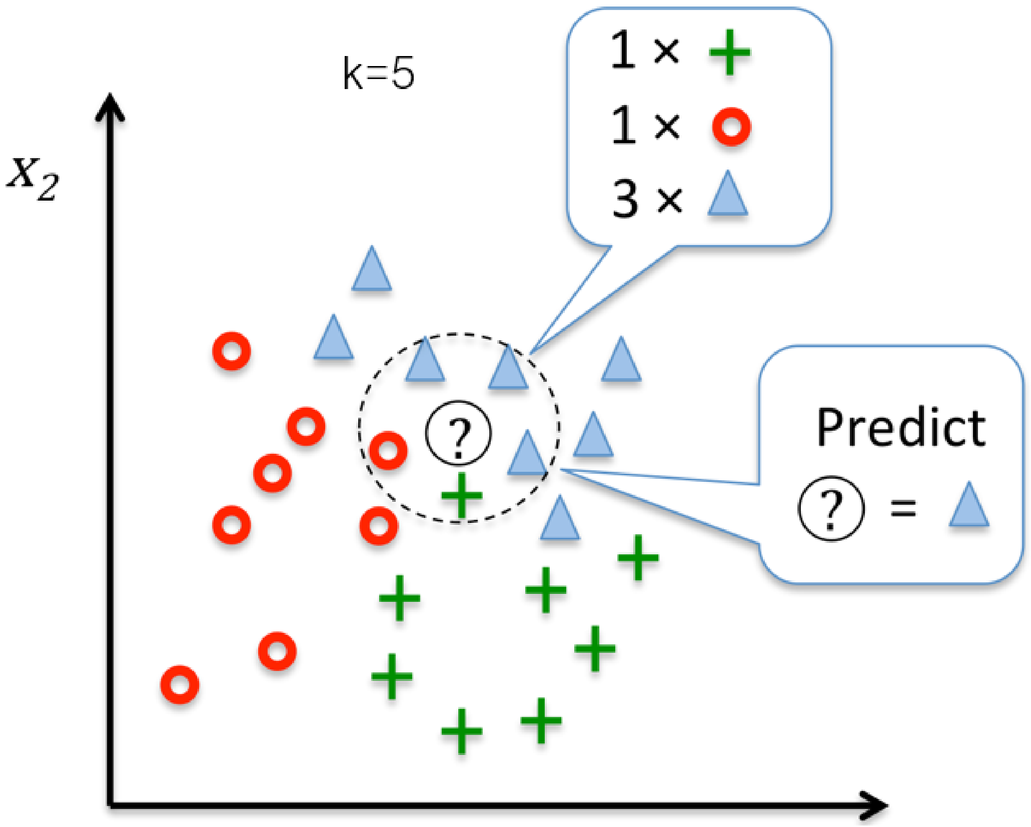


Figure 4. Diagram showing working of KNN. [3]

2.1.2 Implementation and Considered Hyperparameters For KNN

The KNN classifier was built utilizing the scikit-learn library, with preliminary steps taken to preprocess the image data from its original .jpg format. As the dataset exhibited class imbalance, we also applied several techniques, including Random OverSampling, Resampling, SMOTE, and Balanced Bagging Classification, as part of our preprocessing strategy. These steps aimed to provide a more balanced data distribution for the KNN classifier. Despite these comprehensive preprocessing efforts, the improvement in the KNN model's performance was modest, suggesting that KNN may not be the most effective model for facial emotion recognition within our dataset. To explore dimensionality reduction, PCA was applied, but its impact on model performance was less than favorable. The details about the process can be seen in the results section.

2.2. Decision Tree

2.2.1 Background Information Regarding Decision Trees

The decision tree algorithm resembles a sorting tool for data, creating a tree-shaped structure. Nodes in the tree make decisions based on specific characteristics of the data. The initial node, called the "root," starts the decision process, and subsequent nodes, known as "internal" nodes, refine the categorization. The final points in the tree, called "leaves," provide the ultimate classification or prediction for the target attribute [5].

Decision trees are functional, working well with different types of data, such as categories and numbers. The goal is to keep the tree simple and easy to understand. This simplicity is determined by factors like the number of decision points and the depth of the tree. This part is going to be elaborated in the tuning part.

In the context of modeling for image processing, the decision tree algorithm stands out as a viable option despite concerns about overfitting commonly associated with its use. While decision trees are recognized for their tendency to capture a broad range of data variations, their distinctive characteristics make them suitable for tasks in image processing [6]. Notably, decision trees provide an understandable and handy structure, which is particularly valuable in the complicated field of image analysis. Ensemble methods, such as random forests built upon decision trees, contribute to the model's robustness by addressing concerns about overfitting. However, random forest models may face challenges with the gradient search algorithm in image processing. This complexity can make fine-tuning more intricate. Conversely, the simplicity of decision tree models provides an advantage in this context. Due to their straightforward structure, decision trees offer an easier process for fine-tuning. Overall, decision trees prove robust due to their interpretability, adaptability to feature scales, and simplified fine-tuning.

A sample decision tree model can be shown as follows,

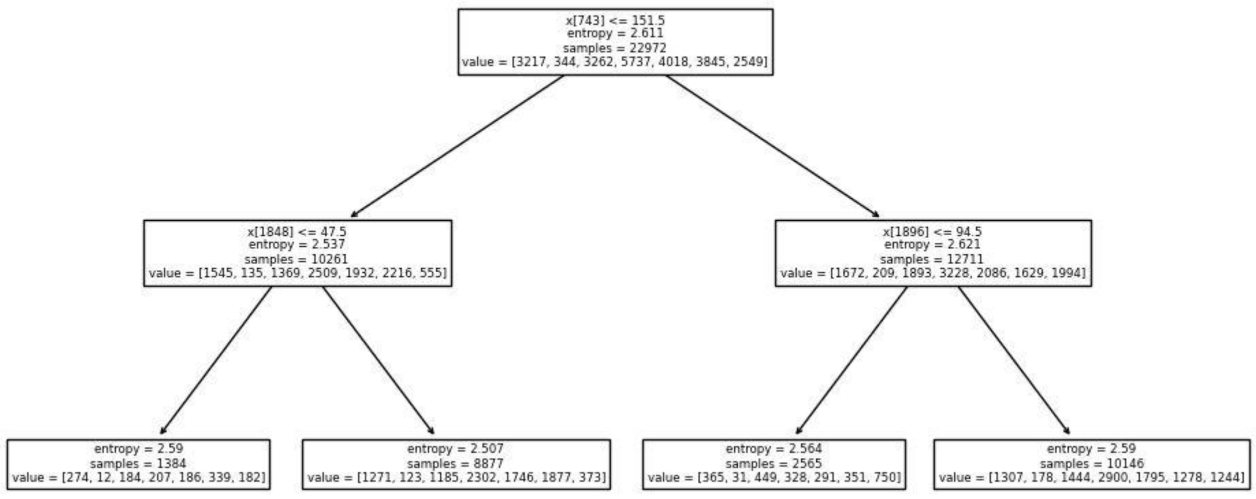

x[743] <= 151.5
entropy = 2.611
samples = 22972
value = [3217, 344, 3262, 5737, 4018, 3845, 2549]
x[1848] <= 47.5
entropy = 2.537
samples = 10261
value = [1545, 135, 1369, 2509, 1932, 2216, 555]
x[1896] <= 94.5
entropy = 2.621
samples = 12711
value = [1672, 209, 1893, 3228, 2086, 1629, 1994]
entropy = 2.59
samples = 1384
value = [274, 12, 184, 207, 186, 339, 182]
entropy = 2.507
samples = 8877
value = [1271, 123, 1185, 2302, 1746, 1877, 373]
entropy = 2.564
samples = 2565
value = [365, 31, 449, 328, 291, 351, 750]
entropy = 2.59
samples = 10146
value = [1307, 178, 1444, 2900, 1795, 1278, 1244]

Figure 5. The decision tree we generated using the dataset (max_depth = 2, criterion = entropy)

2.2.2. Implementation and Considered Hyperparameters For Decision Tree

We implemented a DecisionTreeClassifier using scikit-learn with parameters such as criterion, depth, min_samples_split, and min_samples_leaf. Notably, changes in criterion, depth, and other parameters had a significant impact on model accuracy, while min_samples_split and min_samples_leaf showed less noticeable effects.

To eliminate skewness, we experimented with multiple methods and selected the most effective one for each model. For the decision tree, the optimal choice was the BalancedBaggingClassifier, emphasizing its efficacy in addressing imbalances and contributing to enhanced model performance. The details about the process can be seen in the results section.

2.3 Random Forest

2.3.1 Background Information Regarding Random Forests

Random Forest is a strong ensemble learning technique comprising an assembly of decision trees. Each decision tree within the ensemble is constructed utilizing a distinct, or partially distinct, subset of the training dataset. The diversity in these subsets is achieved through techniques such as bootstrap sampling. Subsequently, predictions are made based on the compound of predictions from these decision trees. The final prediction of the Random Forest is determined by a majority vote mechanism, where the most frequently predicted class decides the class assigned to a given sample across the ensemble. For instance, if 58% of the decision trees expect a particular model belonging to class happy during testing, the Random Forest assigns the sample to class happy. This mechanism not only enhances predictive accuracy but also imparts resilience to overfitting and improves the overall generalization capability of the model [2].

Several factors exert influence on the accuracy and predictive performance of Random Forests. Foremost among these is the number of decision trees constituting the ensemble. When the quantity of decision trees is set to one, the Random Forest resembles a singular decision tree. The accuracy of the Random Forest generally demonstrates improvement with an increasing number of decision trees, provided that the augmented quantity leads to a manageable reduction in tree heights or predictive capabilities due to the relatively limited number of samples [2]. Additionally, the distribution of the training samples among the ensemble, encompassing considerations such as the maximum number of samples allocated to each decision tree, plays a pivotal role in determining the overall accuracy. Furthermore, any factor influencing the accuracy of individual decision trees is inherently appropriate to the Random Forest, given its foundational nature as a collection of such trees. Consequently, factors impacting the accuracy of decision trees contribute in common to the predictive efficacy of the Random Forest model. The architecture can be seen in Figure 8, in which light greens represent their class.

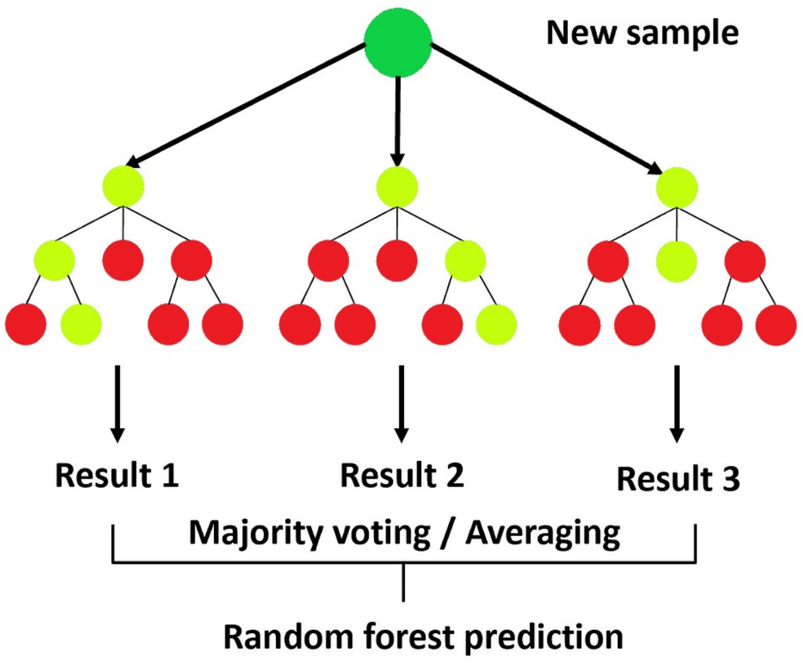


Figure 6. Random Forest Simplified

2.3.2. Implementation and Considered Hyperparameters For Random Forest

We implemented the random forest using scikit-learn's random forest classifier model. We mainly focus on two hyperparameters, which are n_estimators and min_samples_split. n_estimators number denotes how many decision trees the random forest constructs to make predictions. min_samples_split is the minimum number of samples required to split an internal node.

We trained the models and got outputs by tuning our hyperparameters for different oversampling models. We also considered model training times, so we gave up training on BalancedBaggingClassifier because training the models took the same time as a neural network, and the results did not increase that much. In the previous report, we applied the PCA and looked at the outcome, but in this time we did not, because the results we obtained with PCA were bad. Considering all these, we got our best result in SMOTE. The details about the process can be seen in the results section.

2.4 CNN

2.4.1 Background Information

CNN is a famous deep-learning architecture that is used mainly for image classification tasks. It is also successfully used in natural language processing, recommender systems, and other fields. CNN's primary benefit over its predecessors is that it can identify essential elements automatically without human oversight. For instance, it can recognize unique characteristics for each class on its own by being shown numerous images of dogs and cats.

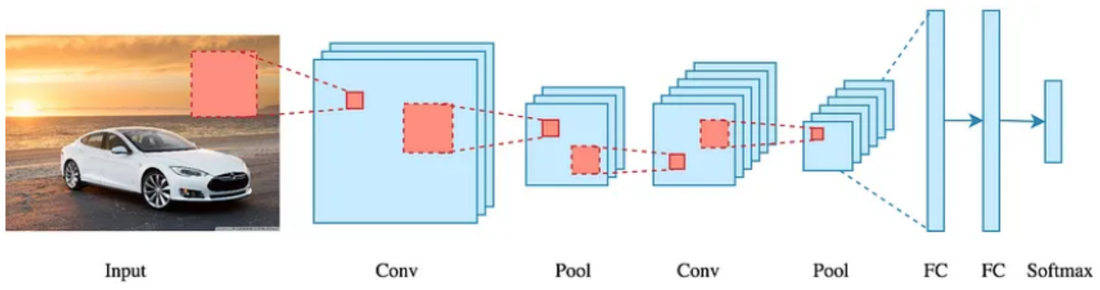


Figure 7: CNN Architecture [4]

All CNN models have an architecture similar to Figure 9. We are utilizing an input image at this time. We first do a sequence of pooling and convolution processes, then several fully linked layers. Softmax is the result of multiclass classification being done.

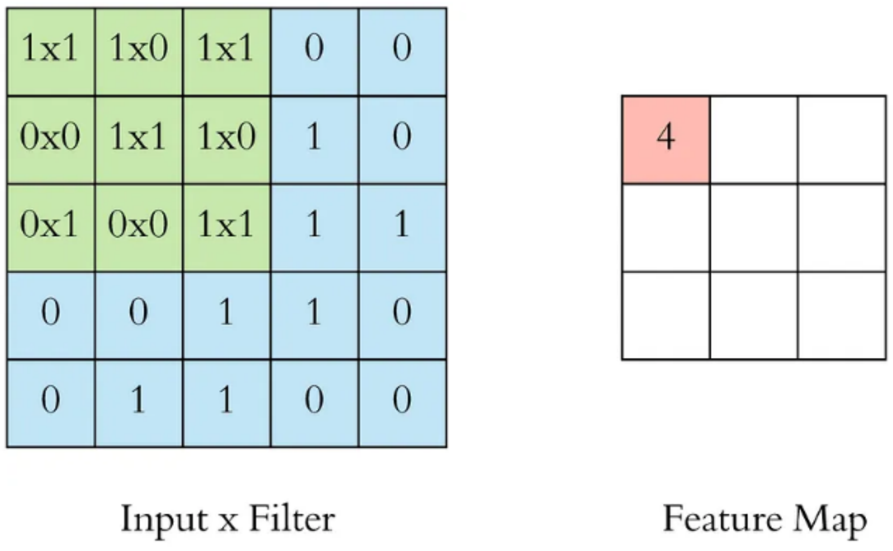


Figure 8: CNN Feature Map Extraction From Input and Filter [4]

The convolutional layer is the primary component of CNN. A mathematical process called convolution is used to combine two sources of data. In this instance, a convolution filter is used to apply the convolution to the input data in order to create a feature map. We apply this filter to the input and do the convolution procedure. We perform element-wise matrix multiplication at each point and add the outcome. The feature map uses this total. The receptive field is the green region where the convolution procedure occurs. Owing to the filter's dimensions, the receptive field measures 3 by 3.

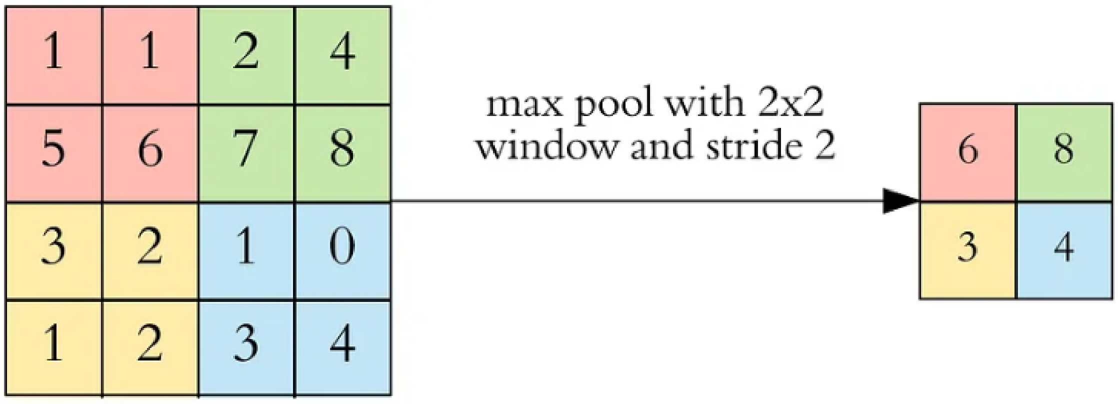


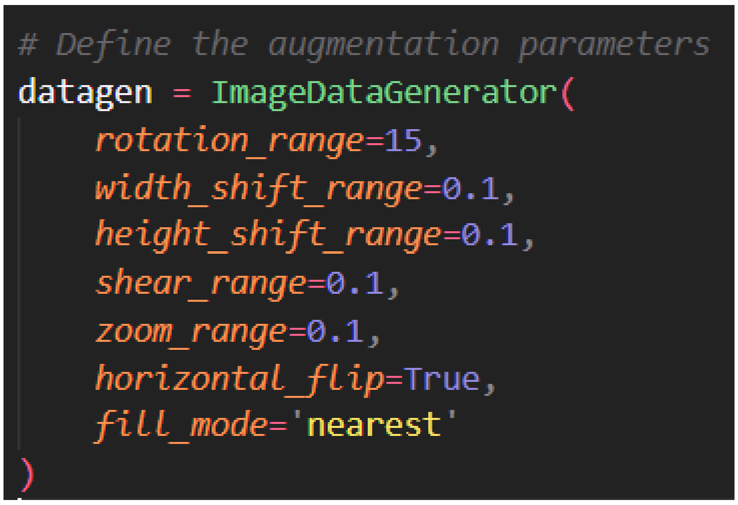
Figure 9: Max Pooling [4]

Neighbor pixel values are typically fairly close to one another, which results in some extra information being stored after the convolution. Typically, pooling is done after a convolution operation to lower the dimensionality. This allows us to decrease the number of parameters, which helps prevent overfitting and speeds up the training process. Max pooling, which simply takes the maximum value within the pooling window, is the most popular kind of pooling. This is the outcome of max pooling with stride 2 and a 2x2 window. Every color represents a distinct window.

There are some methods used with CNN to prevent overfitting. Dropout is a very basic idea that is used to prevent overfitting. Every iteration during training causes a neuron to be momentarily "dropped" or disabled with probability p. This indicates that at this iteration, this neuron's inputs and outputs will all be disabled. Data augmentation is a way to generate more training data from our current set. It enriches or “augments” the training data by developing new examples via random transformation of existing ones. This way we artificially boost the size of the training set, reducing overfitting.

2.4.2 Implementation For CNN

2.4.2.1 Data Preprocessing - Augmentation



```
# Define the augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Figure 10: Data Augmentation Code Example

The code example in Figure 10 initializes a Keras ImageDataGenerator for image augmentation and preprocessing. Firstly, the augmentation process using ImageDataGenerator from TensorFlow's Keras applies various transformations like rotation, shifting, and flipping to increase the number of images for the 'disgust' class to reach about 3000 images. Additionally, it ensures dataset balance by copying the first 3000 images from other emotion classes. This balance is crucial for training a more robust and unbiased model. When combined, these augmentation methods improve the dataset's quality and boost the neural network's capacity for pattern recognition and learning.

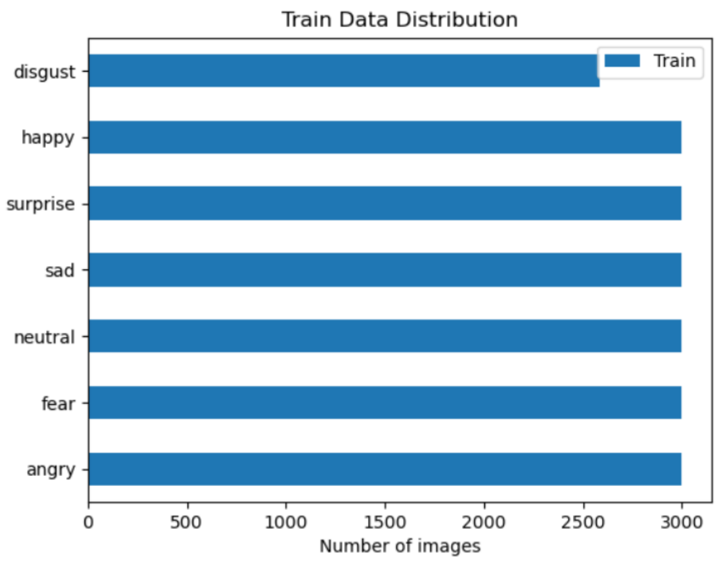
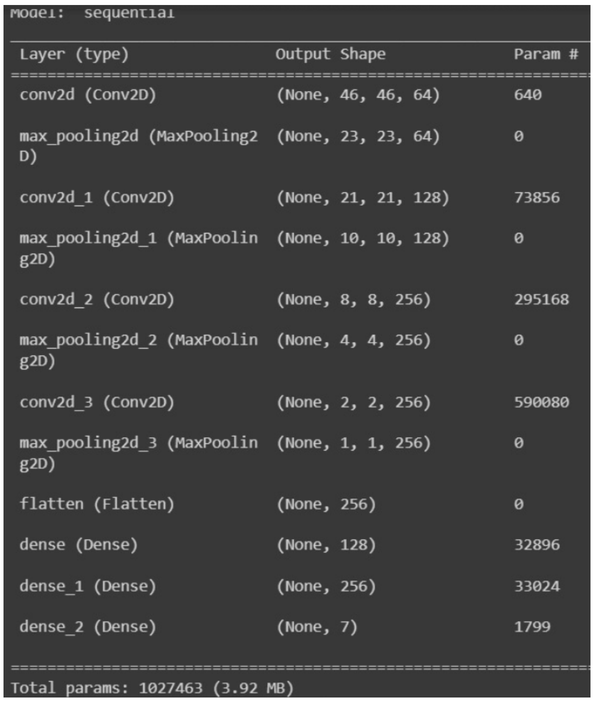


Figure 11: Data Distribution After Augmentation

2.4.2.2 Model Architechture

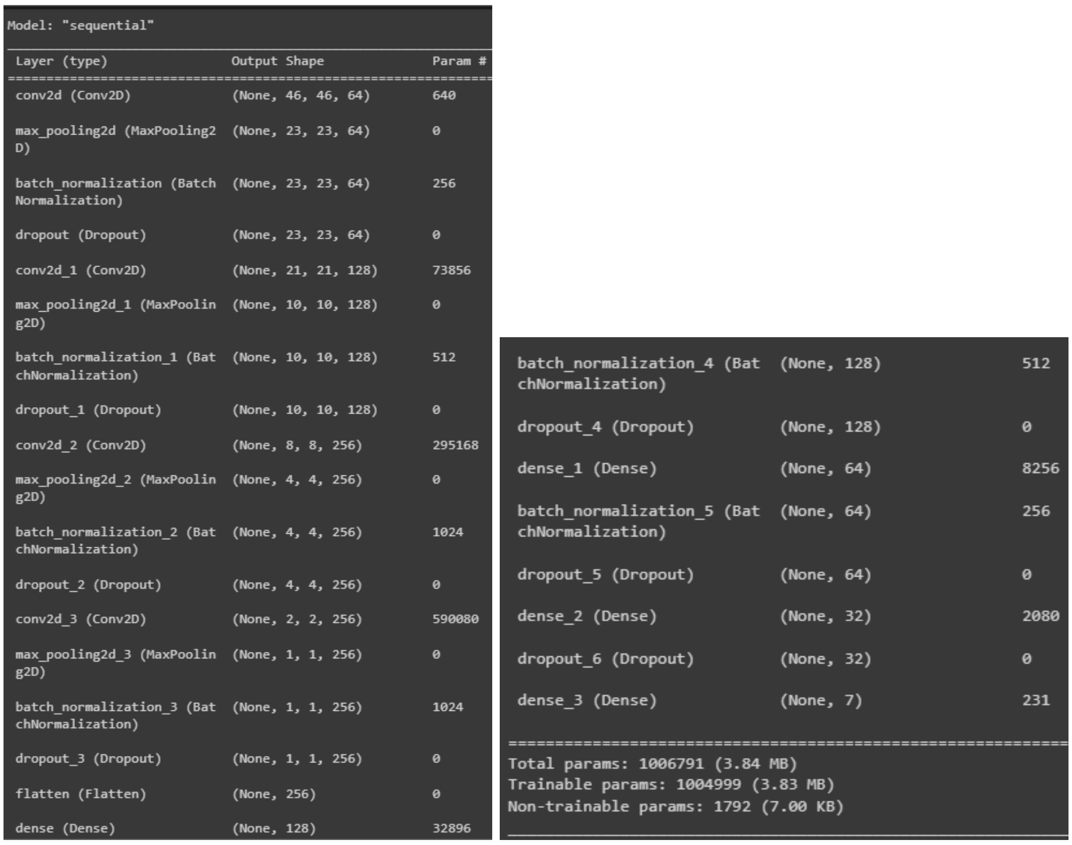
There are 12 layers consisting of four convolutional, four max pooling, one flatten layer, and three dense layers in our CNN architecture. ReLu is used as an activation function for all layers except the output layer, which uses the softmax activation function for the finalization of classification. Here is a snapshot of our network architecture generated by TensorFlow:



```
Model: sequential

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 64)        640

 max_pooling2d (MaxPooling2  (None, 23, 23, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 21, 21, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 128)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 2, 2, 256)         590080

 max_pooling2d_3 (MaxPoolin  (None, 1, 1, 256)         0
 g2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dense_1 (Dense)             (None, 256)               33024

 dense_2 (Dense)             (None, 7)                 1799

=================================================================
Total params: 1027463 (3.92 MB)
```

Figure 12: Description of Initial CNN

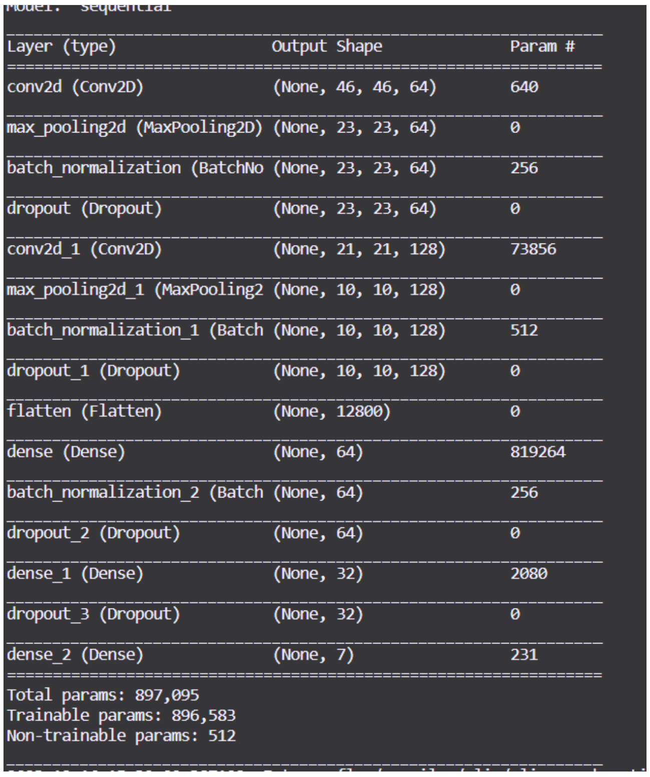
Then to prevent overfitting and improve accuracy, 6 Batch Normalization layers and 7 Dropout layers have been added between layers. A snapshot of improved CNN architecture can be seen in Figure 13.



```
Model: "sequential"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 64)        640

 max_pooling2d (MaxPooling2  (None, 23, 23, 64)        0
 D)

 batch_normalization (Batch  (None, 23, 23, 64)        256
 Normalization)

 dropout (Dropout)           (None, 23, 23, 64)        0

 conv2d_1 (Conv2D)           (None, 21, 21, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 128)       0
 g2D)

 batch_normalization_1 (Bat  (None, 10, 10, 128)       512
 chNormalization)

 dropout_1 (Dropout)         (None, 10, 10, 128)       0

 conv2d_2 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 batch_normalization_2 (Bat  (None, 4, 4, 256)         1024
 chNormalization)

 dropout_2 (Dropout)         (None, 4, 4, 256)         0

 conv2d_3 (Conv2D)           (None, 2, 2, 256)         590080

 max_pooling2d_3 (MaxPoolin  (None, 1, 1, 256)         0
 g2D)

 batch_normalization_3 (Bat  (None, 1, 1, 256)         1024
 chNormalization)

 dropout_3 (Dropout)         (None, 1, 1, 256)         0

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 128)               32896

 batch_normalization_4 (Bat  (None, 128)               512
 chNormalization)

 dropout_4 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 batch_normalization_5 (Bat  (None, 64)                256
 chNormalization)

 dropout_5 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 32)                2080

 dropout_6 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 7)                 231

=================================================================
Total params: 1006791 (3.84 MB)
Trainable params: 1004999 (3.83 MB)
Non-trainable params: 1792 (7.00 KB)
```

Figure 13: CNN Architecture 1

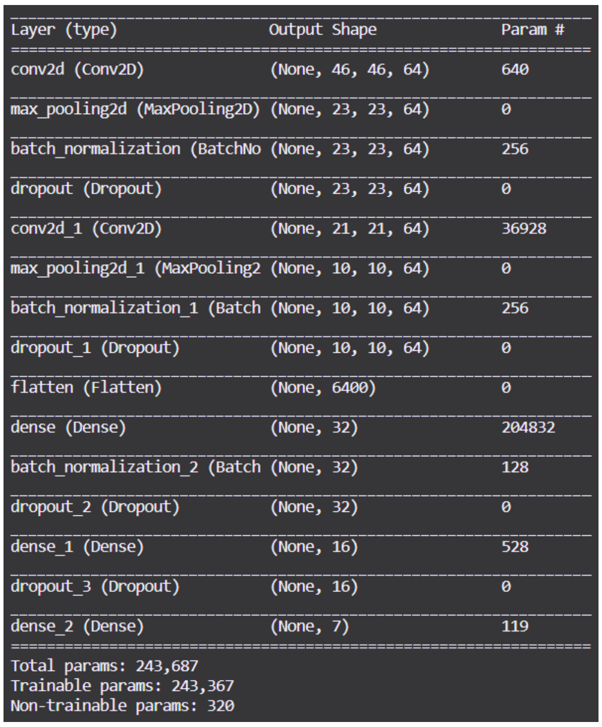
Since this architecture with 1 million parameters is too complex and takes so much time, I decreased the number of layers and learnable parameters. As seen in Figure 14, CNN has 897,095 parameters.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 23, 23, 64)        256
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 10, 128)       512
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 10, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 12800)             0
_________________________________________________________________
dense (Dense)                (None, 64)                819264
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 231
=================================================================
Total params: 897,095
Trainable params: 896,583
Non-trainable params: 512
_________________________________________________________________
```

Figure 14: CNN Architecture 2

Then in order to see the change in the accuracy with a much less complex CNN model, we decreased the number of parameters to 243,687, as seen in Figure 15.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 23, 23, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 23, 23, 64)        256
_________________________________________________________________
dropout (Dropout)            (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 32)                204832
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                528
_________________________________________________________________
dropout_3 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 119
=================================================================
Total params: 243,687
Trainable params: 243,367
Non-trainable params: 320
```

Figure 15: CNN Architecture 3

2.4.2.3 Training and Evaluation

The networks are trained on the original training data and balanced training data for 20 epochs using categorical cross entropy as a loss function. The accuracy is evaluated on test data, and the results are given in the results section.

3. Results

3.1. Metrics

When evaluating the models, we primarily evaluated the accuracy of the models but then we calculated the F1 scores since the classes in our dataset are not distributed evenly and there is class imbalance.

3.2. K Nearest Neighbors (KNN)

We applied the k-nearest neighbors (KNN) model to the imbalanced FER2013 dataset with the goal of determining the optimal k-value that yields the highest accuracy. After extensive testing, it was found that k=5 provided the best results, with the accuracy peaking at 34%. However, it was observed that beyond this point, as the k-value continued to increase, the model began to underfit, evidenced by a decline in accuracy. Generally, the accuracy of the KNN model on the imbalanced dataset hovered around the low 33% range.

K-Value	Accuracy
2	34.28%
5	34.57%
10	33.15%
15	32.49%

Table 3: Accuracy Values for KNN Classifier with Uniform Weights

Subsequently, we focused on preprocessing the dataset using various data balancing techniques to enhance the KNN model's performance. The techniques applied were Random OverSampling, Resampling, Synthetic Minority Oversampling Technique (SMOTE), and Balanced Bagging Classification, with their effectiveness reflected in the F1 scores presented in the accompanying chart. Despite the application of these techniques, the highest F1 score was achieved using Random OverSampler, which was approximately 32%—interestingly, still slightly below the performance of the model on the default, imbalanced dataset. This suggests that while data balancing is typically beneficial for model performance, in this specific instance, the inherent characteristics of the dataset and the chosen model may require a more nuanced approach to preprocessing for improving classification outcomes.

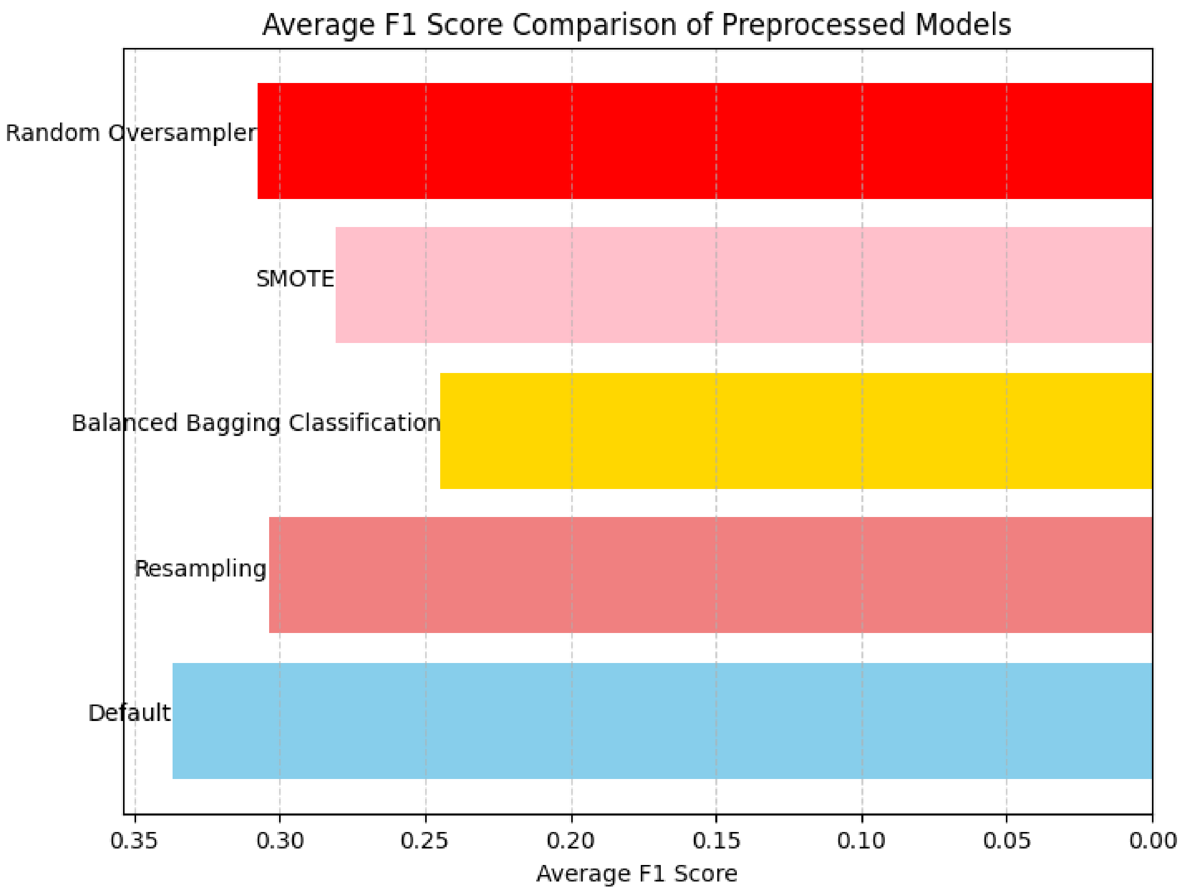
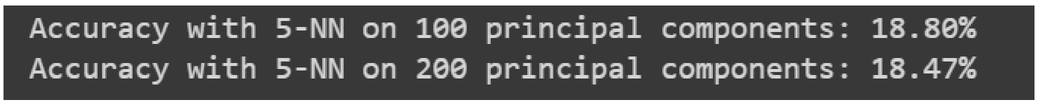


Figure 16: Weighted F1 Score Comparison between Preprocessed Models

PCA was executed with two different numbers of principal components to observe the variance in accuracy: first with 100 and then with 200 principal components. Contrary to the anticipated improvement, the introduction of PCA resulted in a significant decrease in accuracy. This substantial drop suggests that while PCA is often used to reduce dimensionality and improve computational efficiency, in the case of our emotion classification task, reducing the dataset to 100 or 200 principal components might have led to the loss of critical information that was essential for accurate emotion recognition.



```
Accuracy with 5-NN on 100 principal components: 18.80%
Accuracy with 5-NN on 200 principal components: 18.47%
```

Figure 17: The Accuracy Values with PCA

3.3. Decision Tree

- Preprocess

Initiating the data preprocessing phase, to enhance computational efficiency, a simplification strategy was adopted, resulting in the compression of images into a unidimensional format. Unlike some machine learning algorithms that are sensitive to the scale of input features, decision trees inherently accommodate varying scales without compromising their effectiveness. The tree structure's split decisions are based on feature thresholds rather than absolute values, allowing it to handle features with different scales naturally. In this way, there's no requirement to standardize or normalize features, contributing to a more straightforward and efficient modeling process.

Recognizing the skewness in the dataset (disgust category) makes correct measuring impossible. Due to the imbalance in the trained data, various imputation models were employed for equalization, with the most promising outcome achieved through Balanced Bagging Classification, yielding an weighted F1 score of 34.18%. This additional classification layer, contributing to the visualization, addresses the imbalanced nature of the dataset. While Balanced Bagging Classification introduces its own set of parameters, for simplicity, hyperparameter tuning will be applied only to the decision tree parameters.

Let's show the performances of different preprocesses in a chart:

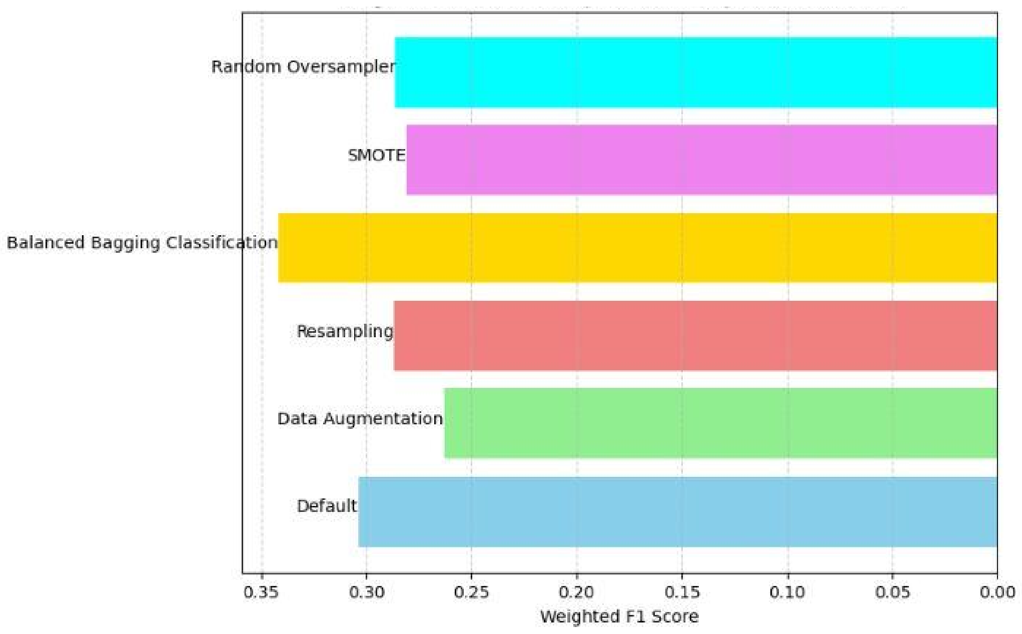


Figure 18: Weighted F1 Score Comparison between Preprocessed Models

- Tuning

A grid search has been performed over the model. However, a grid parameter including 10 variant models runs about 16 hours. Therefore, only essential parameters should be chosen as a search area, and 2 parameters at a time should be in the grid to sketch a 2-dimension graph. Because of this, prior information on what to search is essential. Here are the definitions of the parameters to be searched;

Criterion: The criterion parameter determines the function used to measure the quality of a split. For decision trees, this is the metric used to evaluate how well a split separates the data into classes.

Maximum Depth: It limits the number of nodes in the tree, preventing it from becoming too complex and overfitting the training data.

Minimum Impurity Decrease: This parameter determines the minimum amount of impurity decrease required for a split to occur. It helps control the growth of the tree by avoiding splits that do not significantly improve the model.

Minimum Samples Leaf: The min_samples_leaf parameter sets the minimum number of samples required to be in a leaf node. It controls the size of the leaves in the tree.

In optimizing a decision tree algorithm, a grid search was conducted with six distinct configurations, varying the criterion (Gini and entropy) and max depth parameters (6,8,10). The analysis revealed that the model with Gini as the criterion and a max depth of 10 performed best through accuracy;

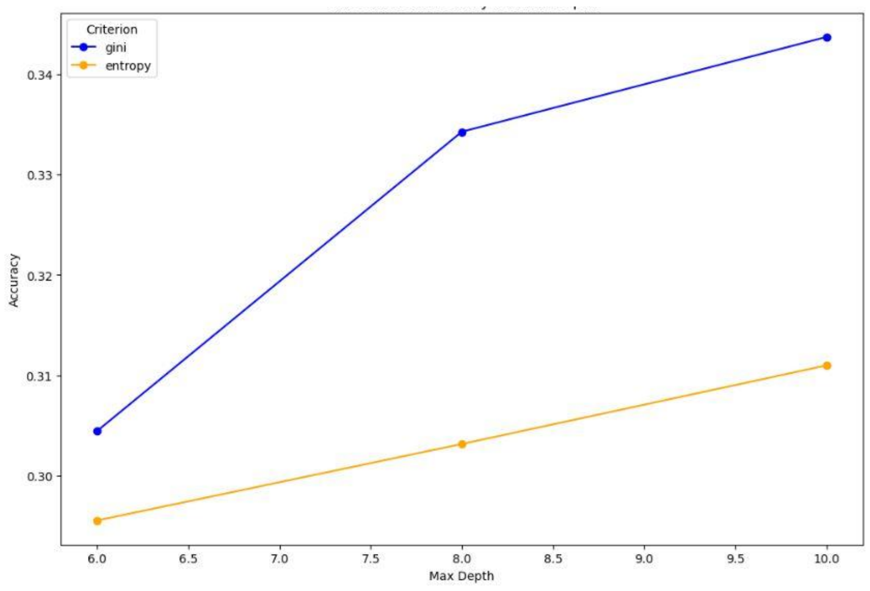


Figure 19. Accuracy values with respect to max_depth vs. criterion

A secondary grid search was conducted to refine the decision tree model's hyperparameters, explicitly focusing on max depth and minimum impurity decrease. The search space included max depth values of 10, 12, and 14, coupled with minimum impurity values of 0, 0.25 utilizing the gini criterion that demonstrated superior performance in the preceding analysis. This investigation is about capturing subtle data patterns and avoiding overfitting, which helps improve the overall decision tree model. The 0 value at depth 10 gave by far the best result among others. This means that the model is underfitting above the minimum impurity decrease values. Here is the result;

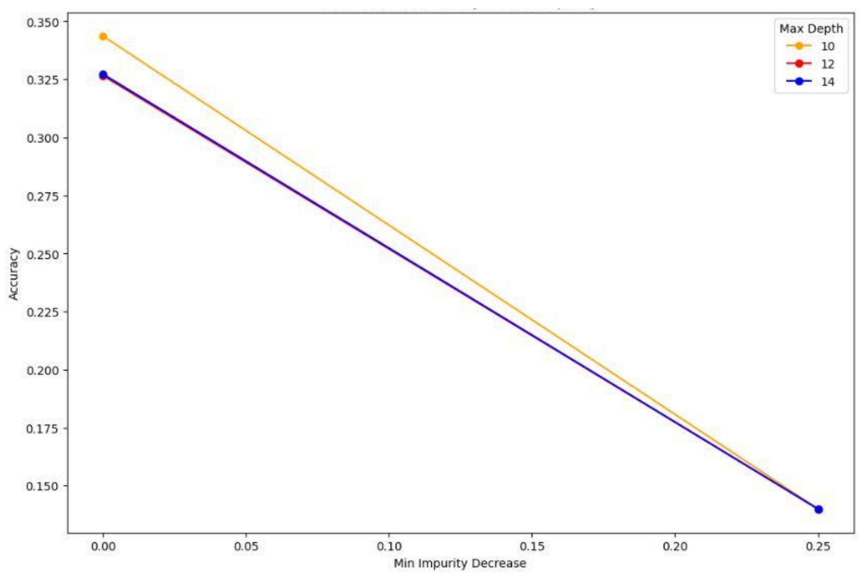


Figure 20: Accuracy values with respect to max_depth vs. min_impurity_decrease

For the exploration of the minimum samples leaf parameter with a focus on accuracy, a simplified approach was employed, testing only two models. The investigation revealed that the model with a minimum sample leaf set to 2 exhibited superior performance compared to the alternative configuration. The choice of a smaller minimum sample leaf value allows the model to capture more complicated patterns in the data, potentially leading to improved accuracy.

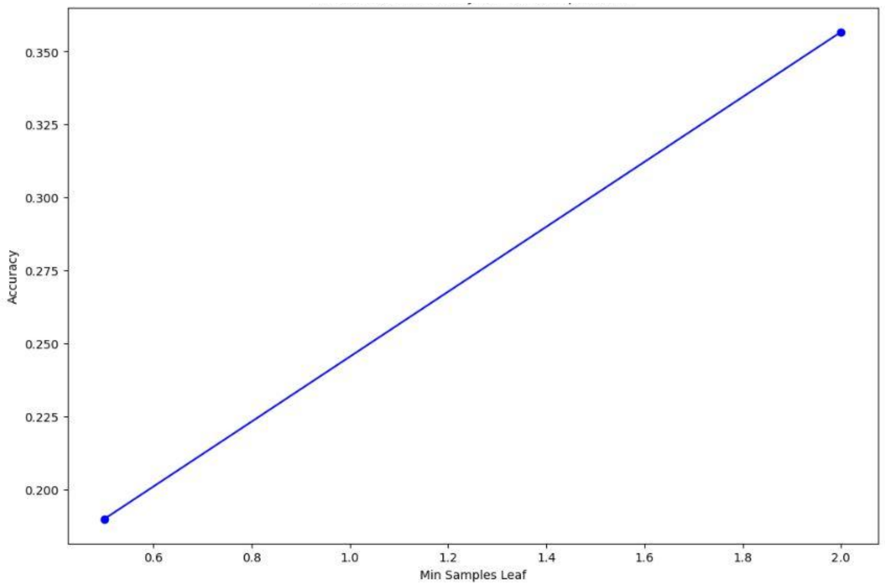
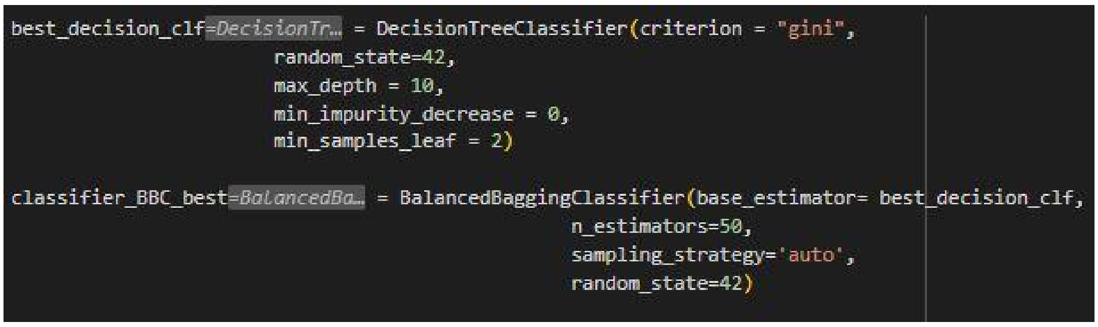


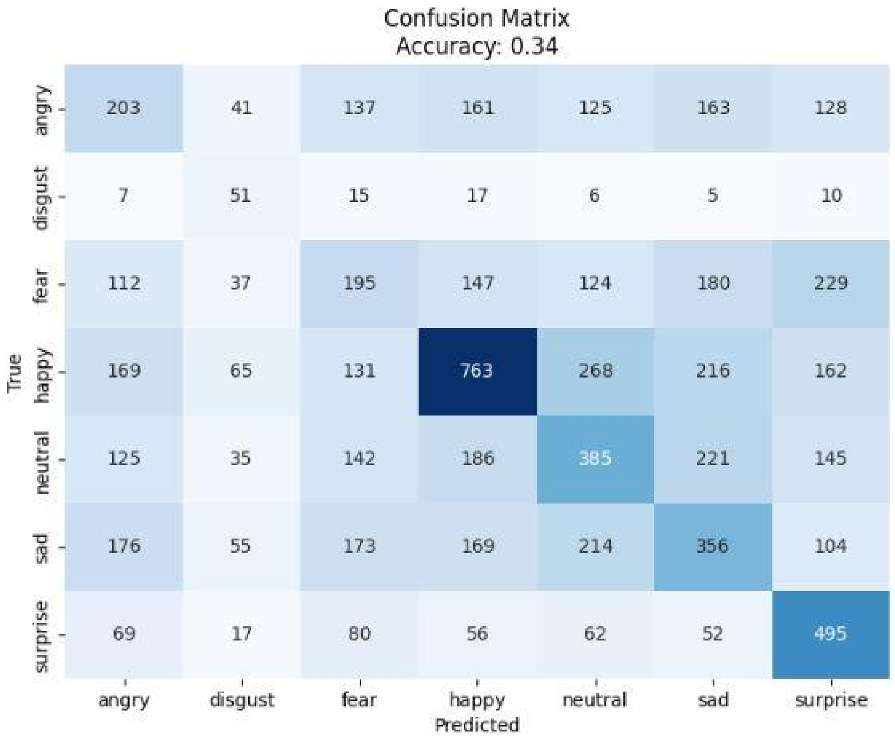
Figure 21: Accuracy values with respect to min_samples_leaf

All in all the best model is;



```
best_decision_clf=DecisionTr… = DecisionTreeClassifier(criterion = "gini",
                              random_state=42,
                              max_depth = 10,
                              min_impurity_decrease = 0,
                              min_samples_leaf = 2)

classifier_BBC_best=BalancedBa… = BalancedBaggingClassifier(base_estimator= best_decision_clf,
                                  n_estimators=50,
                                  sampling_strategy='auto',
                                  random_state=42)
```

Figure 22: The Parameters of The Best Classifier



True \ Predicted	angry	disgust	fear	happy	neutral	sad	surprise
angry	203	41	137	161	125	163	128
disgust	7	51	15	17	6	5	10
fear	112	37	195	147	124	180	229
happy	169	65	131	763	268	216	162
neutral	125	35	142	186	385	221	145
sad	176	55	173	169	214	356	104
surprise	69	17	80	56	62	52	495

Figure 23. Confusion Matrix of the Best Classifier

3.4. Random Forest

As we did in other parts, we tried to reduce the effect of imbalance in the data with different methods and looked at its effect on our f1 score.

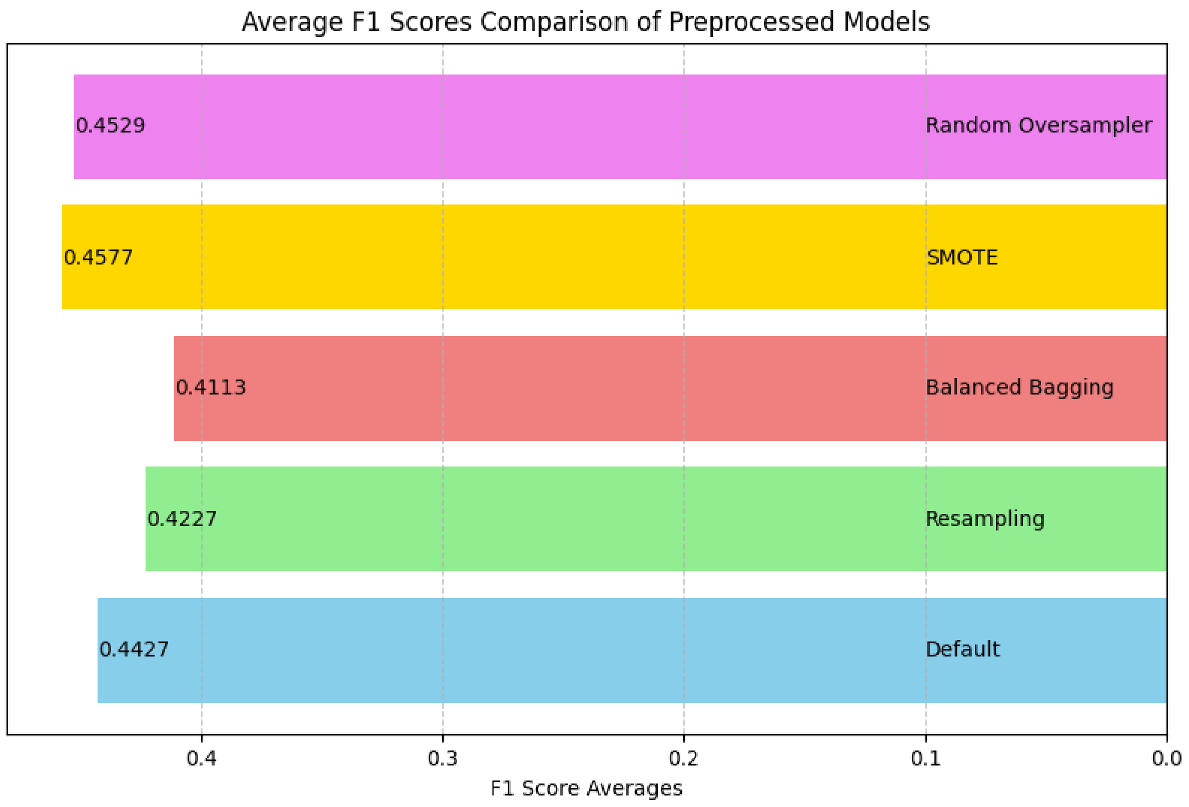


Figure 24: Weighted F1 Score Comparison between Preprocessed Models

In general, the random forest model performs much better than the decision trees, as expected. Also, handling the imbalanced data improved our model predictability in a better way. We can easily say that by looking the Figure 24. The main hyperparameter here is the n_estimator hyperparameter, which is fundamentally how many decision trees are found in the random forest. We also looked at min_samples_split, which we did not look at the decision tree part. These hyperparameter tunings are done for each oversampling method separately, except the Balanced Bagging Classification, because of its high running time without achieving high accuracies.

Resampling

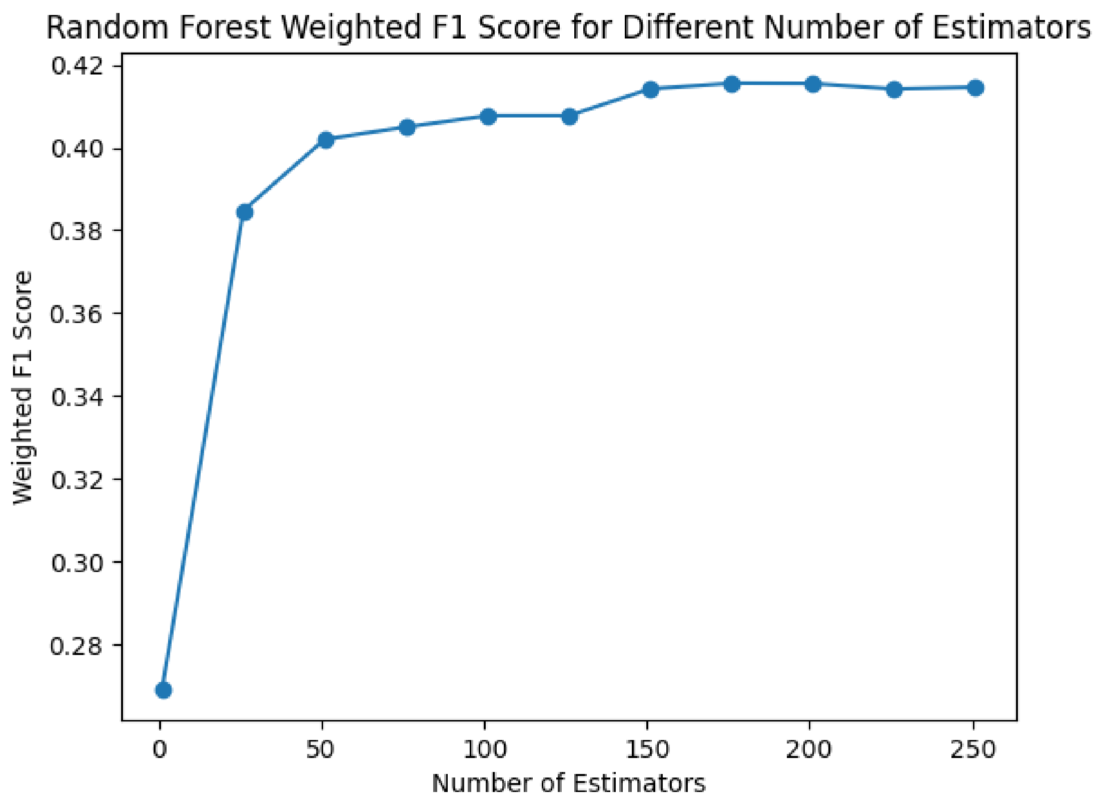


Figure 25. Weighted f1 Scores with respect to the number of estimators for Resampling

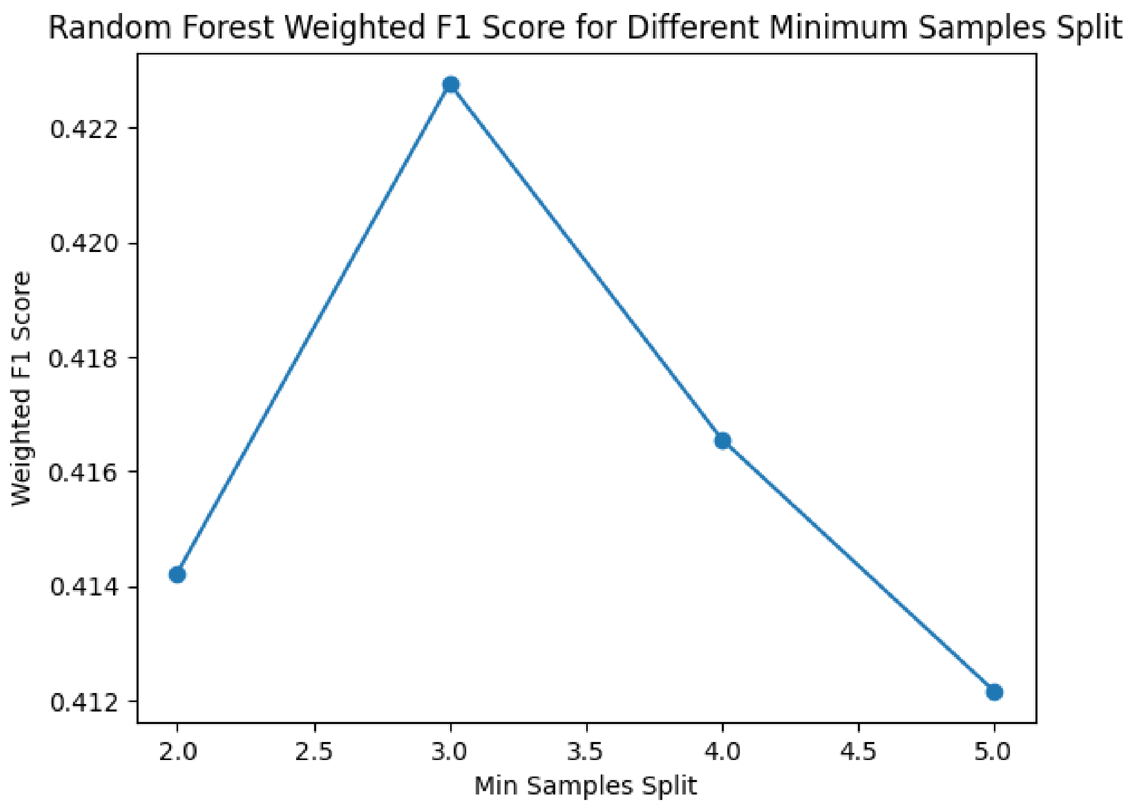


Figure 26. Weighted f1 Scores with respect to minimum samples split for Resampling

SMOTE

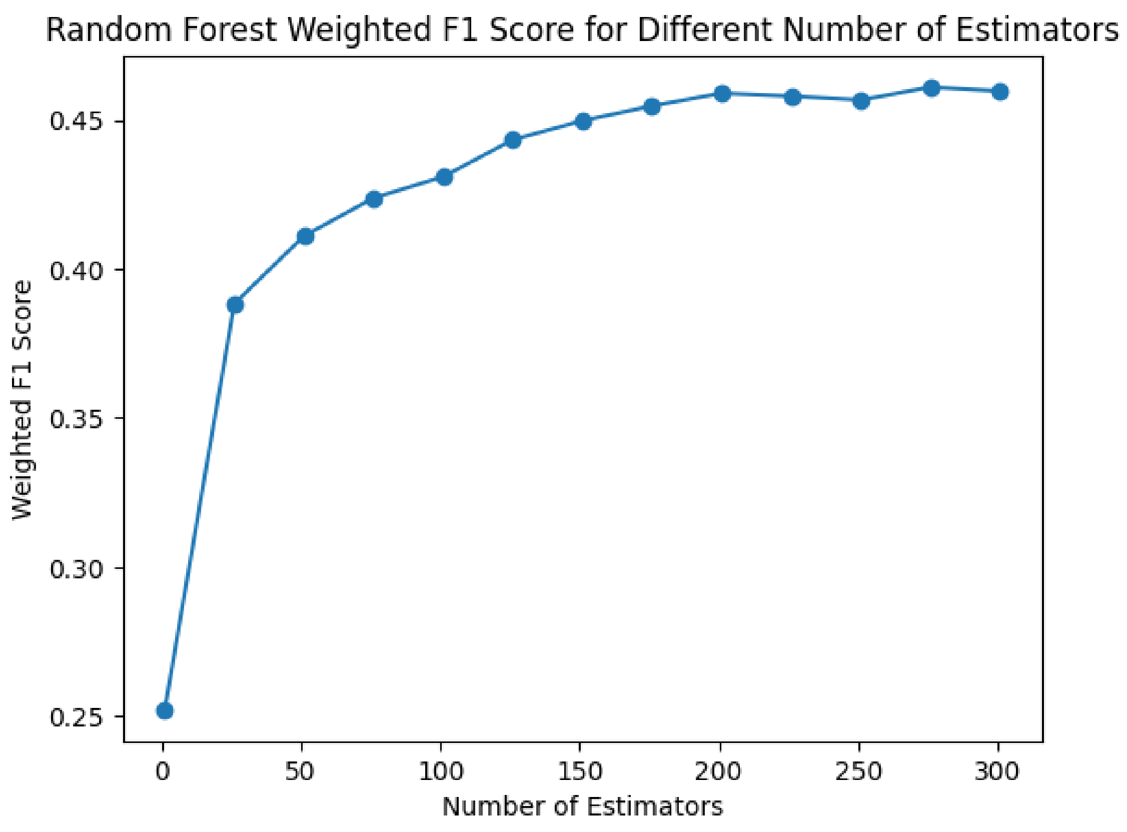


Figure 27. Weighted f1 Scores with respect to the number of estimators for SMOTE

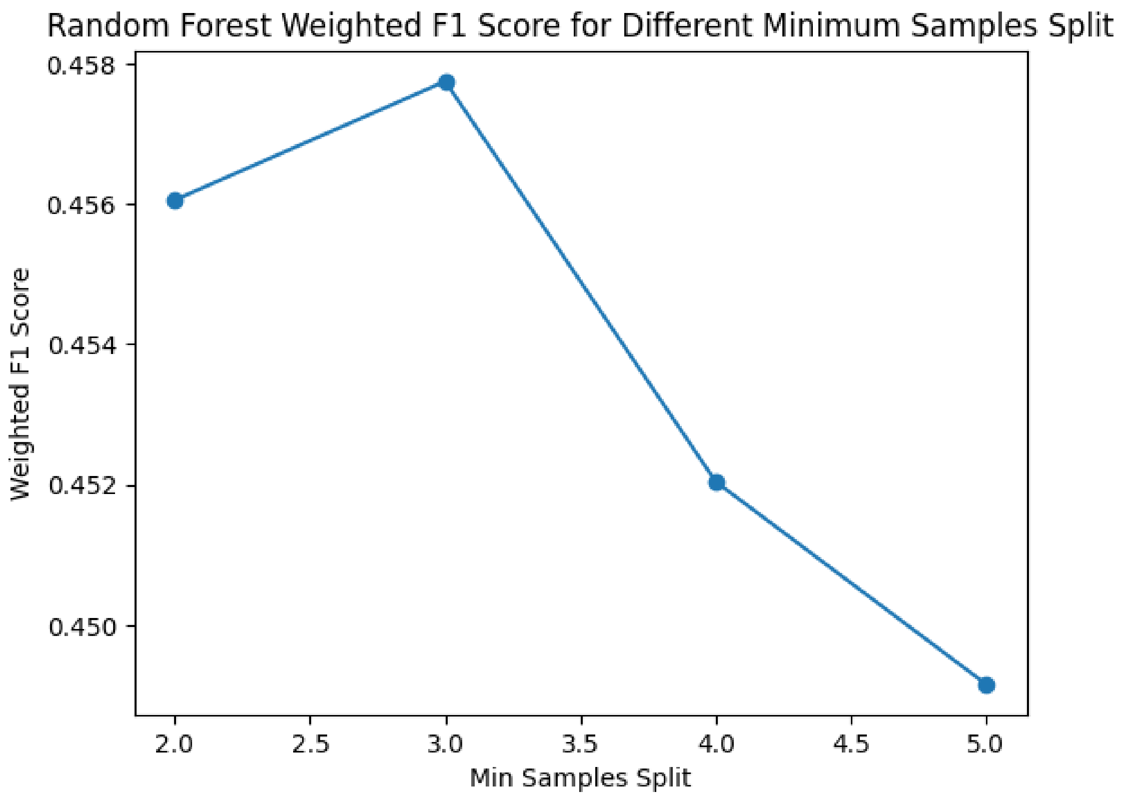


Figure 28. Weighted f1 Scores with respect to minimum samples split for SMOTE

Random Over Sampling

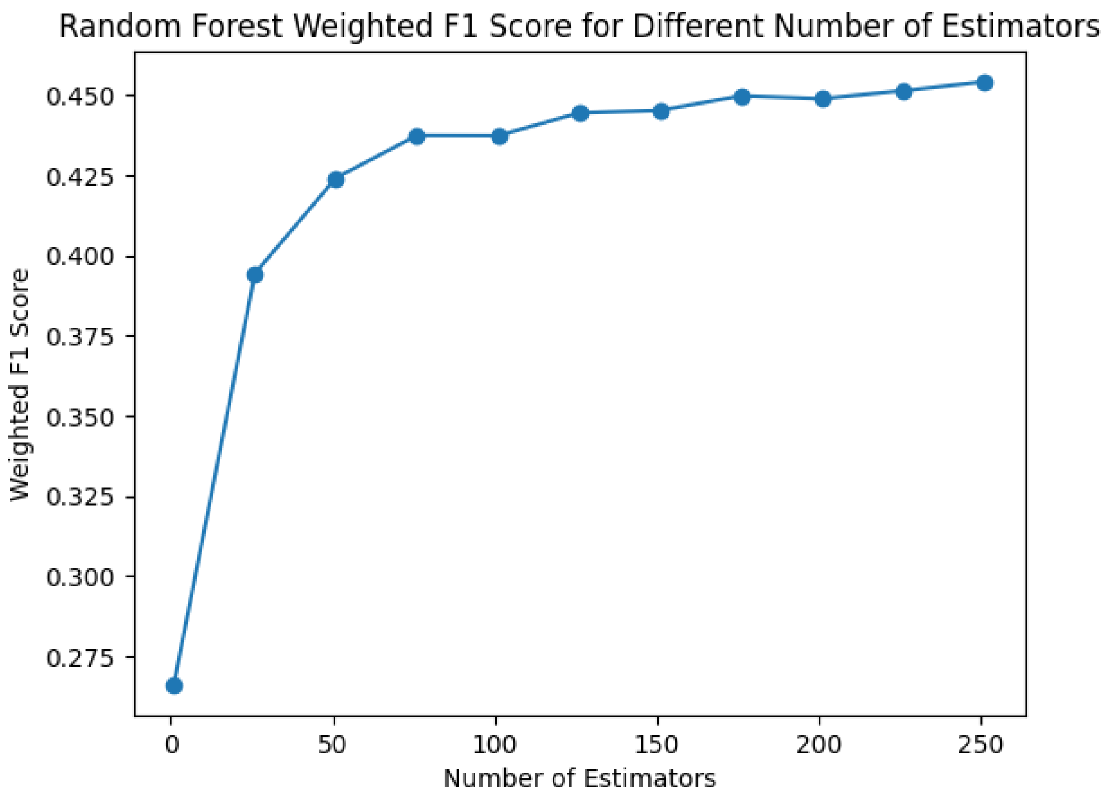


Figure 29. Weighted f1 Scores with respect to the number of estimators for Random Over Sampling

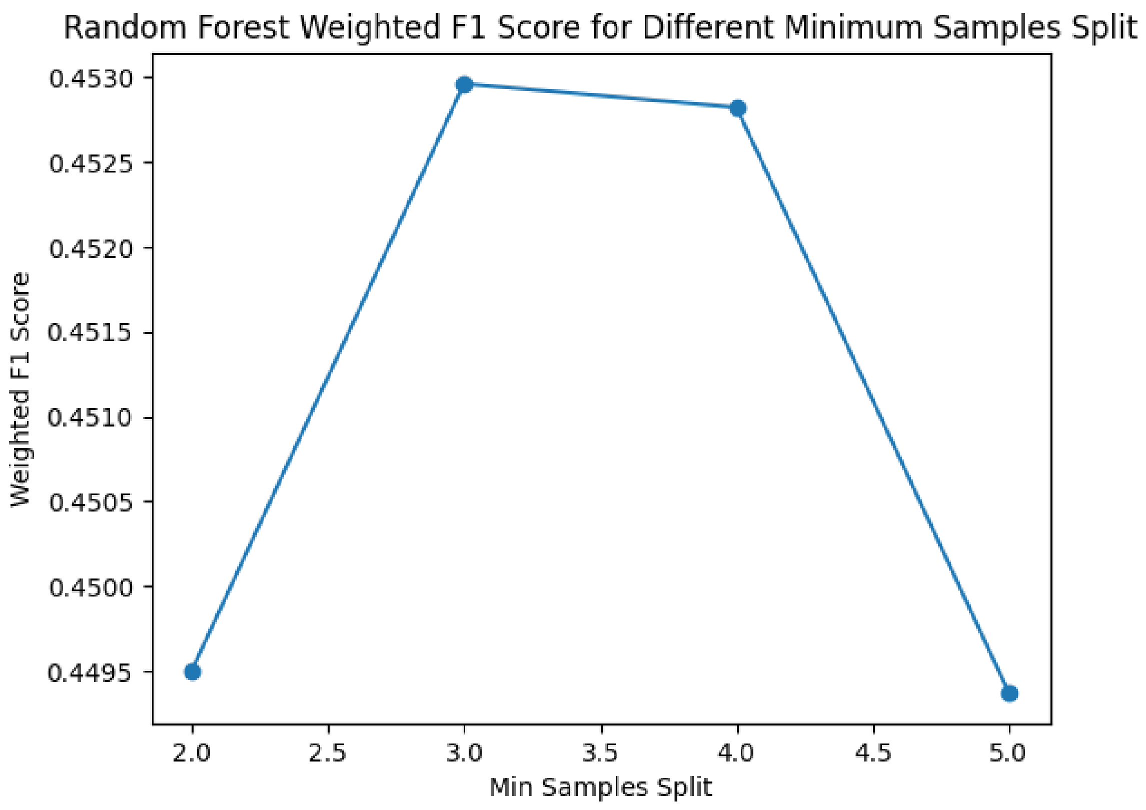


Figure 30. Weighted f1 Scores with respect to minimum samples split for Random Over Sampling

So, we get the best result in SMOTE for random forest classifier models with n_estimator equal to 175 and min_sample_split equal to 3. When the number of estimators was more than 175, the results did not significantly change, so the trade between running time and f1scores is not enough to wait. After observing this, we predict our test set using this model, and the results can be seen below.

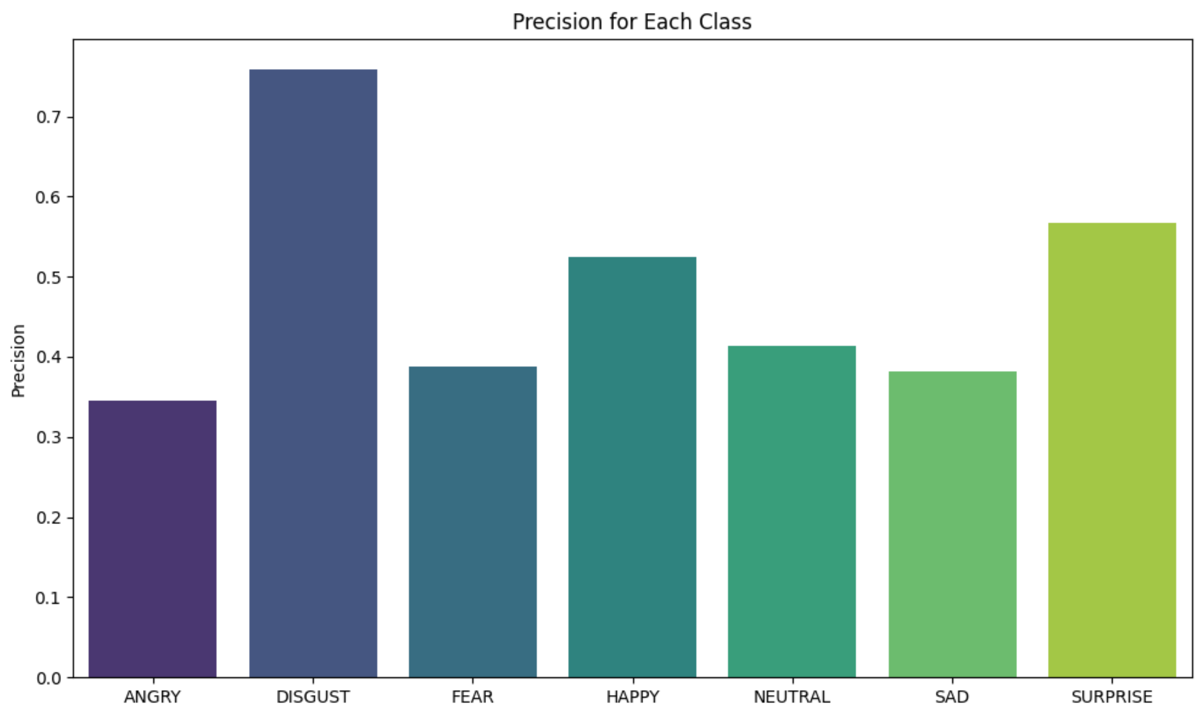


Figure 31. Precision for Each Class with Best Model

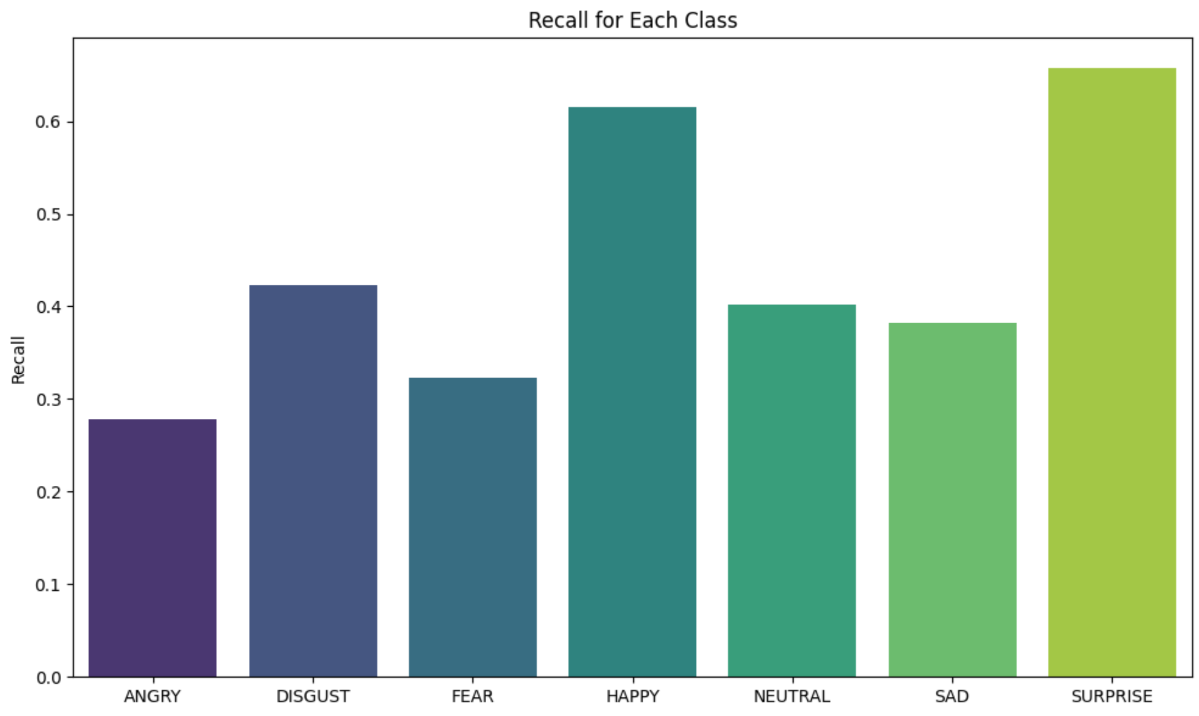


Figure 32. Recall for Each Class with Best Model

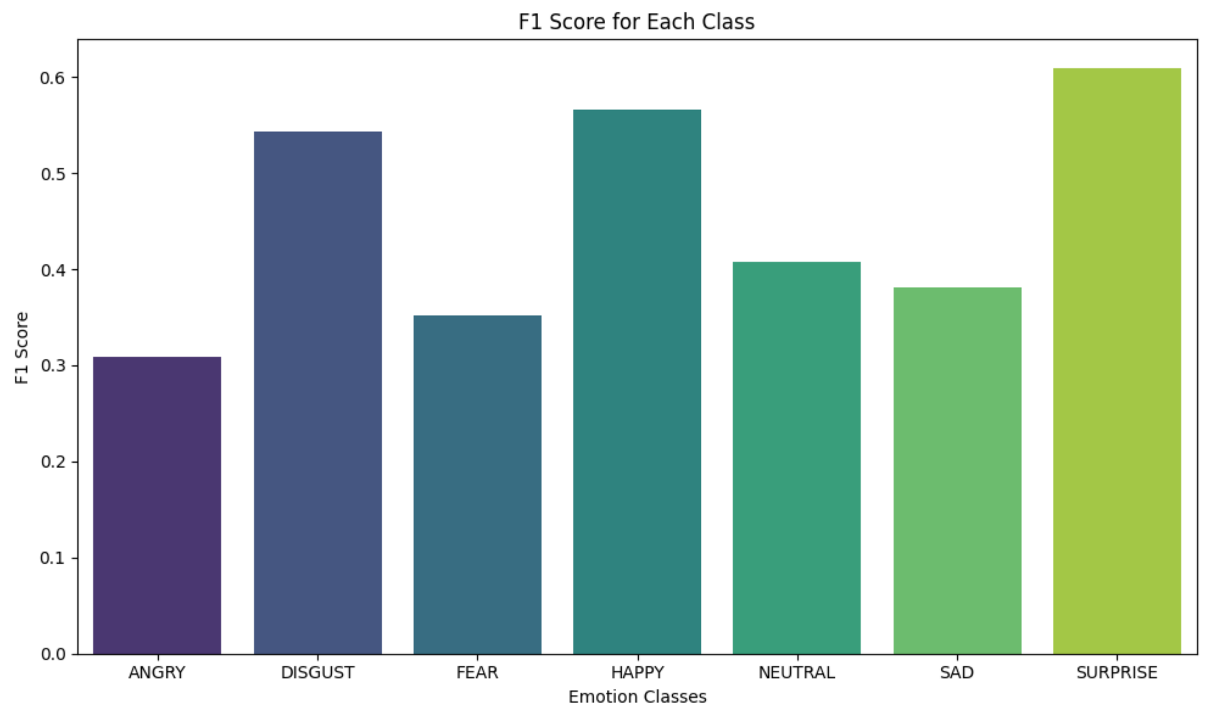
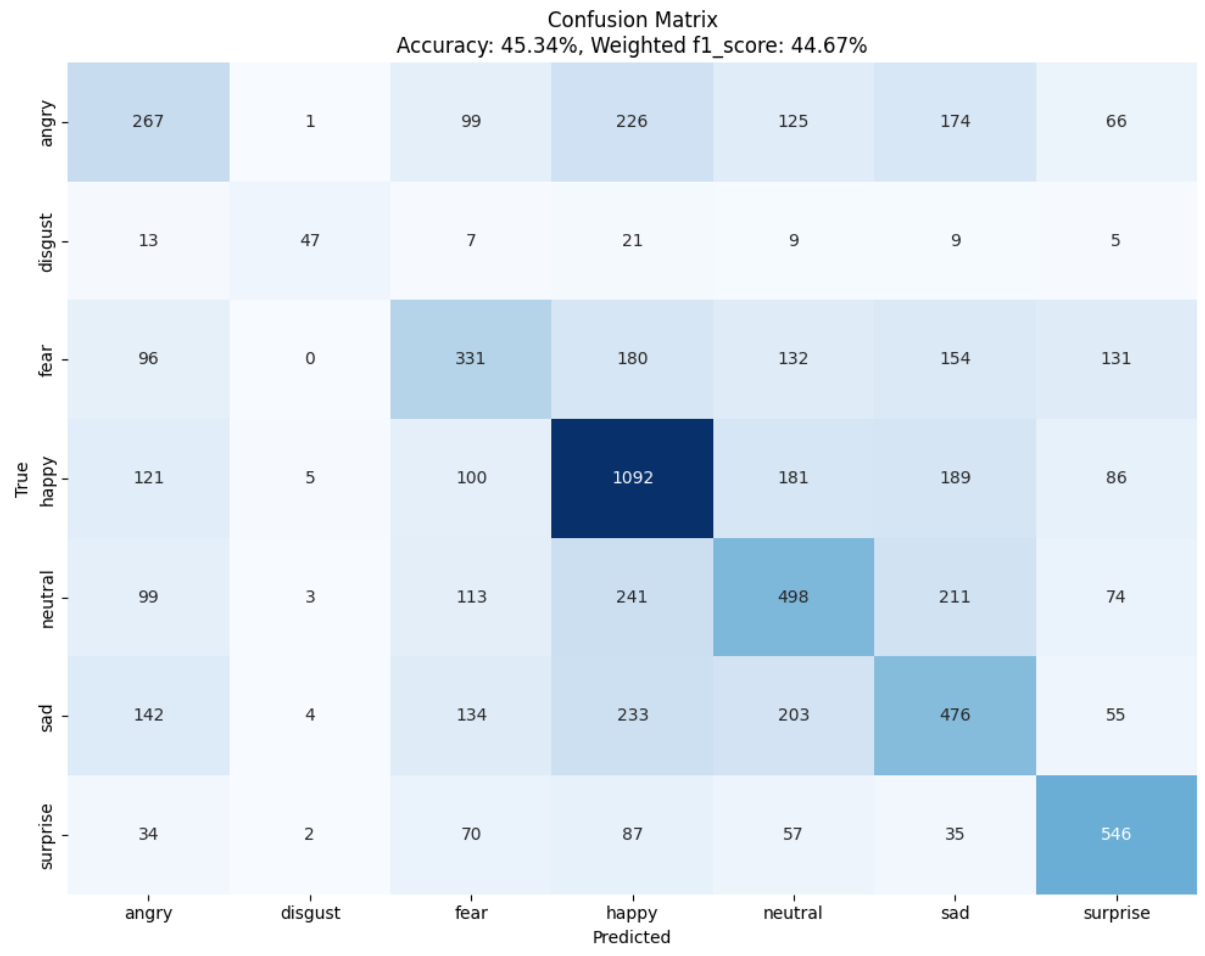


Figure 33. F1 Score for Each Class with Best Model



	angry	disgust	fear	happy	neutral	sad	surprise
angry	267	1	99	226	125	174	66
disgust	13	47	7	21	9	9	5
fear	96	0	331	180	132	154	131
happy	121	5	100	1092	181	189	86
neutral	99	3	113	241	498	211	74
sad	142	4	134	233	203	476	55
surprise	34	2	70	87	57	35	546

Figure 34. F1 Score, Accuracy, and Confusion Matrix with Best Model

3.5. Convolutional Neural Network (CNN)

	Layer	Parameter	Accuracy
CNN Model 1	4 Conv + 4 Dense	1006791	58.37%
CNN Model 2	2 Conv + 3 Dense	897095	55.21%
CNN Model 3	2 Conv + 3 Dense	243687	52.70%

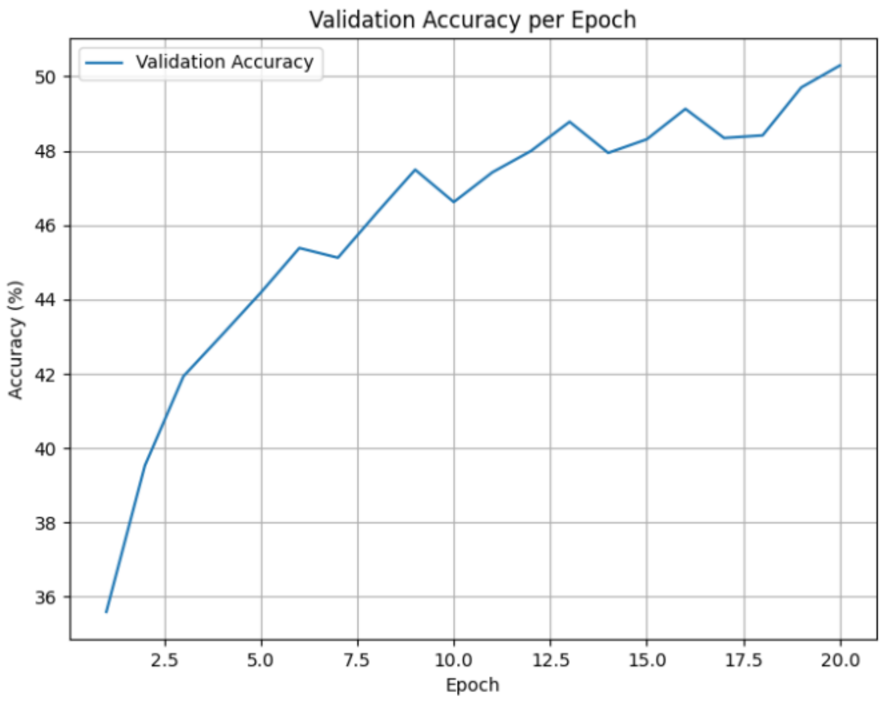


Figure 35: Validation Accuracy Per Epoch Unbalanced Data

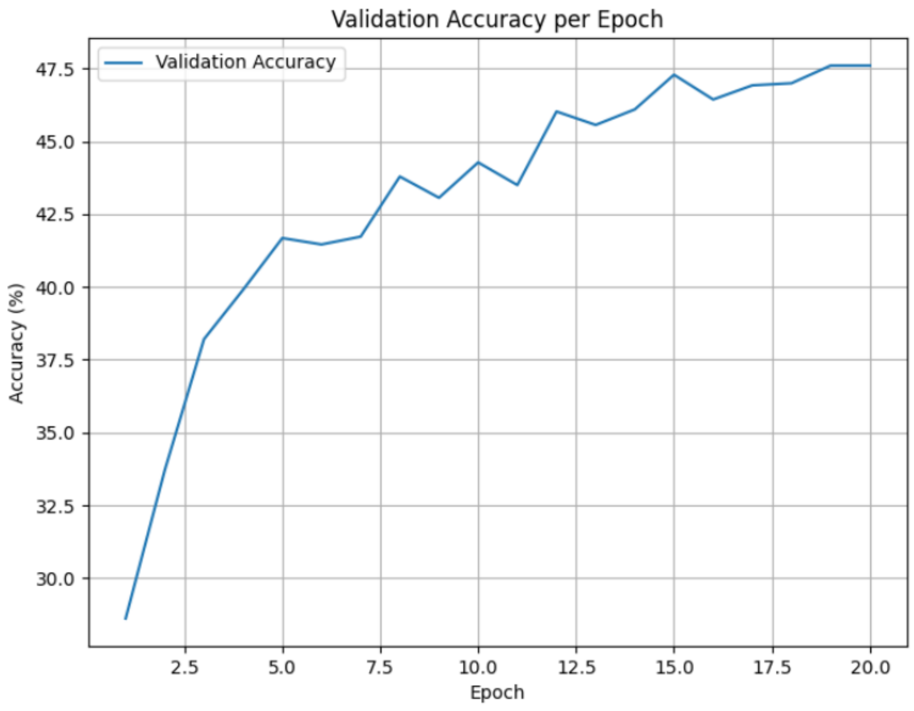
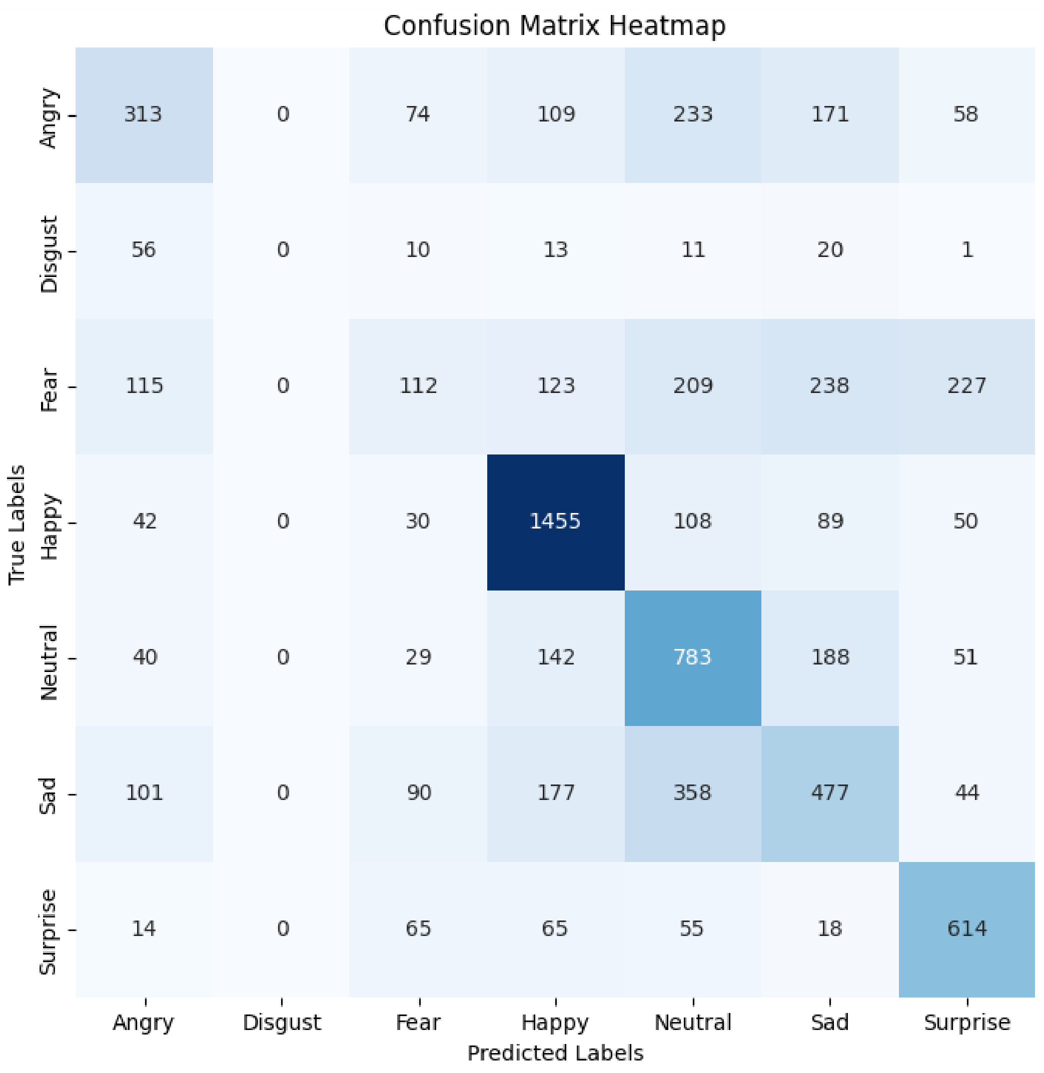


Figure 36: Validation Accuracy Per Epoch Balanced Data

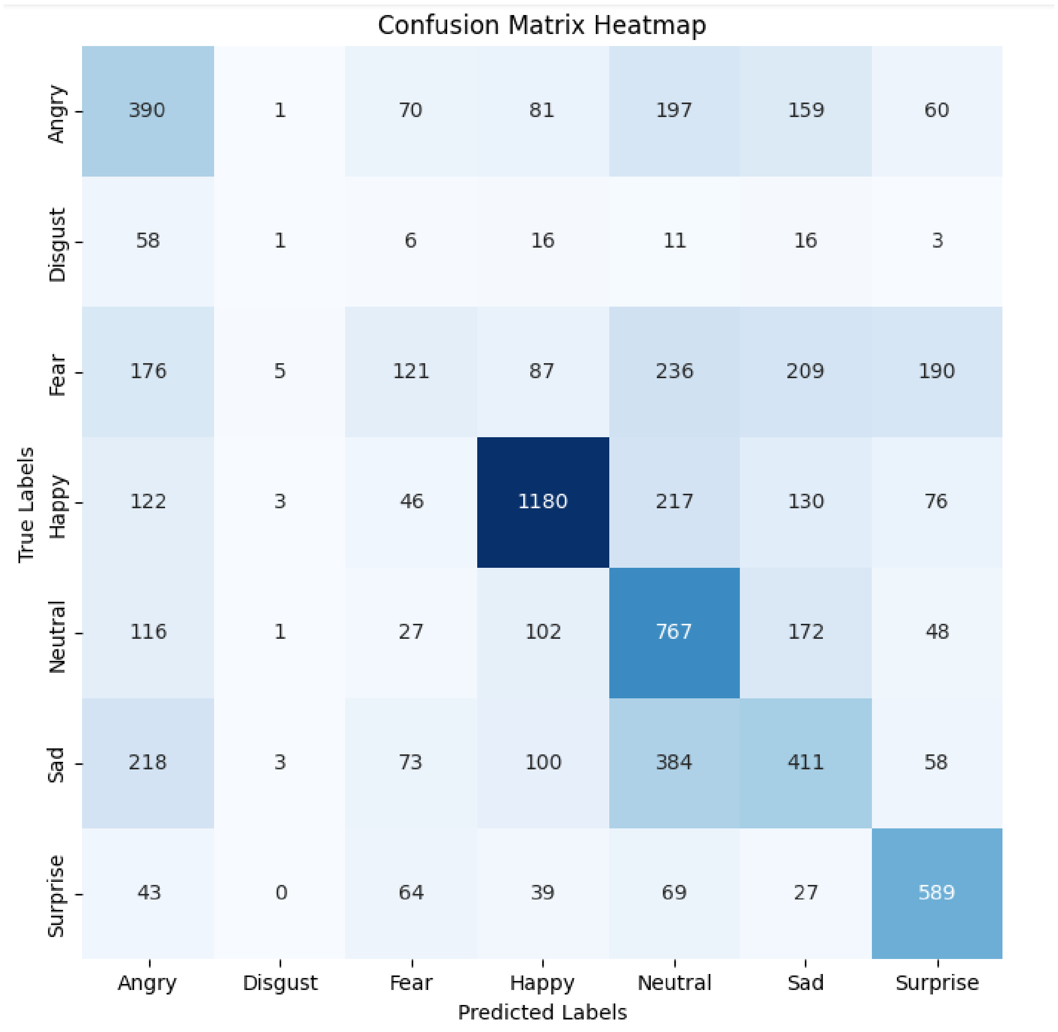
Accuracy	
Unbalanced	Balanced
52.70%	50.38%

F1 Score	
Unbalanced	Balanced
40.8%	39.0%



True \ Predicted	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Angry	313	0	74	109	233	171	58
Disgust	56	0	10	13	11	20	1
Fear	115	0	112	123	209	238	227
Happy	42	0	30	1455	108	89	50
Neutral	40	0	29	142	783	188	51
Sad	101	0	90	177	358	477	44
Surprise	14	0	65	65	55	18	614

Figure 37: Confusion Matrix Unbalanced Data



True \ Predicted	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Angry	390	1	70	81	197	159	60
Disgust	58	1	6	16	11	16	3
Fear	176	5	121	87	236	209	190
Happy	122	3	46	1180	217	130	76
Neutral	116	1	27	102	767	172	48
Sad	218	3	73	100	384	411	58
Surprise	43	0	64	39	69	27	589

Figure 38: Confusion Matrix Balanced Data

4. Discussion

4.1.Discussion About The Effect of Hyperparameters

4.1.1. K Nearest Neighbors (KNN)

In our study with the FER2013 dataset, the k-nearest neighbors (KNN) model's performance was modest, with the best accuracy reaching 34% at k=5. Any increase in k led to underfitting and a subsequent drop in accuracy. Preprocessing techniques, including Random OverSampling, Resampling, SMOTE, and Balanced Bagging Classification, did not significantly enhance the KNN's performance; the F1 score marginally improved to 32%.

The use of Principal Component Analysis (PCA) for dimensionality reduction further diminished accuracy, indicating the loss of crucial information for emotion recognition. These findings suggest KNN's limited suitability for this dataset, as it struggles with the subtleties of facial emotion classification. It may be prudent to consider more sophisticated models like Convolutional Neural Networks (CNN) for such complex tasks.

4.1.2. Decision Tree

In the preprocessing phase, we explored four distinct imputations and one additional classification method, leveraging the f1 score to assess their effectiveness in handling the skewed profile of the overall data as can be seen in Figure 17.

However, during the tuning process, accuracy was employed to evaluate and select optimal parameter combinations. Despite the versatility of decision trees, our findings indicate that they do not consistently yield high accuracies, as illustrated in Figures 18,19 and 20. Initially, with a basic model, we achieved 30% accuracy, which marginally improved to approximately 35%. It's worth noting that decision trees may not be the most suitable models for image classification tasks, primarily due to their limited ability to consider spatial relationships between pixels compared to more specialized models like Convolutional Neural Networks (CNN).

4.1.3. Random Forest

We leverage the f1 score to assess their effectiveness in handling the imbalance profile of the overall data, as can be seen in Figure 24.

During the tuning process, the f1 score was employed to evaluate and select optimal parameter combinations for the Random Forest model. Despite the versatility of decision trees, our findings indicate that they do not consistently yield high f1 scores, as illustrated in Figures 25 to 30. Upon replacing the decision tree with a Random Forest model, we observed a notable improvement. The f1 score increased from 30% to 40%, and the subsequent tuning process further enhanced the performance, reaching an f1 score of 45%. This highlights the effectiveness of Random Forests in capturing complex patterns within the data and addressing the limitations observed with basic decision trees.

These results underscore the importance of choosing an appropriate model and evaluation metric. In image classification, Random Forests, focusing on the f1 score, outperformed decision trees and demonstrated better suitability for handling spatial relationships in the data. However, it is not enough for our case.

4.1.4. Convolutional Neural Network (CNN)

The study evaluated various model architectures and data balancing techniques to understand their impact on image classification accuracy. Among the architectures tested, the most complex one with 1 million parameters exhibited the highest accuracy at 58%, while a significant reduction in complexity to around 200,000 parameters resulted in a modest decline to 52%, suggesting a relatively robust performance with decreased complexity. Surprisingly, data balancing through augmentation did not substantially enhance accuracy, as both balanced and unbalanced datasets yielded similar accuracies of around 50% and 52%, respectively. Moreover, the F1 metrics, measuring class-wise performance, showed consistent scores of approximately 39% for balanced data and 40% for unbalanced data. We used TensorFlow initially to perform CNN, but then, due to the huge gap between ‘model.evaluate’ and ‘model.predict’, we also implemented CNN with PyTorch to get a better confusion matrix and better F1 scores. Both implementations can be seen in zip file.

Despite these variations, the Convolutional Neural Network (CNN) emerged as the top-performing model, as anticipated, due to its inherent ability to effectively process spatial information in images through convolution and pooling layers, enabling it to learn intricate features efficiently. Ultimately, while model complexity and data balancing influenced performance to some extent, the CNN architecture demonstrated superior adaptability in handling image classification tasks, solidifying its effectiveness in this domain.

4.2. Comparison of Different Methods

Our investigation into facial emotion recognition on the FER2013 dataset entailed a comprehensive comparison of four distinct models: K Nearest Neighbors (KNN), Decision Tree, Random Forest, and Convolutional Neural Network (CNN).

The KNN model demonstrated a peak accuracy of 34% at k=5, which was the highest recorded for this model on the dataset. However, this performance was compromised when k was increased, leading to underfitting and a reduction in accuracy. Despite various data balancing techniques such as Random OverSampling, Resampling, SMOTE, and Balanced Bagging Classification, the performance improvement was marginal, with the F1 score increasing only to 32%. The application of PCA further reduced accuracy, indicating KNN's limitations in capturing the nuanced features necessary for accurate facial emotion recognition.

In contrast, the Decision Tree and Random Forest models showed different levels of improvement with preprocessing adjustments. The Decision Tree's initial accuracy of 30% improved slightly to 35% with tuning, but the model's inability to consider spatial relationships between pixels limited its performance in image classification tasks. The Random Forest model, with its ensemble approach, improved upon the Decision Tree's foundation, increasing the F1 score from 30% to 45% after tuning. This improvement reflects the Random Forest's capacity to handle more complex data patterns, but it still fell short of the benchmark necessary for our dataset.

The CNN model outshone the others, affirming its suitability for image classification tasks. With its deep learning capabilities, it achieved an accuracy of 58% with a more complex architecture, and a reduction in complexity to around 200,000 parameters saw only a slight decline in accuracy to 52%. Data balancing techniques did not significantly impact CNN's performance, which remained robust across balanced and unbalanced datasets. The transition from TensorFlow to PyTorch for CNN implementation provided a more reliable confusion matrix and F1 scores, highlighting the importance of the right tools in addition to the right model.

The comparison underlines that while KNN and tree-based models may have their merits in certain applications, CNNs, with their deep architectures designed for handling spatial information and feature extraction, are more adept for complex image classification tasks like facial emotion recognition. The results also suggest that while data balancing and preprocessing are important, the choice of model plays a crucial role in achieving optimal performance in image-based classification tasks.

5. Conclusions

Our exploration of facial emotion recognition using the FER2013 dataset has been an insightful journey through four distinct machine learning and deep learning models, each with its unique approach. This project provided us with a foundational understanding of various machine learning paradigms, from the simplicity of K Nearest Neighbors (KNN) to the complexity of Convolutional Neural Networks (CNN).

KNN demonstrated limited success, achieving an accuracy peak of 34%. The Decision Tree and Random Forest models showed moderate improvements, with the latter slightly outperforming the former. However, CNN stood out as the most effective, achieving a high accuracy of 58% and showcasing its aptitude for spatial information processing and feature extraction.

This study has not only highlighted the importance of choosing the right model for specific tasks but also the role of different preprocessing techniques. It emphasizes that while some models may excel in one task, they might not be as effective in others.

As for future work, applying these four models to different image classification tasks using various datasets could further validate the findings of this project. This will help determine if the results are specific to the FER2013 dataset or if they are generalizable across different contexts. Additionally, expanding the comparison to other areas beyond image classification could offer deeper insights into the strengths and limitations of these models in diverse machine learning applications. This broadened exploration would contribute significantly to understanding the adaptability and efficacy of these models in various machine learning tasks.

6. Division of Work Among Group Members

Introduction and background information, applying SMOTE and Random OverSampling Hazal Buluş, Data augmentation Kürşad Güzelkaya, kNN Süleyman Gökhan Tekin, Decision Tree Bahadır Yüzlü, Random Forest Mustafa Mert Türkmen, Convolutional Neural Network Kürşad Güzelkaya.

All coding for these four models was done separately from Google Colab, but we gave feedback to each other's codes.

References

[1] S. Manas, “FER-2013,” Kaggle, 19-Jul-2020. [Online]. Available: https://www.kaggle.com/datasets/msambare/fer2013 [Accessed: 24-Nov-2023].

[2] L. Breiman, ‘Random Forests’, Machine Learning, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[3] “K-Nearest Neighbors (KNN) Python Examples,” Analytics Yogi. [Online]. Available: https://vitalflux.com/k-nearest-neighbors-explained-with-python-examples/. [Accessed: 24-Nov-2023].

[4] A. Dertat, Applied Deep Learning - Part 4: Convolutional Neural Networks, towardsdatascience.com, Nov. 8, 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2. [Accessed: 24-Nov-2023]

[5] L. Rokach and O. Maimon, "Decision Trees," 10.1007/0-387-25465-X_9, 2005. Available: https://www.researchgate.net/publication/225237661_Decision_Trees. [Accessed: 24-Nov-2023]

[6] R. Janković, "Classifying Cultural Heritage Images by Using Decision Tree Classifiers in WEKA (short paper)," in VIPERC@IRCDL, 2019. Available: https://api.semanticscholar.org/CorpusID:67866102.