FACIAL EXPRESSION RECOGNITION

Vũ Tùng Linh 20210523
Tạ Quang Duy 20214884
Đào Hà Xuân Mai 20210562
Nguyễn Việt Dũng 20214883

ABSTRACT

Recognizing facial expressions accurately poses a significant challenge in computer vision and artificial intelligence. Interpreting emotional states solely based on facial cues requires overcoming variations in lighting conditions, occlusions, and individual differences. By utilizing multiple traditional Machine Learning methods, we can achieve a commendable level of accuracy in this scientific field.

1. Introduction:

Facial expression recognition enables more intuitive human-machine interactions and has applications in diverse domains such as psychology and human-computer interaction, which brings a huge advantage to numerous fields such as health care, education... etc.

In this report, we investigate the problem of improving prediction accuracy through the application of various advanced methods. Specifically, we explore the implementation of Random Forest (RF), K-Nearest Neighbors (KNN), and XGBoost Classifier as individual classifiers.

To further enhance accuracy, we utilize ensemble learning algorithms: Bagging and boosting (AdaBoost) techniques. These approaches aim to reduce overfitting and combine diverse predictions for improved performance. Through empirical evaluations, we assess the effectiveness of these methods in achieving higher accuracy.

2. Data Prepareation:

2.1 Dataset Introduction: The FER2013 dataset

The FER2013 dataset is a widely used dataset in the field of facial expression recognition. It contains grayscale images of facial expressions captured from various sources, including movies, YouTube videos, and Google Images.

The dataset consists of seven different expression classes: anger, disgust, fear, happiness, sadness, surprise, and neutral. Each image is labeled with one of these emotion categories, representing the dominant expression observed.

In terms of size, the FER2013 dataset comprises approximately 35,887 images. These images are divided into three subsets: a training set containing around 28,709 images, a public test set with approximately 3,589 images, and a private test set consisting of roughly 3,589 images as well.

Researchers often utilize the FER2013 dataset to train and evaluate facial expression recognition models due to its diversity of expression classes and substantial number of labeled samples.

2.2 Feature Engineering:

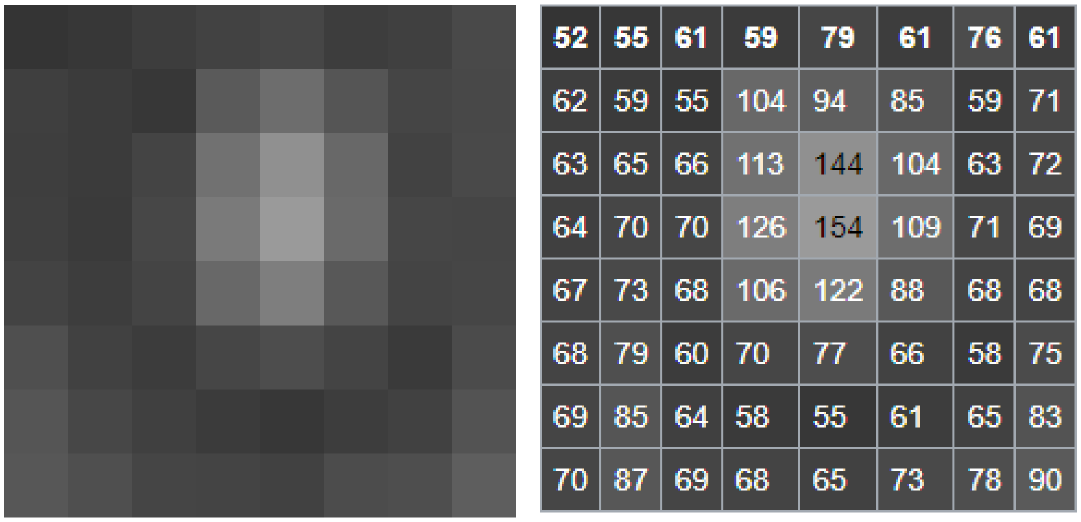
2.2.1 Histogram Equalizer:

Histogram equalization is a technique used to enhance the contrast and improve the visual quality of an image. It works by redistributing the intensity values in the image's histogram, aiming to achieve a more uniform distribution. This process adjusts the pixel intensities to span the entire range from the minimum to the maximum value.

The method is useful in images with backgrounds and foregrounds that are both bright or both dark (in our case grayscale images). In particular, the method can lead to better detail in photographs that are either over or under-exposed.

For example, we try the technique on an 8x8 sub-image

1. *Compute the histogram:*
    - Let H(i) represent the histogram bin count for intensity level i.
    - Iterate over all pixels in the image and increment the corresponding histogram bin.



52	55	61	59	79	61	76	61
62	59	55	104	94	85	59	71
63	65	66	113	144	104	63	72
64	70	70	126	154	109	71	69
67	73	68	106	122	88	68	68
68	79	60	70	77	66	58	75
69	85	64	58	55	61	65	83
70	87	69	68	65	73	78	90

Figure 1. The 8×8 sub-image shown in 8-bit grayscale [1]

2. *Calculate the cumulative distribution function (CDF):*

- Normalize the histogram values by dividing each value by the total number of pixels in the image, K, to obtain the probability distribution function (PDF):$\mathrm{P(i)} = \frac{\mathrm{H(i)}}{K}$.
- Calculate the cumulative distribution function (CDF) as the sum of probabilities up to each intensity level: $\mathrm{cdf}(i) = \sum_{j=0}^{i} P(i)$

Pixel Intensity v	cdf(v)	Equalized v h(v)	Pixel Intensity v	cdf(v)	Equalized v	Pixel Intensity v	cdf(v)	Equalized v h(v)
52	1	0	69	33	130	88	53	210
55	4	12	70	37	146	90	54	215
58	6	20	71	39	154	94	55	219
59	9	32	72	40	158	104	57	227
60	10	36	73	42	166	106	58	231
61	14	53	75	43	170	109	59	235
62	15	57	76	44	174	113	60	239
63	17	65	77	45	178	122	61	243
64	19	73	78	46	182	126	62	247
65	22	85	79	48	190	144	63	251
66	24	93	83	49	194	154	64	255
67	25	97	85	51	202			
68	30	117	87	52	206			

Table 1. Cumulative distribution and their equalized value of pixel intensity

3. *Normalize the CDF:*

- The normalized CDF maps the intensity levels to the desired output range [0, L-1], where L represents the maximum intensity level (such as 256 for an 8-bit image).
- Calculate the equalized CDF values h(v):

$$\mathrm{h}(v) = \mathrm{round}\left(\frac{cdf(v) - cdf_{min}}{(MxN) - cdf_{min}} * (L-1)\right)$$

Where: M, N are width and height of the image respectively.

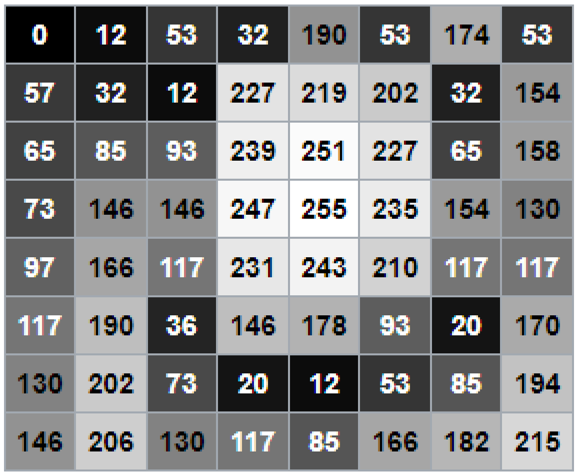
$\mathrm{cdf}_{\mathrm{min}}$ is the cdf(v) with the smallest value.

4. *Create a mapping function:*

- Generate a mapping function that maps the original intensity values to their equalized values using the equalized CDF: Mapping(i) = h(v)(i).

5. *Apply the mapping function to each pixel:*

- Iterate over all pixels in the image and replace their original intensity value with the corresponding equalized value: EqualizedPixel(i, j) = Mapping(Image(i, j)).



0	12	53	32	190	53	174	53
57	32	12	227	219	202	32	154
65	85	93	239	251	227	65	158
73	146	146	247	255	235	154	130
97	166	117	231	243	210	117	117
117	190	36	146	178	93	20	170
130	202	73	20	12	53	85	194
146	206	130	117	85	166	182	215

Figure 2. Equalized images after mappping

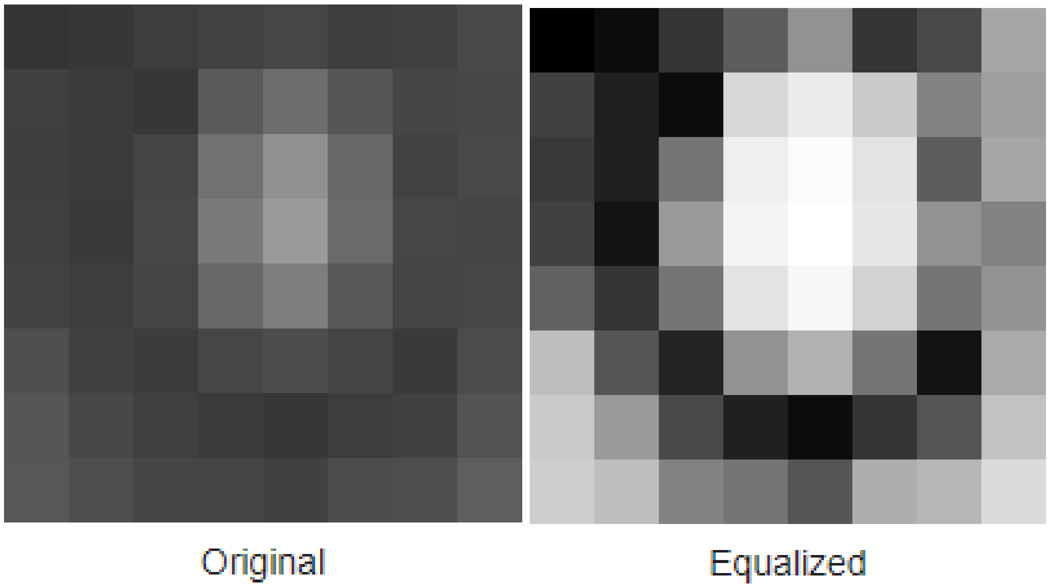


Figure 3. Image before and after histogram equalized

The result of the histogram equalization process is an image with improved contrast and a more uniform distribution of intensity values across the entire range. By redistributing the intensities, details and features that were previously hidden or obscured can become more apparent.

2.2.2 Local Binary Patterns:

Local Binary Pattern (LBP) : A texture descriptor algorithm widely used in computer vision and image processing. It was introduced by Ojala, Pietikäinen, and Harwood in 1994 as a means to characterize local textures within an image.

The core concept of LBP lies in encoding the relationship between a central pixel and its surrounding neighbor pixels by comparing their intensity values. This comparison involves treating each neighbor pixel as a binary digit (0 or 1) based on whether its intensity value is lower or higher than the intensity value of the central pixel. These binary digits are then concatenated to form a binary pattern.

To perform the LBP operation, the following steps are followed for each pixel in the image:

1. Select a pixel as the center.
2. Define a neighborhood around the central pixel with a fixed size.
3. Compare the intensity value of each neighboring pixel with that of the central pixel.
4. Assign a binary value of 1 if the neighbor's intensity value is smaller than or equal to the central pixel's intensity value; otherwise, assign a value of 0.
5. Concatenate the obtained binary values in a specific order (clockwise or counterclockwise) to form a binary pattern.
6. Convert the binary pattern to decimal to obtain the final LBP value for that pixel.

To get a better understanding of the algorithm, we add an example:

6	6	2
7	5	4
8	1	4

We use the figure above as a 3x3 neighborhood of pixels with the center pixel has the value 5.

After Step 3 and 4, we obtain:

0	0	1
0		1

0	1	1

If we choose the clockwise order, and start from the top-right corner, we obtain its binary paattern:

1	1	1	1	0	0	0	0

Finally, we convert the pattern above into decimal presentation to obtain new center pixel value:

$$2^0 + 2^1 + 2^2 + 2^3 + 0^4 + 0^5 + 0^6 + 0^7 = 15$$

After obtaining new center value, we add it to the output LBP image:

	15	

We apply this for every other neighborhood of pixel figures possible.

By applying the LBP operation to every pixel in an image, we can generate a new image where each pixel represents the LBP value calculated for the corresponding pixel in the original image.

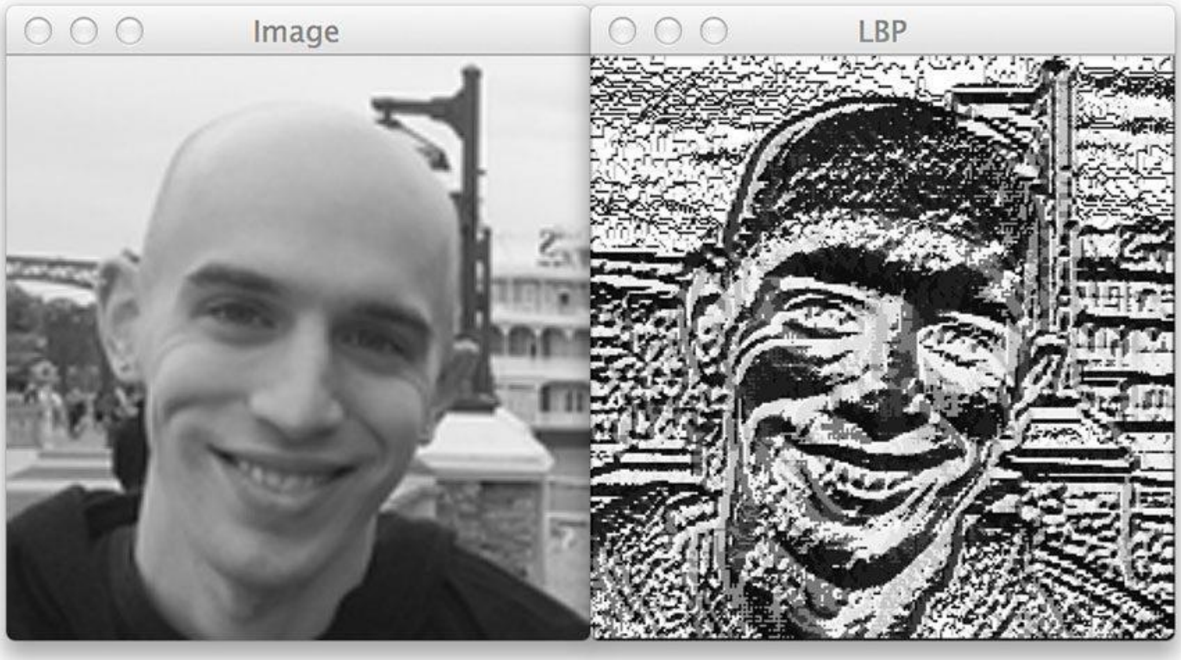


Figure 4. Image before and after LBP operation

LBP offers several advantages, including its simplicity, computational efficiency, and robustness against common image transformations like rotation, scale changes, and grayscale variations. This technique offered great help in picture preprocessing, especially for our dataset which is full of grayscale images.

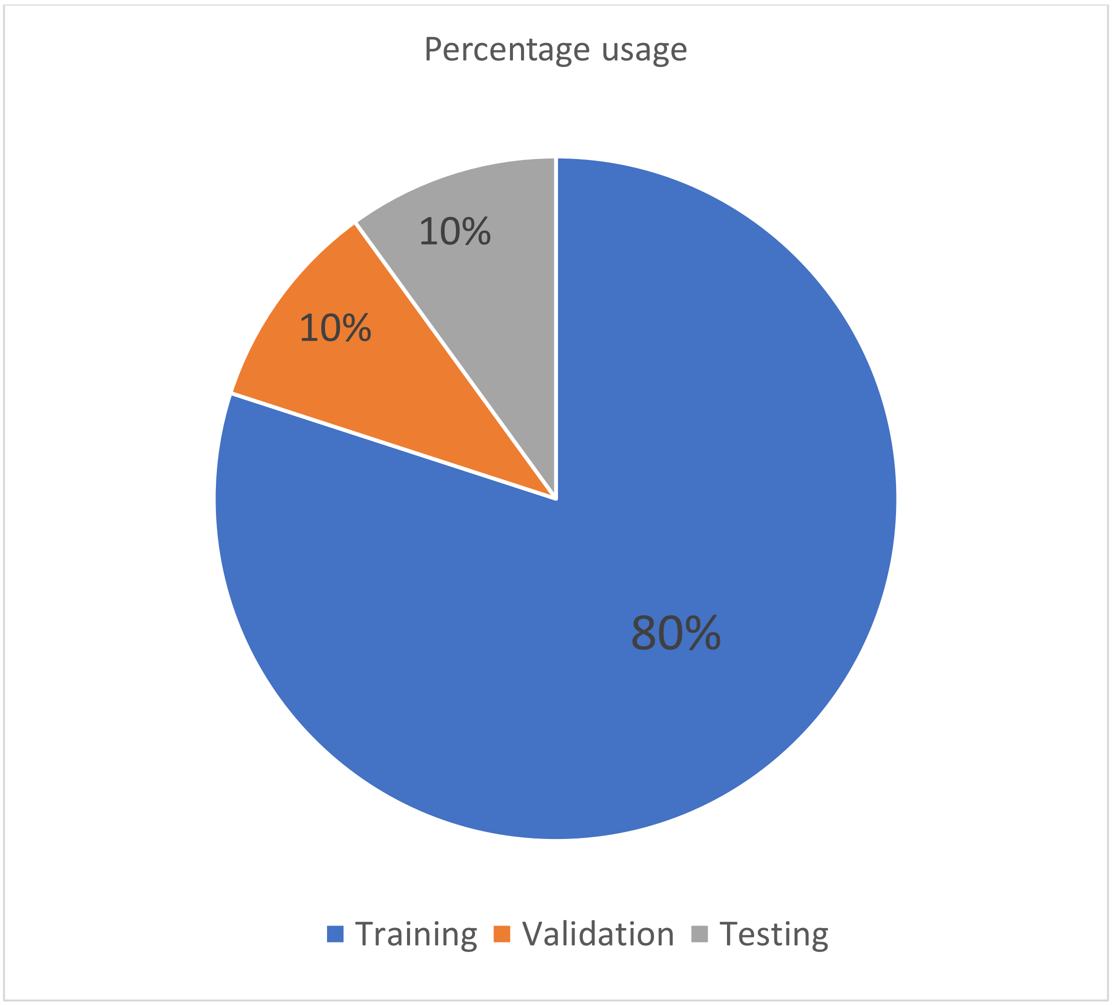
2.3 Data Preprocessing:

We execute Data Preprocessing with the following steps:

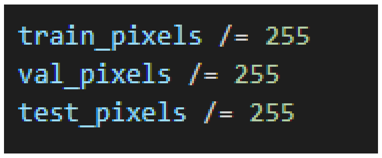
1. Read and extract the FER2013 dataset from the fer2013.csv file.
2. Split the data into three sets: training, validation, and testing.

   The FER2013 data set is divided into 2 parts: Public (80%) and Private (20%)

1. We use the Public part for the whole training purpose, while the Private part, half for validation, half for testing. The training set is used to train the model, while the validation set is used to evaluate its performance with different hyperparameter combinations, and the testing set is to assess the model's performance on completely unseen data, providing an unbiased evaluation. The test dataset won’t be used during the model development phase to avoid overfitting.



3. Convert pixel values to numpy arrays and normalize them to the range [0, 1].



```
train_pixels /= 255
val_pixels /= 255
test_pixels /= 255
```

4. We expand the training set using the existing images by augmenting them with multiple methods:

   - HorizontalFlip: Flip the image horizontally with a probability of p=0.5.
   - RandomCrop: Randomly crop a portion of the image with a size of height=48, width=48, and a probability of p=0.1.
   - Rotate: Randomly rotate the image by an angle between -8 to 8 degrees (limit=8), with a border value of 0 (border_mode=cv2.BORDER_CONSTANT), and a probability of p=0.5.

- GaussianBlur: Blur the image using a Gaussian filter with a random blur radius between 3 to 7 pixels (blur_limit=(3,7)), a random standard deviation between 0 to 0.5 (sigma_limit=(0,0.5)), and a probability of p=0.5.

This way, we can create new training samples which are slightly different from the original datas.

5. Store the preprocessed data, including the original and augmented images, into npy files for future use.

3. Method Explanation:

Before diving into the specific details of our approach, let's briefly discuss ensemble learning technique and grid search technique.

Ensemble learning: A technique in machine learning where multiple models are combined to make predictions or decisions. Rather than relying on a single model, ensemble learning leverages the collective wisdom of multiple models to improve overall performance and accuracy.

Each individual model in the ensemble may have its own strengths and weaknesses, but by combining their predictions or decisions, the ensemble can often achieve better results than any single model alone. Ensemble learning methods include techniques like Bagging, Boosting, and Stacking, which help to mitigate biases, reduce overfitting, and enhance generalization capabilities. In this report, we apply both Bagging and Boosting for Random Forest.

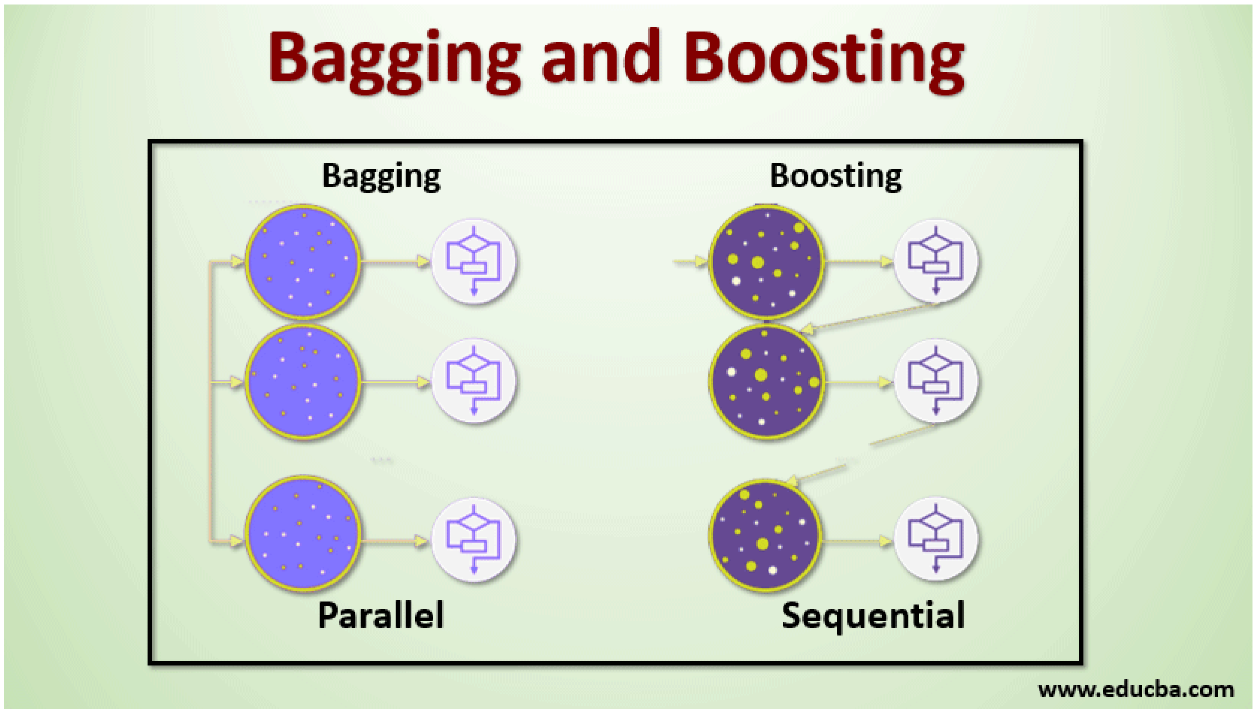


Figure 5. Bagging and Boosting

Grid search: A technique used in machine learning to systematically search for the optimal combination of hyperparameter values for a given model. Hyperparameters are settings or configurations that are not learned from the training data but need to be specified by the user before training the model.

Grid search works by systematically searching through a predefined grid of hyperparameter values to find the optimal combination for a given machine learning model.

In this report. Grid search systematically explores hyperparameter combinations using k-fold cross-validation with n_folds = 5 and prioritizes the F1-score for evaluation and selection.

Grid search's operation include:

1. Define the parameter grid: Specify the hyperparameters to be tuned and the possible values for each hyperparameter.
2. Create the training and validation sets: Split the available dataset into training and validation sets. To ensure reliable evaluation, use k-fold cross-validation, which divides the data into k equally sized folds and iteratively uses each fold as the validation set while training on the remaining data.
3. Training and evaluation loop: Iterate over all the hyperparameter combinations in the grid. For each combination:

   a. Train the model: Build and train the machine learning model using the current hyperparameter values on the training set.

   b. Evaluate the model: Use the trained model to make predictions on the validation set and calculate F1-score.

   c. Store the results: Keep track of the evaluation metric(s) obtained for each hyperparameter combination.
4. Select the best hyperparameters: After evaluating all the combinations, select the combination that yielded the highest or most desirable evaluation metric(s). This combination represents the optimal hyperparameter values for the model.
5. Optional step: Retrain the model: Once the best hyperparameters are identified, the final step may involve retraining the model using the entire dataset (including the validation set) and the chosen hyperparameters to obtain the final model.

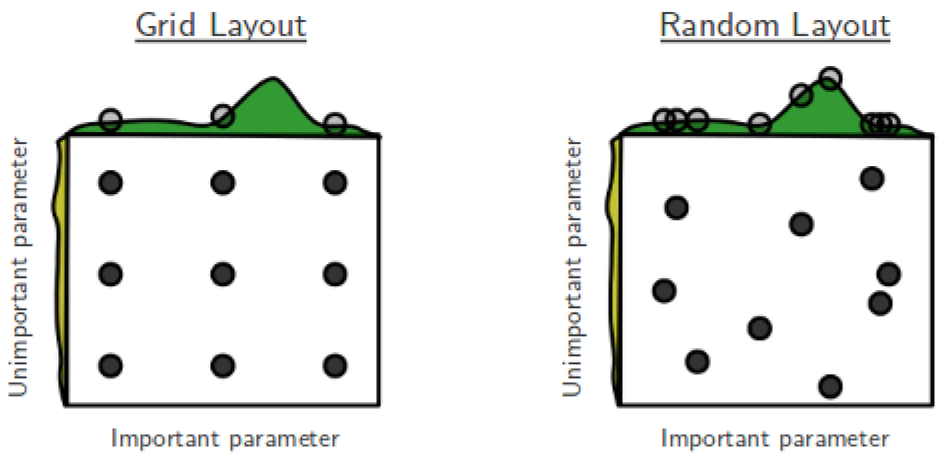


Figure 6. Grid search and random search

3.1 Random Forest :

Random Forest is a popular machine learning algorithm that can be applied to facial expression recognition tasks.

3.1.1 Bagging:

Bagging, or **Bootstrap Aggregation**, is an ensemble technique commonly employed by the random forest algorithm.

In bagging, a random subset is selected from the complete dataset. Consequently, each model is created using bootstrap samples, which are obtained by sampling with replacement from the original data. The process of sampling rows with replacement is referred to as ***bootstrapping***.

Subsequently, each model is trained independently, resulting in individual results. The final output is determined through **aggregation** step involves merging all the outcomes and producing the output based on the majority vote.

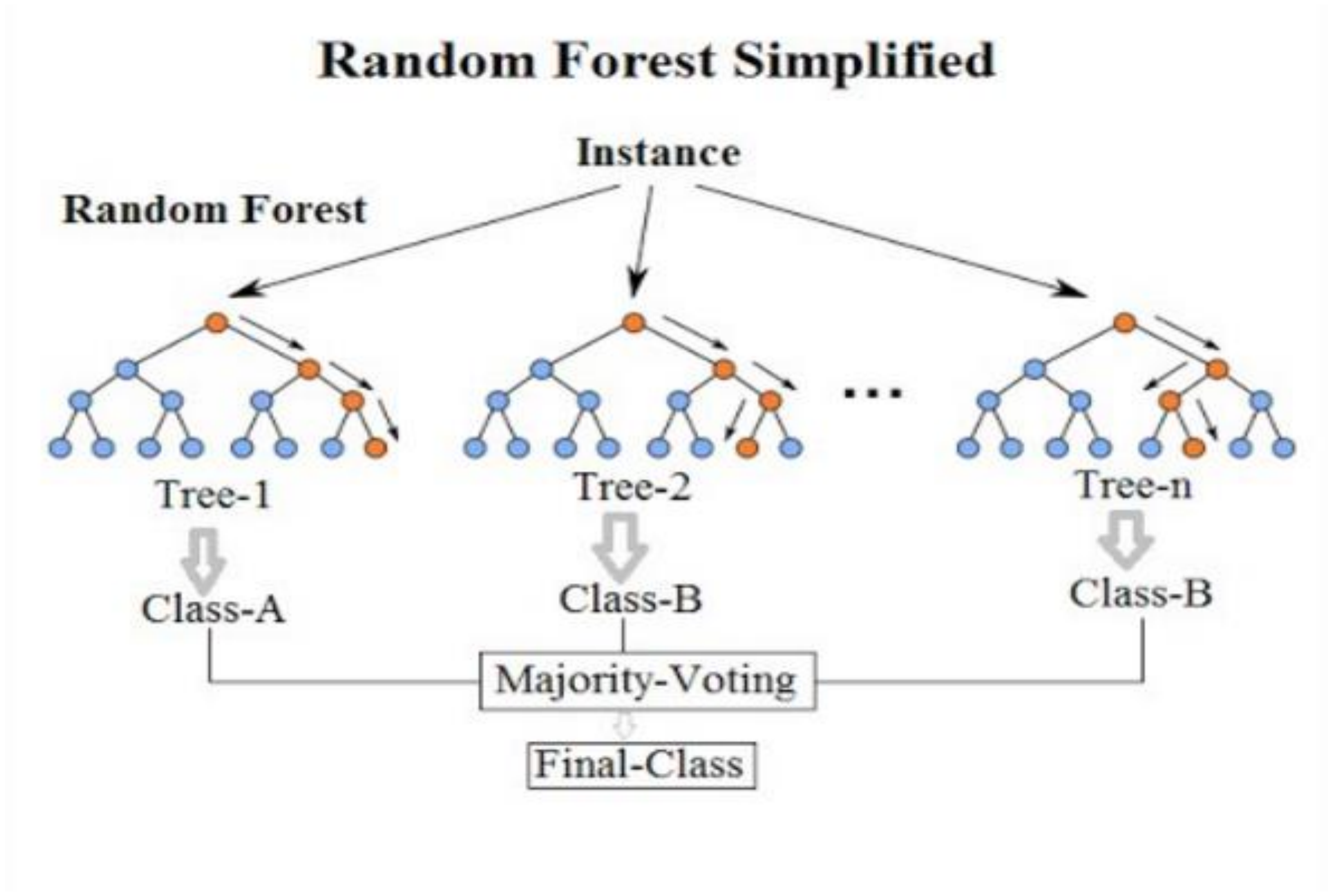


Figure 7. Random Forest with Bagging

3.1.2 Grid Search:

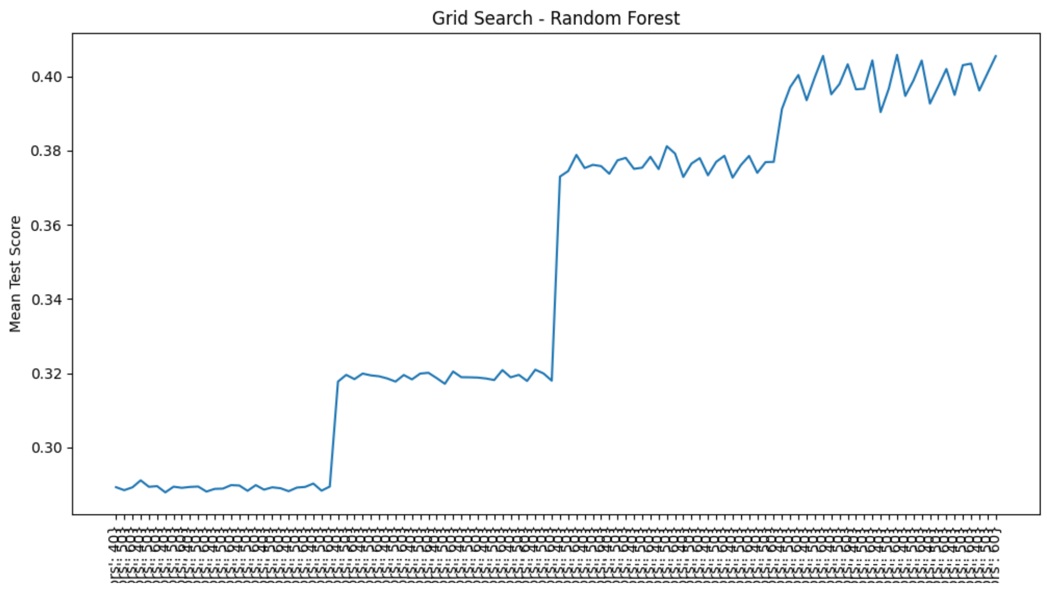
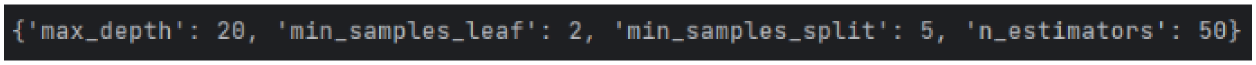


Figure 8. Grid search result of Random Forest.

After Grid Search, we conclude that the best parameters are:



```
{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}
```

3.1.3 Procedure:

We execute Random Forest:

1. Feature extraction: Extract relevant features from the facial images.
2. Training the Random Forest: Train a Random Forest classifier using the extracted features. Each decision tree in the forest will be trained on a randomly selected subset of the training data and a random subset of features. The trees are trained to predict the correct label (identity) of a given facial image based on its features.
3. Ensemble prediction: During inference, each decision tree in the Random Forest makes a prediction for a given facial image. The final prediction is determined by aggregating the individual predictions from all the trees. This can be done through majority voting.
4. Evaluation: Evaluate the performance of the model using the testing set: accuracy, precision, recall, and F1-score.

3.1.4 Boosting (AdaBoost):

Adaptive Boosting, also known as **AdaBoost**, is a machine learning algorithm that combines multiple weak classifiers to create a strong classifier. It is a popular and effective ensemble method used for binary classification problems.

The basic idea behind AdaBoost is to iteratively train weak classifiers on different subsets of the training data, giving more weight to misclassified instances in each iteration. The weak classifiers refer to simple classifiers that perform slightly better than random guessing, such as decision stumps (one-level decision trees).

AdaBoost can also be applied in conjunction with Random Forests to enhance their performance. This combination is known as **AdaBoost in Random Forest**.

When AdaBoost is used in Random Forest, it adds an adaptive component to the construction of individual trees. Here's how AdaBoost can be incorporated into Random Forest:

1. Initialize weights: Set equal weights to all training instances.

2. For each iteration (t = 1 to T):

   - Train a decision tree on a subset of the training data using weighted sampling, where instances with higher weights have a higher chance of being selected.
   - Compute the error rate (ε) of the decision tree based on the weighted misclassifications:

$$\varepsilon = \frac{\Sigma(weightedmisclassifiedinstances)}{\Sigma(allinstanceweights)}$$

- Calculate the weight (α) of the decision tree based on its error rate:

$$\alpha = 0.5 * ln\left(\frac{(1-\varepsilon)}{\varepsilon}\right)$$

- Update the instance weights:
  - Increase the weights of misclassified instances:

    $$w_i(t+1) = w_i(t) * exp(\alpha)$$

    Where: $w_i(t)$ is the weight of the ith instance at iteration t.
  - Normalize the weights so that they sum up to 1:

$$w_i(t+1) = \frac{w_i(t+1)}{\Sigma(allweights)}$$

3. Combine the decision trees:

   3.2 Assign a weight (β) to each decision tree based on its accuracy:

$$\beta = \frac{exp(\alpha)}{\Sigma(exp(\alpha))}$$

   3.3 The final prediction is obtained by aggregating the predictions of all decision trees using weighted voting or averaging, where the weight of each decision tree is β.

4. Repeat steps 2-3 for the desired number of iterations.

AdaBoost in Random Forest leverages the benefits of both algorithms:

- Random Forest provides robustness against overfitting

-AdaBoost adapts the weights of the instances to focus on the most challenging cases.

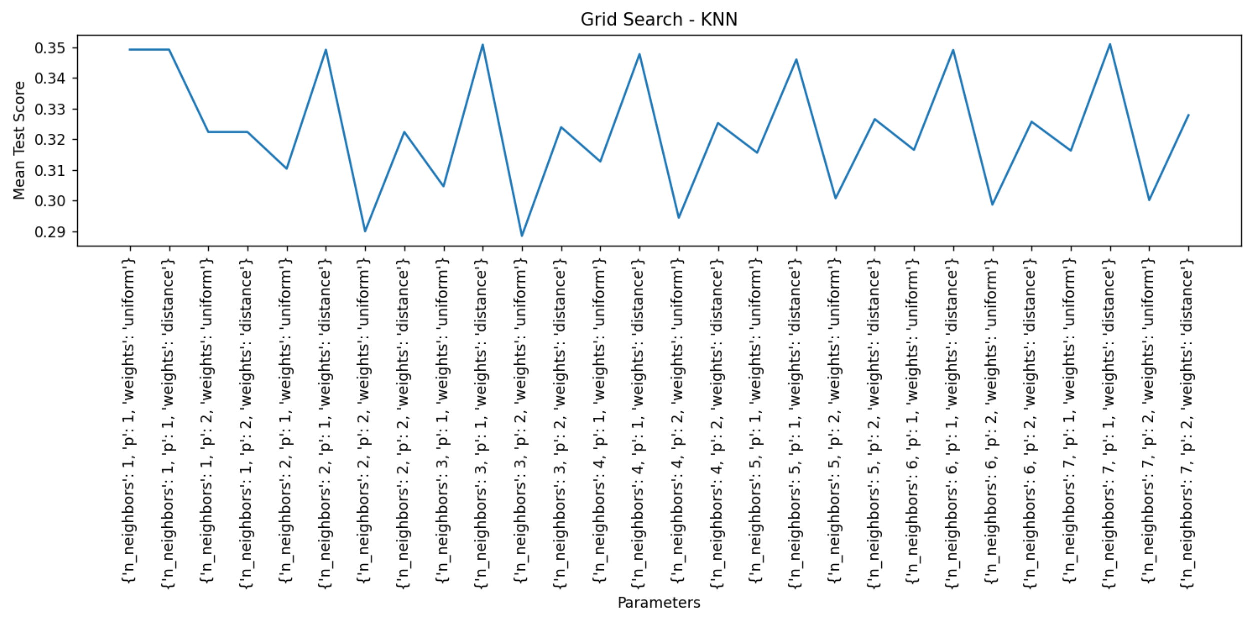
By combining these two techniques, the overall predictive performance can be improved.

3.1.5 Conclusion:

In summary, Random Forest is a powerful algorithm for facial expression recognition. It offers robustness against overfitting, provides feature importance insights, handles missing data effectively, captures non-linear relationships, and utilizes ensemble-based voting to improve accuracy. Random Forest is a strong tool for accurately predicting facial expressions when combined with preprocessing and feature engineering techniques.

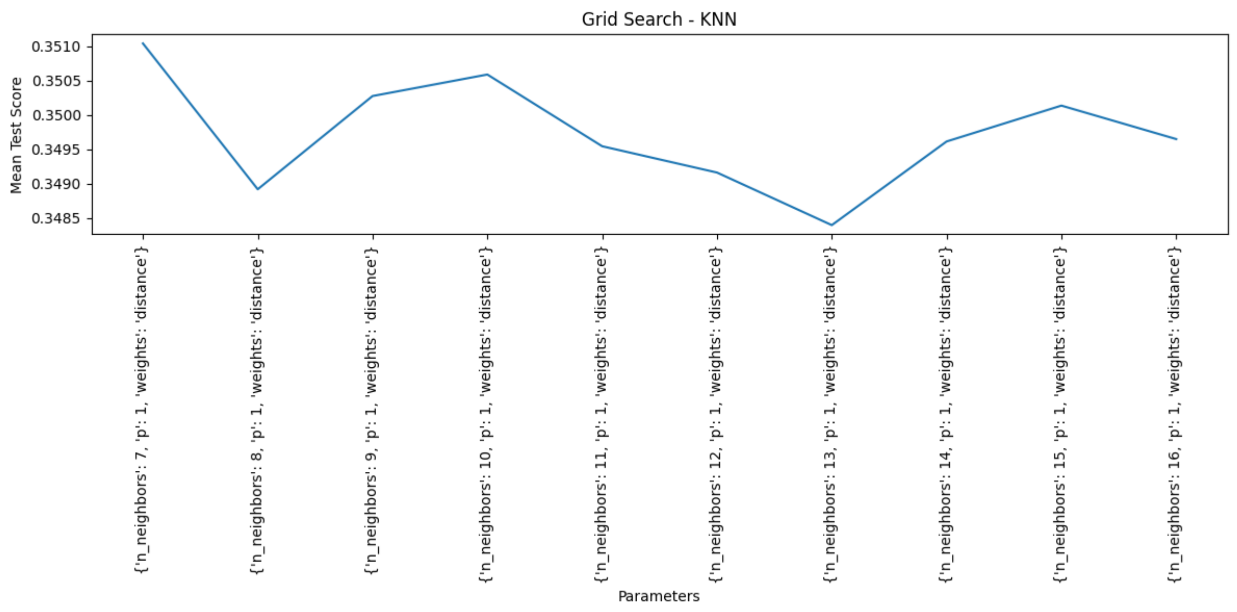
3.2 K-nearest neighbors (KNN):

3.2.1 Grid Search:



From the grid search for KNN with neighbors ranging from 1 to 7, p=1 and p=2, and weights being uniform or distance, the optimal hyperparameters are found to be p = 1, weight = distance, and n_neighbor = 7.

Due to the graph, we can conclude p=2 and weigh=uniform is not good.



Even after trying neighbors from 1 to 16, the optimal hyperparameters are still found to be p = 1, weight = distance, and n_neighbor = 7.

So we concluded that the optimal hyperparameters for your KNN model, obtained through grid search with the neighbors range from 1 to 16, remain unchanged.

The best hyperparameters are p=1, weight=distance, and n_neighbor=7.

Which mean we choose Manhattan distance and 7 nearest neighbors.

3.2.2 Procedure:

We execute KNN with the following steps:

1. Feature Extraction: Extract relevant features from the preprocessed facial data.
2. Training: During the training phase, the KNN algorithm stores the feature vectors and their corresponding emotion labels from the training set.
3. Distance Calculation: When predicting the emotion label for a new facial instance, calculate the distances between this instance and all feature vectors in the training set using Manhattan distance.

$$\text{Manhattan Distance} = d(x, y) = \left(\sum_{i=1}^{m} |x_i - y_i| \right)$$

4. Selecting k Neighbors: Select the knearest neighbors based on the calculated distances. k is the hyperparameter *neighbor* (from the result of 3.2.1).

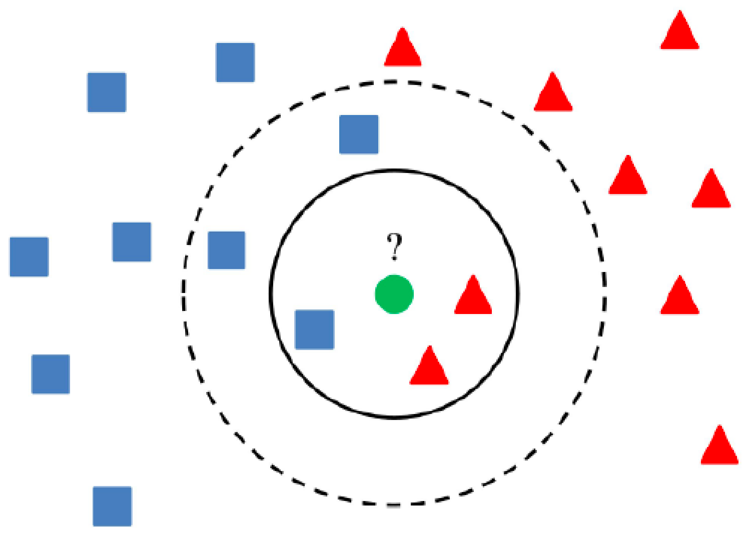


Figure 9. Select nearest neighbors example

5. Class Label Prediction: Determine the majority class label among the k nearest neighbors using voting. The predicted emotion label for the new instance will be the most frequent emotion label among the k neighbors.

6. Evaluation: Evaluate the performance of the KNN model using the testing set: accuracy, precision, recall, and F1-score.

3.2.3 Conclusion:

K-Nearest Neighbors is a simple and intuitive algorithm for facial expression recognition. It can effectively classify facial images based on the nearest neighbors in feature space. KNN is easy to implement and can provide reasonable results, especially with smaller datasets.

3.3 XGBoostClassifier:

XGBoostClassifier is an implementation of the XGBoost algorithm specifically designed for classification tasks.

XGBoost stands for "Extreme Gradient Boosting," which is a powerful and popular machine learning algorithm known for its accuracy and efficiency.

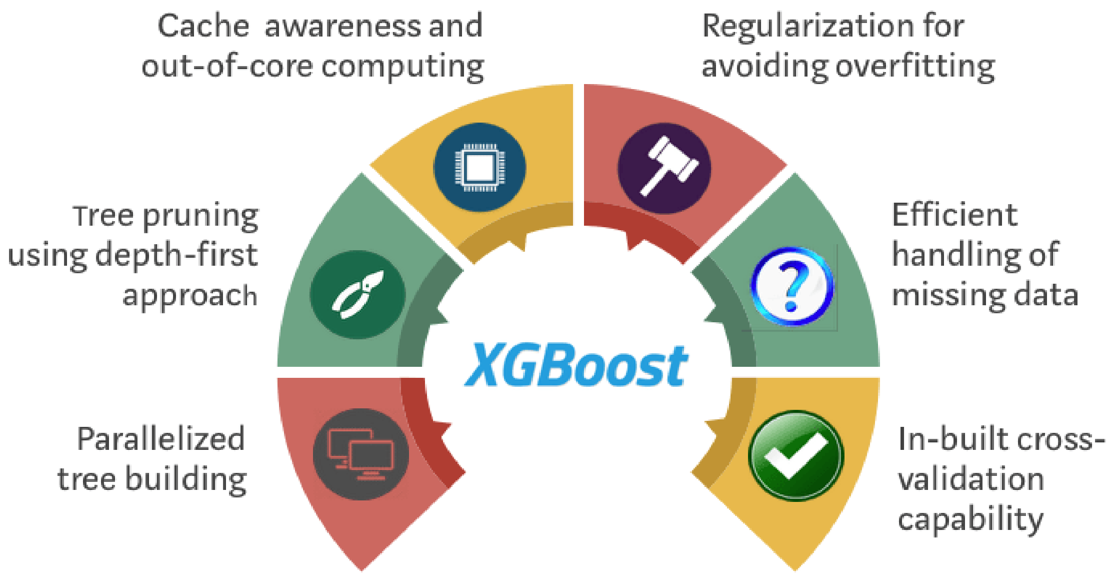

Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

Figure 10. XGBoost

XGBoostClassifier is built upon the concept of gradient boosting, which combines multiple weak predictive models (typically decision trees) to create a strong predictive model. It trains an ensemble of decision trees sequentially, where each subsequent tree corrects the mistakes of the previous ones. The model continuously optimizes a loss function by minimizing it during the training process.

3.3.1 Grid Search:

After performing Grid Search, we concluded that n_estimators = 400, lr = 0.01, max_ depth = 15 is the best parameters.

3.3.2 Procedure:

1. Feature Extraction: Extract meaningful features from the preprocessed facial images.
2. Model Training: Train the XGBoostClassifier on the training set using the extracted features and corresponding labels. The model learns to classify facial expressions based on the provided features.

3. Model Evaluation: Evaluate the trained XGBoostClassifier on the testing set to assess its performance in facial expression recognition: accuracy, F1-score, recall, precision.

4. Hyperparameter Tuning: Fine-tune the hyperparameters of the XGBoostClassifier to optimize its performance. This can be done using techniques like grid search, random search, or Bayesian optimization.

5. Final Model Training: Once the optimal hyperparameters are determined, retrain the XGBoostClassifier using the training dataset with the chosen hyperparameters to obtain the final model.

6. Model Prediction: Use the trained XGBoostClassifier to make predictions on new, unseen facial images to recognize facial expressions.

7. Model Evaluation: Assess the performance of the final trained XGBoostClassifier on facial images using evaluation metrics: accuracy, precision, F1- score, recall.

3.3.3 Conclusion:

In conclusion, XGBoostClassifier is a powerful algorithm for machine learning tasks, including facial expression recognition. Its ability to leverage gradient boosting, regularization, and feature importance analysis makes it a strong learner in capturing complex patterns and achieving high predictive performance.

5. Result and Evaluation:

- First, we comapre Random Forest technique with and without AdaBoost:
  - With AdaBoost applied, the runtime is significantly greater.
  - Meanwhile, the improvement is not significant.

⇨ The implementation of AdaBoost might be unnecessary unless better outcome is highly demanded.

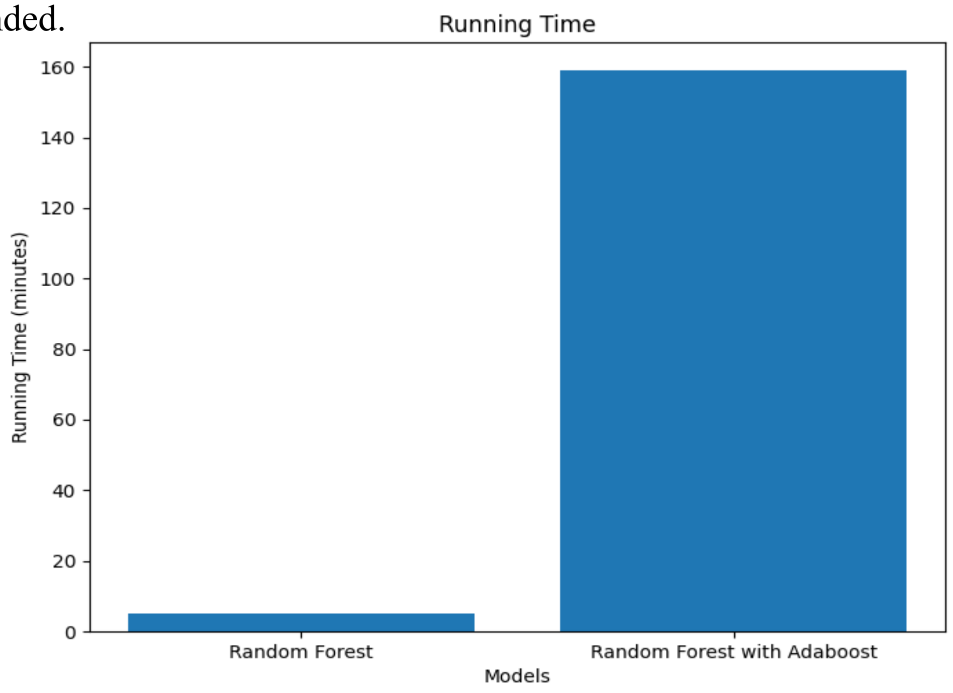


Figure 11. Running time comparison

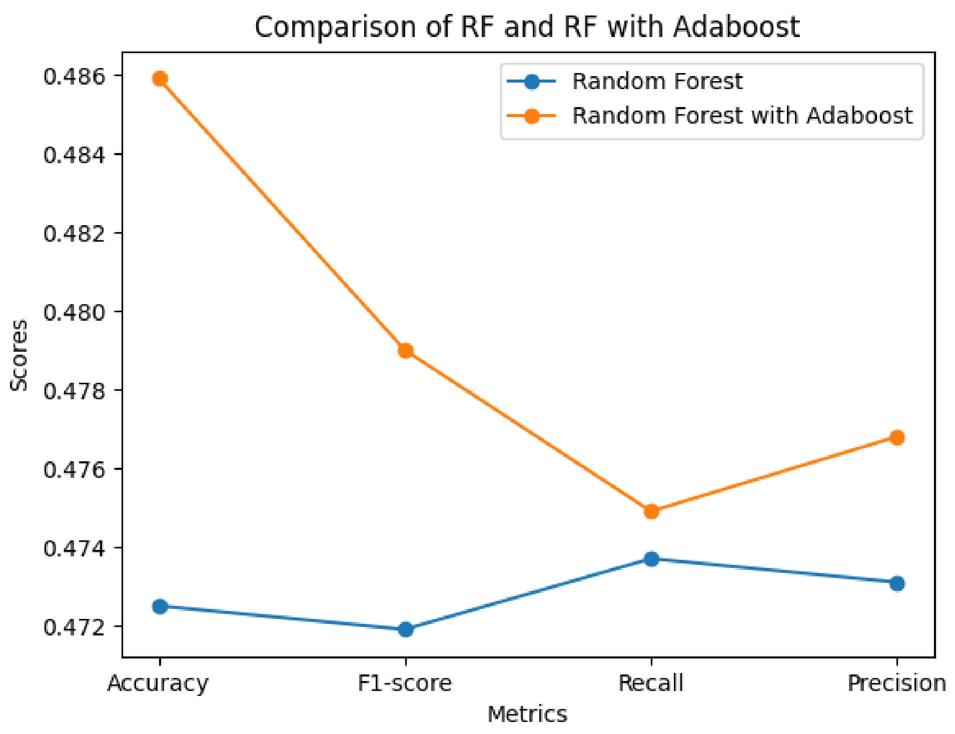
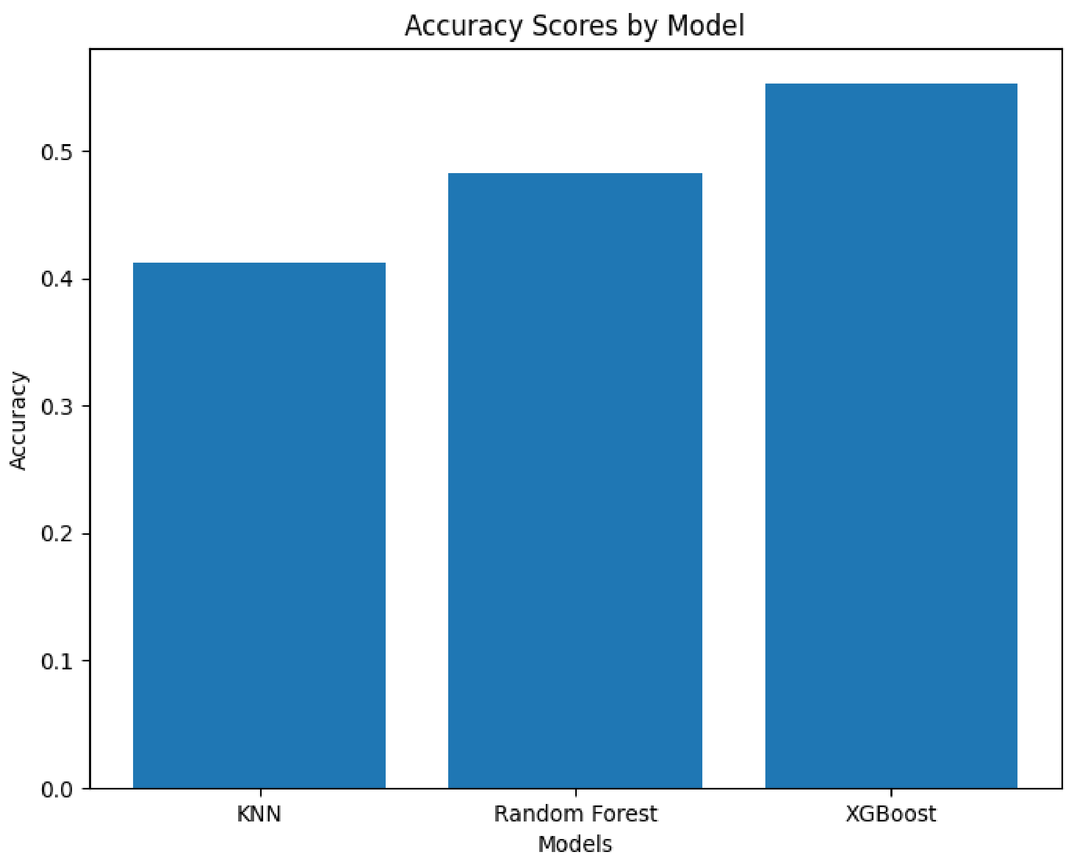
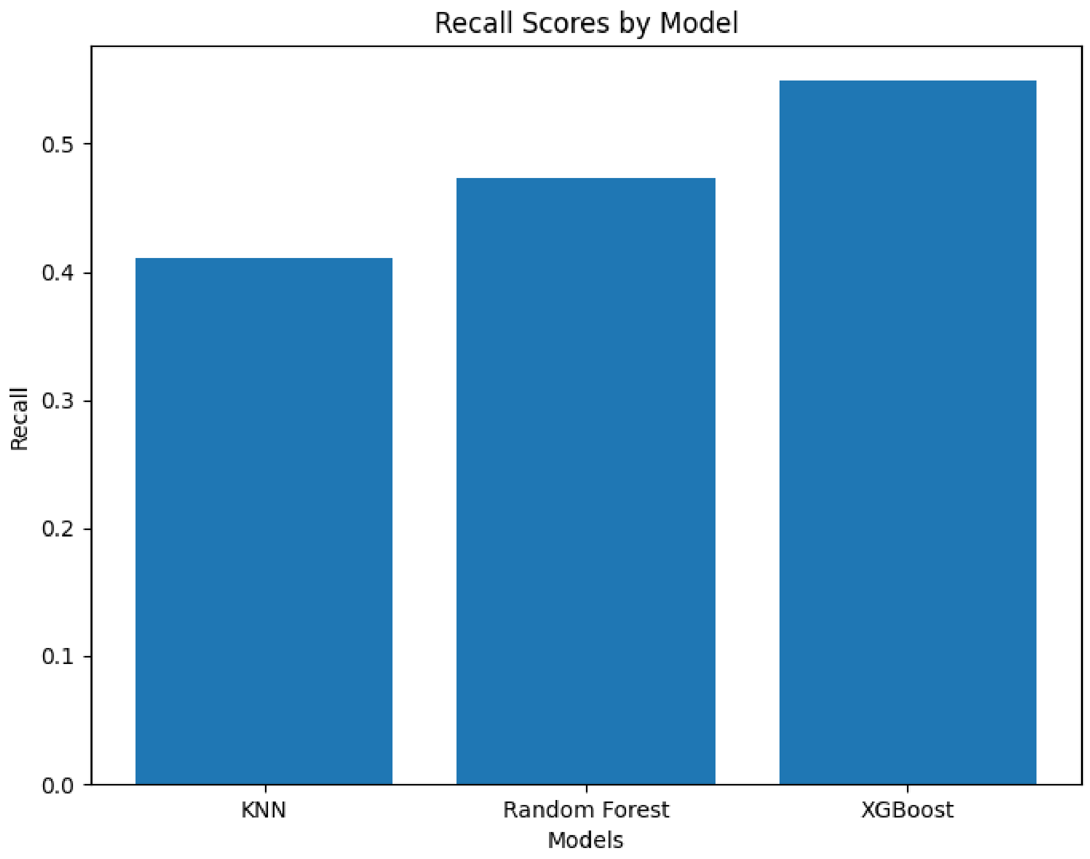
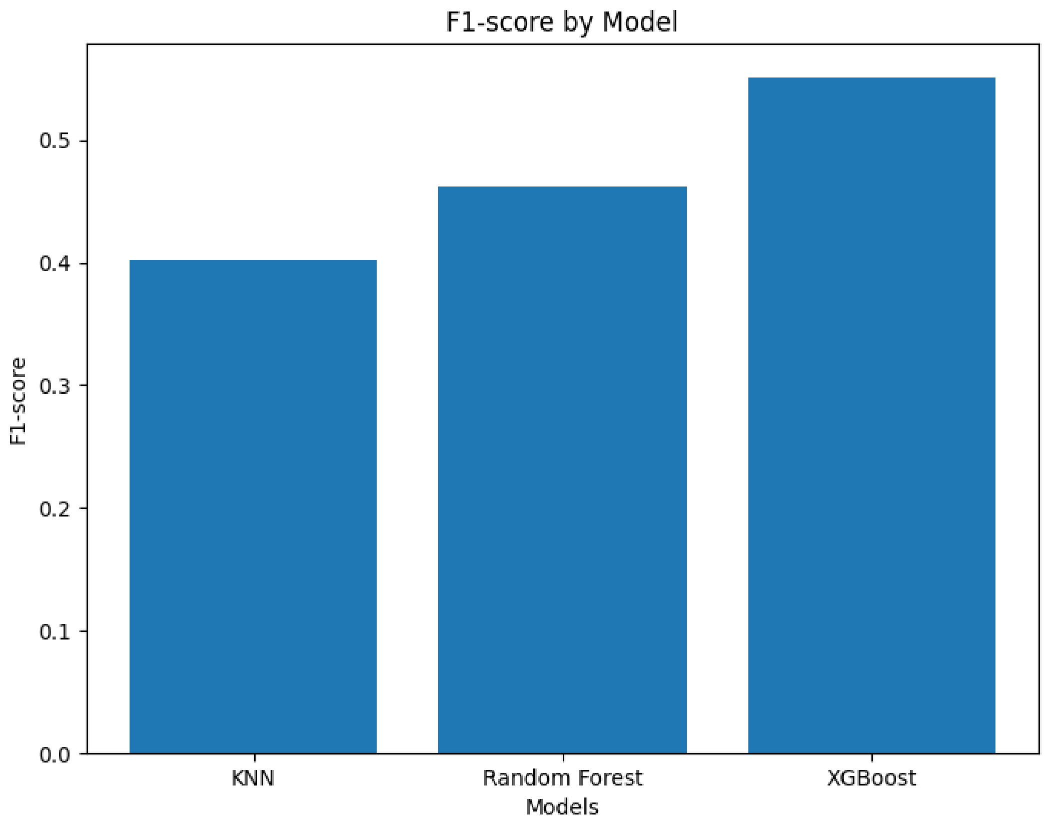


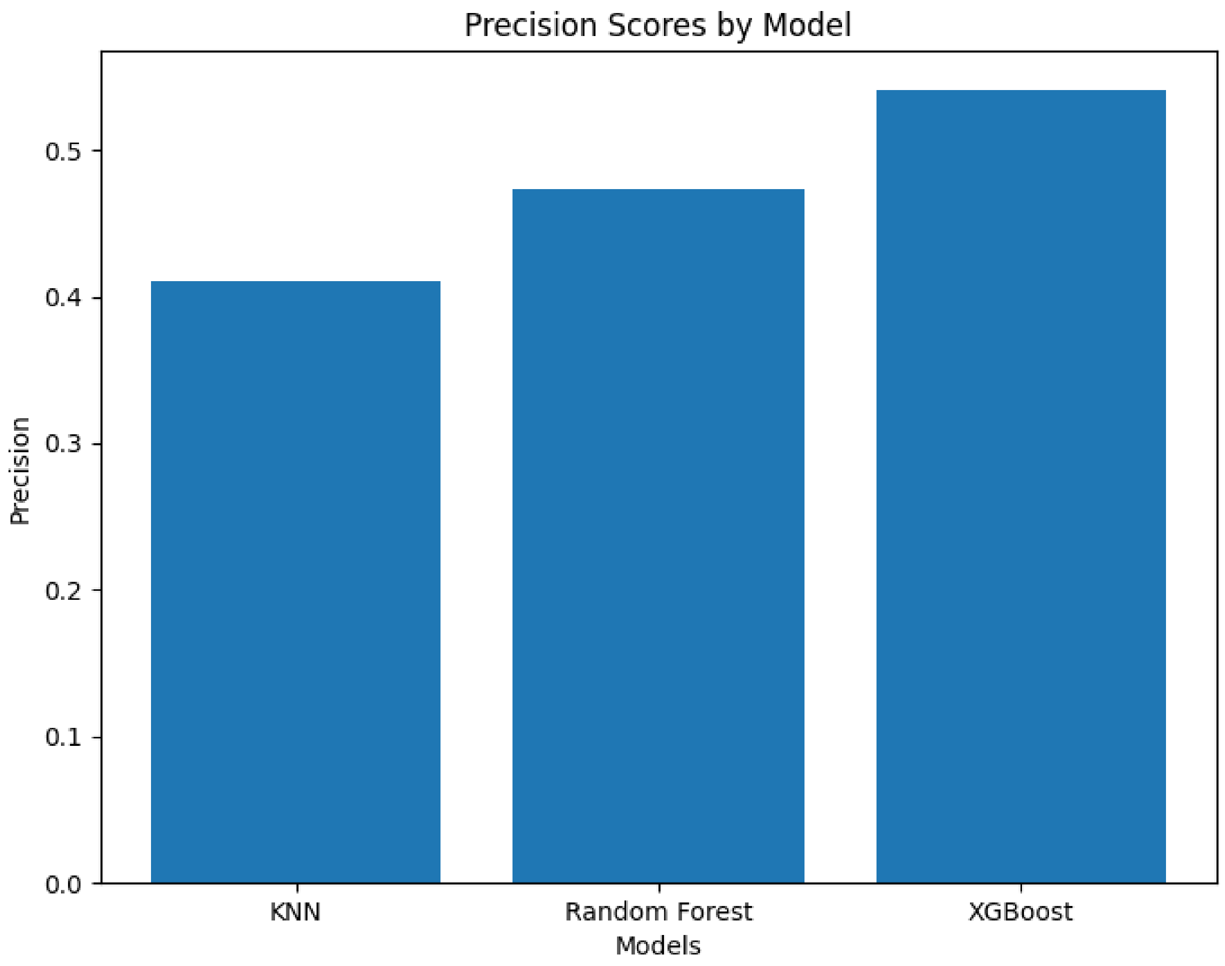
Figure 12. Result comparison

- Second, we compare each method's accuracy, recall, F1-score, and precision:




Recall Scores by Model
Recall
0.0
0.1
0.2
0.3
0.4
0.5
KNN
Random Forest
XGBoost
Models


F1-score by Model
F1-score
0.0
0.1
0.2
0.3
0.4
0.5
KNN
Random Forest
XGBoost
Models



Overall, in terms of performance and predictive capabilities, the K-Nearest Neighbors (KNN) algorithm is considered to be comparatively weaker, Random Forest exhibits improved performance, and XGBoost stands out as the most superior option.

Among these algorithms, KNN is often regarded as having limitations in handling large datasets or complex patterns. Its reliance on the nearest neighbors for classification can lead to suboptimal performance in certain scenarios.

On the other hand, Random Forest, an ensemble algorithm, demonstrates better performance than KNN. It combines multiple decision trees and leverages their collective predictions to achieve enhanced accuracy and robustness in handling various types of data.

However, XGBoost surpasses both KNN and Random Forest in terms of performance and versatility. XGBoost utilizes gradient boosting techniques and incorporates advanced features such as regularization and parallel processing. This enables XGBoost to handle complex relationships within the data and yield exceptional predictive accuracy.

Considering these factors, XGBoost emerges as the preferred choice among the three algorithms, offering the highest performance and superior predictive capabilities.

6. Conclusion and Future work:

6.1 Conclusion:

In conclusion, traditional machine learning methods have been successfully applied to facial expression recognition tasks. Techniques such as K-Nearest Neighbors (KNN), Random Forest, and other ensemble algorithms have demonstrated promising results in classifying facial expressions.

6.2 Future work:

However, it's important to note that these traditional methods may have limitations in capturing complex spatial dependencies and extracting intricate facial features. As a result, there is a growing trend towards using deep learning approaches, such as Convolutional Neural Networks (CNNs), which have shown superior performance in facial expression recognition tasks. While traditional machine learning methods can still provide satisfactory results, leveraging deep learning techniques has become increasingly prevalent for more accurate and robust facial expression recognition. In the future, our intention is to incorporate Convolutional Neural Networks (CNNs) into our approach.