UNIVERSITY OF ILLINOIS, SPRINGFIELD

CSC 532 – MACHINE LEARNING

FINAL PROJECT REPORT

FACIAL EXPRESSION RECOGNITION FROM FACE PROFILE IMAGES

BY

ANUSHKA KUMARI (UIN: 669671211)

MIRIYALA SREE HARISH KUMAR (UIN: 663258586)

Facial Expression Recognition from Face Profile Images

Abstract

Facial expression plays a major role in expressing what a person feels, and it expresses inner feeling and his or her mental situation or human perspective. This project focuses on predicting human emotions from facial expressions using machine learning techniques. Recognizing emotions through facial features is a complex task due to the variety of expressions. We used multiple methods to predict emotions, including K-Nearest Neighbors (KNN), Logistic Regression, OpenCV using Neural Networks, and YOLO V4. Each model analyzes the data differently, allowing us to compare their performance in predicting emotions.

While these models help recognize emotions, they vary in how accurate and efficient they are. By testing and refining these techniques, we aim to improve the system's accuracy. This approach makes the system more reliable and useful for real-world applications in areas like healthcare, education, and security, where understanding emotions can enhance interactions and decision-making.

Problem Definition and Project Goals

This project aims to identify the basic Human emotions such as sad, fear, happy, anger, surprised, and neutral emotions. The dataset we have used is **ICML Face Dataset,** We used from https://figshare.com/articles/dataset/icml_face_data_csv/19792891?file=35165338.

The ICML Face Dataset is a set of facial pictures that have been converted into grayscale and made available on Figshare. The dataset is in a CSV file, where we can find the pixel information of each image along with the respective emotion label. The dataset has about 36,000 images, making it ideal for training ML models. Each figure is allocated by numbers displaying the brightness of each pixel, starting from 0 (black) to 255 (white). Every picture also has a label that identifies the emotion on the face. The seven emotions are divided as Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral, each represented by a number that begins from Zero to 6. This structure greatly assists in training models to recognize emotions based on the pixel patterns.

Firstly, our plan of action for using this data is to load it into Python Notebook and later organize it into pixel intensity values as input features and emotion labels as output. Secondly, we normalize the pixel values on a scale of 0 to 1 to ensure the data is stable and improve the training process ahead. We plan to use popular methods like K-Nearest Neighbors (KNN), OpenCV Neural Networks, Logistic Regression, and YOLOv4, as each model analyzes the data differently, allowing us to compare their performance in predicting emotions.

Post training, we will test the model on images fed to see how well it predicts emotions, using evaluation metrics like accuracy, precision, recall, F1-score, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) to evaluate its performance. Once success value is observed, the model could be applied to real-world scenarios like analyzing customer emotions, assessing mental state, or enhancing virtual assistants to respond to emotions more effectively.

Related Work

We have found a few related works to Facial Emotion Recognition:

1. Facial Emotion Recognition Using VGG19 (FER-2013 Dataset)

Link:https://www.kaggle.com/code/enesztrk/facial-emotion-recognition-vgg19-fer2013#Facial-Emotion-Recognition-%7C-VGG19-Model---FER2013-Dataset

This Kaggle project focuses on facial emotion recognition using a deep learning model called VGG19 (Visual Geometry Group 19). It uses the FER-2013 dataset, that contains grayscale figures of faces labeled with nearly 7 emotions that are distributed as angry, disgust, fear, happy, neutral, sad, and surprise. The goal of the project is to train a model to recognize these emotions accurately and effectively from facial images. VGG19, which is a popular convolutional neural network, is pre-trained on a huge image dataset (ImageNet) and fine-tuned for emotion recognition. It includes around 28,000 images for training, 3,500 for validation, and 3,500 for testing. The data is stored in a CSV file format, where every image is presented as a long dimensional array of 2304 pixel values and its respective emotion label. Pixel arrays are restructured into 48x48 grids, normalized to a 0-1 range for quicker model training, and enhanced with flipping, rotating, and cropping procedures to increase dataset variation and enhance the model's generalization capabilities in order to prepare the dataset for models such as VGG19.

The FER-2013 dataset is used to train models for recognizing emotions in faces, with real world implementations in areas like virtual assistants, mental health monitoring, and analysis of customer feedback. However, it has some obstacles, like fewer images for certain emotions (e.g., "disgust"), low resolution image that makes subtle expressions harder for the model to detect, and the absence of background context that might assist in understanding emotions. Despite these known issues, it remains a valuable tool for projects where emotion is considered as the major key factor.

Facial Human Emotion Recognition using YOLO Face Detection Algorithm

Link:
https://www.researchgate.net/publication/374779447_FACIAL_HUMAN_EMOTION_RECOGNITION_BY_USING_YOLO_FACES_DETECTION_ALGORITHM

It's a research paper from Research Gate. The authors Mustafa Asaad Hasan and Ali Lazem mentioned their ideology about Human emotions. These emotions are used in real-world scenarios such as human-machine interactions, safety, and healthcare. From their study, they used the YOLO face detection to identify faces and analyze facial features. These features helped them allocate faces into one of seven emotions like natural, happy, sad, angry, surprised, fearful, or disgusted. This experiment proved that their system is both Quick and reliable. They tested their approach with the FER2013 dataset and successfully achieved an impressive accuracy of 94%.

Data Exploration and Preprocessing

1. Data Cleaning

Data cleaning is the process of fixing incorrect or duplicate data. In this process we find error in data and then changing, updating or removing data to correct them.

Steps for Data cleaning are as follows:

1.1 Understand the Dataset

The first step in data cleaning is clear understanding of the structure and contents of the dataset. This includes examining the features, such as emotion labels ("happy," "sad," "neutral") and pixel data that represent the facial images. We have checked the format of the pixel data, often stored as a string of space-separated values, and confirm whether it matches the expected resolution (48x48). Additionally, we inspect the dataset for thorough completeness, data types, and value ranges to identify any inconsistencies or issues that may need to be addressed later.

1.2 Handle Missing Data

Missing data can cause errors or inconsistencies during training, so it is crucial to identify and handle it effectively. Missing emotion labels can make images unusable, while missing or empty pixel data renders images incomplete. Rows with such missing values should typically be removed to ensure the dataset is clean. Alternatively, if missing data is minimal and can be inferred, imputation strategies may be applied. However, caution is necessary to avoid introducing bias.

1.3 Detect and Remove Duplicates

Duplicate data can skew the method of training process by over representing specific samples, leading to biased models. Detecting duplicates includes checking for rows with identical pixel data and labels. Removing these redundant entries makes sure that each and every sample has an equal influence on model training, upgrading generalizability and preventing overfitting to repeated patterns.

1.4 Inspect and Clean Pixel Data

The pixel data should be inspected for validity and consistency. For facial emotion recognition datasets, pixel data is often stored as strings that need to be converted into numerical arrays. It is also important to ensure that each image has the correct number of pixels (e.g., 48x48 = 2304). Any rows with incomplete or improperly formatted pixel data should be removed to avoid errors during pre-processing or model training.

1.5 Normalize Pixel Intensity

Pixel intensity values in images typically range from 0 to 255, but machine learning models work better with stabilized inputs. Normalizing the pixel values to a range of 0 to 1 by dividing by 255 ensures that the data is scaled uniformly. This step not only helps the model converge faster but also reduces the likelihood of numerical instability during training process.

1.6 Handle Outliers

Outliers in the dataset, such as images that are too bright, too dark, or mislabeled, can negatively impact model performance. Brightness outliers can be identified by analyzing pixel intensity distributions, while mislabeled images may require manual inspection. Removing or correcting these outliers secures that the dataset remains representative of the task and improves model accuracy.

1.7 Validate Labels

Emotion labels should be consistent and error free. It is important to verify that all expected emotion categories are present, correctly spelled, and uniformly labelled. Undefined, unknown, or inconsistent labels should be removed or corrected. This step ensures that the labels accurately correspond to the facial expressions, which is crucial for supervised learning tasks.

1.8 Address Data Imbalance

Emotion classes in facial emotion recognition datasets are often imbalanced, with some emotions having significantly fewer samples. This imbalance can lead to biased models that perform poorly on minority classes. To address this, techniques like oversampling (duplicating samples of minority classes), under sampling (reducing samples from majority classes), or data augmentation (creating synthetic variations of minority class images) can be used to achieve a more balanced dataset.

1.9 Save the Cleaned Dataset

Once the dataset is cleaned, it should be saved in a structured and consistent format, where file format such as CSV or organized folders of images. This ensures that the cleaned data is ready for pre-processing step, feature extraction, and model training. Saving the dataset at this stage also allows for easy reuse or sharing with team members, achieving the state of reproducibility in experiments.

2. Data Preprocessing

2.1 Data Loading and Conversion

The first step involves loading the dataset into workspace. Typically, pixel values in image datasets are stored as strings in a CSV format. Each image is a grid of pixel values, often flattened into a 1D array. For example, a 48x48 grayscale picture which will have 2304 pixel values. You need to restructure this flat data into a 2D matrix to represent the image properly. The formula to reshape the data is:

Formula:

if pixels = [p1, p2, ..., p2304] (1D - array), convert shape into a 2D array:

figure = pixels.reshape (48,48)

It reshapes the one-dimensional array of pixel values into a two-dimensional array with a structured 48 rows and 48 columns, representing the dimensions of the image.

2.2 Resizing Images

Images in your dataset may come in different varying sizes, but machine learning models require a static input size (let's say 48x48 or 224x224 pixels). To ensure all images are of same size, one way is to resize them using libraries like OpenCV or PIL. This step assists in maintaining consistency across the dataset and makes it bit easier for the model to process the images.

Formula (using OpenCV):

resized_image=cv2.resize(image, (height, width))

This formula resizes the image to the desired height and width (e.g., 48x48 pixels).

2.3 Normalize Pixel Values

Normalization makes sure that the pixel values of the figure are scaled to a range that lie between 0 and 1. Since values of pixel are usually in the range of 0 to 255, converting each pixel by 255 scales is down to a [0, 1] range. This helps in achieving the training efficiency of machine learning models.

Formula:

If x is pixel value, then x / 255.0

2.4 Label Encoding

Emotion labels mentioned in the dataset (like "happy," "sad," or "angry") need to be converted into numerical values for machine learning models to understand. There are two common methods namely Label Encoding and One-Hot Encoding.

- Label Encoding assigns a unique integer value to each label.

- One-Hot Encoding creates a binary vector where each element corresponds to a label and only one element is 1 (indicating the class).

Formula for Label Encoding:

Assign a unique integer to each emotion,

For example, "happy" → 0, "sad" → 1.

Syntax for One-Hot Encoding where "happy" is assigned to [1, 0, 0, 0], "sad" = [0, 1, 0, 0]

Both of these methods convert categorical emotions into numerical format that can be fed into a machine learning model.

2.5 Data Augmentation

Data augmentation is a great technique that is used to make a training dataset bigger by creating new versions of the existing data via various transformations. This action improves the model's performance and prevents it from entering too specialized to the training data (overfitting). By making slight minor changes to the images, like rotating, shifting, or changing their brightness, the model gets exposed to a wider boundary of examples. This makes the model ultimately achieve adaptability and capable of identifying objects in various circumstances when it encounters new and unseen data. Essentially, data augmentation assists the model learn to generalize better and perform well in real-world scenarios.

```
# Data augmentation that is used in YOLO Algorithm
train_datagenerator = ImageDataGenerator(
   rotation_range=10,
   width_shift_range=0.1,
   height_shift_range=0.1,
   zoom_range=0.1
)
val_datagenerator = ImageDataGenerator()
```

2.6 Data Flattening

Flattening an image in preprocessing means converting the 2D matrix of pixel values into a 1D single vector. Since machine learning algorithms like Logistic Regression expects input data in the form of a 1D array, each pixel value of the image is treated as a separate feature. For example, we have an image with 28x28 pixels, flattening turns it into a vector with 784 values (28 * 28). This process allows the image to be fed as input for machine learning models, where each number in the vector corresponds to a pixel's value.

```
# Flatten image data for Logistic Regression is expressed below

X = X.reshape(X.shape[0], -1)
```

2.7 Train-Test Split

Post preparation of dataset, it is typically split into three normal parts: one for training the model and rest for fine-tuning (validation), and for evaluating its performance (testing). One common way to split the data is 70% for training, 15% for validation, and 15% for testing.

Syntax:

```
# Separate the data into training and validation sets (from YOLO V4 Algorithm)

X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)
```

This assures that the model learns on single subset of data and it access on unseen data in order to avoid overfitting.

2.8 Handle Class Imbalance

If some emotions are overrepresented in the dataset (e.g., more "happy" faces than "fear"), the model may develop a bias towards the more frequent emotions. To solve this, we can use the below techniques like:

- Oversampling: Add more examples from the underrepresented classes.
- Undersampling: Reduce examples from the overrepresented classes.
- Data Augmentation: Create more images for underrepresented classes.

These methods help in balancing the dataset, ensuring the model learns to predict all emotions accurately.

2.9 Save Preprocessed Data

Once all the preprocessing steps are done, it's important to save the cleaned and processed data so you don’t have to redo all the steps each and every time you train the model. Saving the data allows you to quickly load it later for model training without replicating the work. Typically, this data is saved as NumPy arrays, which are a convenient format for storing numerical data. These arrays can easily be loaded back into memory whenever needed, making the process faster and way more efficient. In simple terms, it’s like storing your cleaned data in a file so that you can use it anytime without starting from scratch.

Data Analysis and Experimental Results

We are using machine learning models for Facial Emotion Recognition (FER) to understand emotions by looking at facial expressions in images. These models are trained using algorithms that learn from many sample pictures of faces with labels mentioned like "happy," "sad," or "angry." We focus on information like the eyes and mouth to figure out the emotion. Once the model is trained well, we can use it to predict the emotion of a new face by recognizing similar patterns in the facial features.

In this Facial Emotion Recognition from Face Profile Images, we are using the below following models:

Simple Models: KNN (K-Nearest Neighbors), Logistic Regression

Advanced Models: OpenCV using Neural Networks, YOLO V4 Algorithm

Simple Models:

KNN (K-Nearest Neighbors)

In this project, we used the K-Nearest Neighbors (KNN) algorithm to classify facial expressions based on pixel data from images. The dataset contains three columns: emotion (the target label), Usage (e.g., training or testing), and pixels (a string of pixel intensity values for each image). We first processed the pixel data by splitting the string into individual numbers, normalizing them to a scale of 0 to 1, and converting them into a usable format for machine learning.

We then prepared our features (image pixels) and labels (emotion categories) and split the data into training and testing sets. After training the KNN model with 5 neighbors on the training data, we evaluated it using the test set. The model achieved an accuracy score of 35.11%, that means it correctly classified 35% of the test samples. We also generated a classification report to analyze precision, recall, and F1-score for each emotion category. Finally, we visualized a few test images along with their actual and predicted labels to better understand the model's predictions. While the accuracy is relatively low, this is expected for a basic model like KNN on a complex task like facial expression recognition.

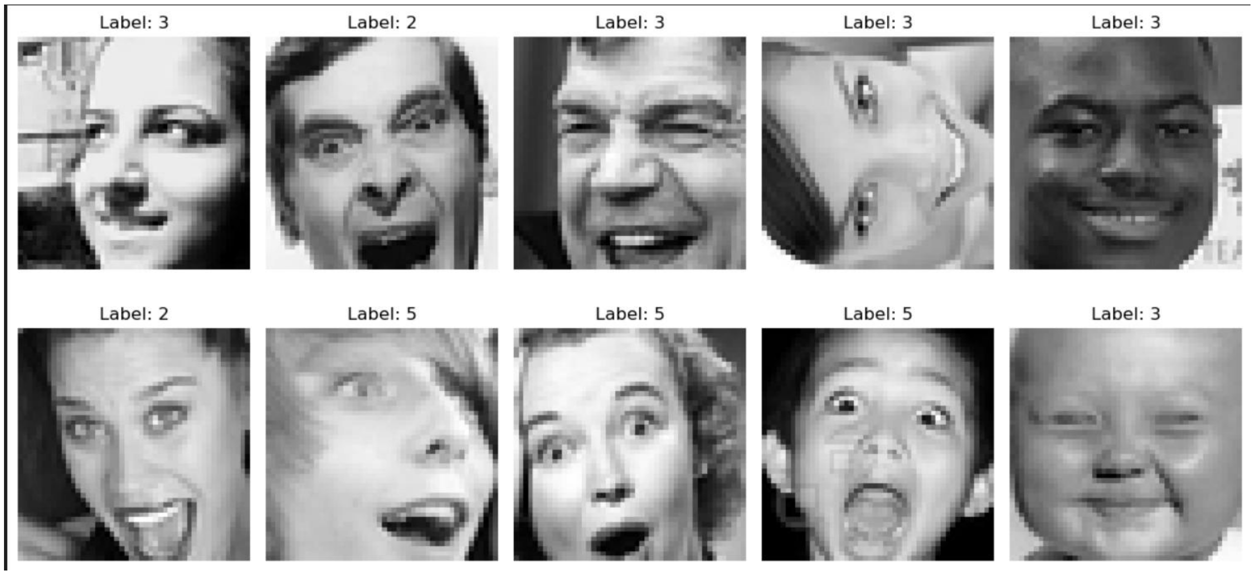

Figure 1: Prediction for KNN Algorithm

Logistic Regression

In this project, usage of Logistic Regression, a simple machine learning algorithm, is used to recognize facial expressions from images. The dataset included emotion labels (like happy or sad) and pixel values that represent the images. We prepared the data by converting the pixel values from text to numbers, normalizing them to fall between 0 and 1, and flattening the image data into rows since Logistic Regression works with this format.

We split the data into training and testing sets, trained the model on the training data, and tested it on new images. The model's performance was measured using accuracy and a detailed report to see how well it predicted each emotion. We also visualized some test images with their actual and predicted emotions to understand its results. This showed how Logistic Regression can be used as a simple and effective method for facial expression recognition, even though it might not capture complex patterns as well as advanced models.

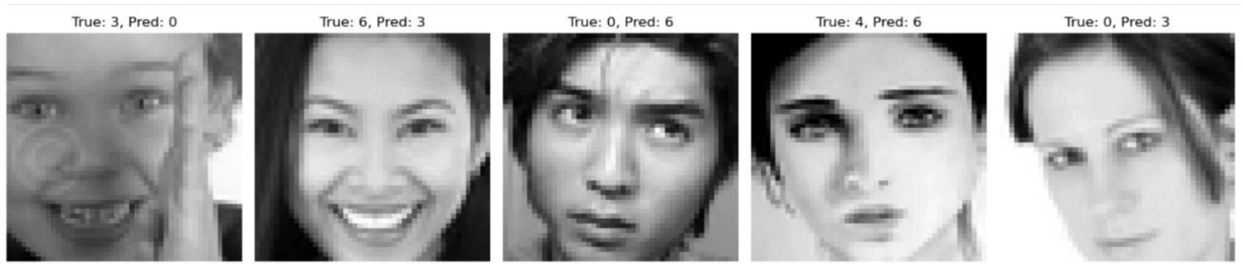

Figure 2: Prediction for Logistic Regression Algorithm

Hyperparameters for Simple models:

For K-Nearest Neighbors (KNN):

- KNN neighbors: We set this value to 5, which means that the model looks at the 5 closest neighbors in the dataset to make a crystal-clear prediction. This mainly helps the KNN algorithm decide the most likely emotion based on nearby data points.

For Logistic Regression:

- max_iter: We fix max_iter to a limit of 1000. This means the model tries thousand steps to find the best solution by adjusting its internal parameters during training.
- solver: We chose the 'lbfgs' solver, which is a reliable and fast method for training logistic regression models, especially for small to medium datasets.
- multi_class: We set multi_class to 'multinomial', telling the model that it needs to classify multiple emotion categories (like happy, sad, angry) instead of just binary outcomes.

Complex models:

OpenCV using Neural Networks

In this code, we are creating a system to recognize emotions from facial images using a neural network. First, we load a dataset of face images and their emotion labels (like happy, sad, etc.). We process the image data by converting it into arrays, reshaping it into 48x48 grayscale images, and normalizing the pixel values to make it easier for the model to learn. We also convert the emotion labels into a format the model can understand (one-hot encoding) and split the data into training and testing sets so we can train the model on one part and evaluate its performance on unseen data.

Next, we build a neural network to learn patterns in the images and classify them into emotions. It includes layers that detect features like edges and patterns, and dropout layers to avoid overfitting. We train the model for 20 rounds (epochs), checking its performance on both training and validation data. After training, we test the model on the test set, and it achieves about 54% accuracy. Finally, we save the trained model for future use and plot graphs to visualize how the accuracy and loss changed during training.

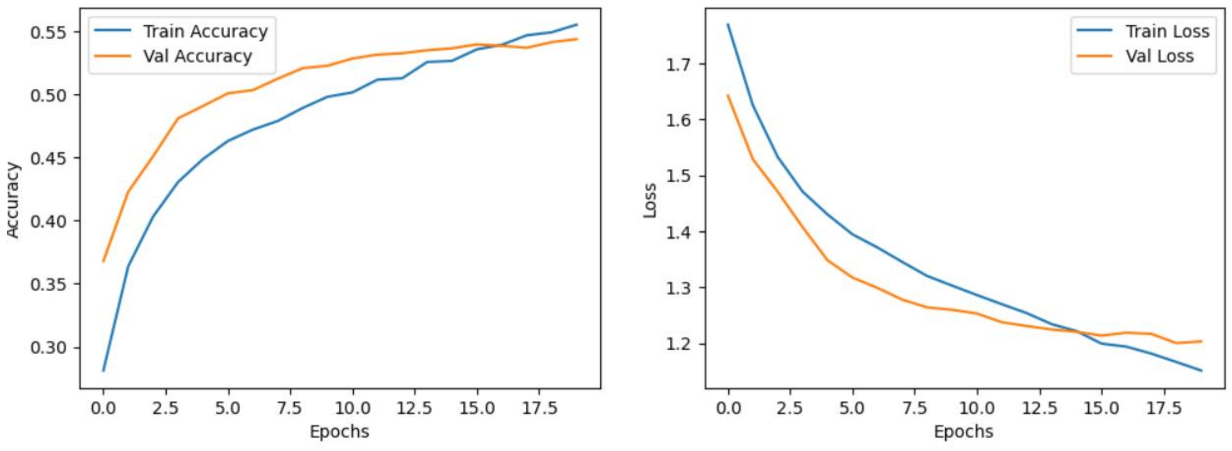

Figure 3: Accuracy and Loss graph using Open-CV Neural Network

YOLO V4 Algorithm

We used following steps for YOLO V4 Algorithm:

Loading YOLO Model and Classes

We start by loading the YOLO (You Only Look Once) model, which is a popular object detecting algorithm. We use its pre-trained configuration (yolov4.cfg) and weights (yolov4.weights) files. The model uses a file called coco.names, which contains the names of 80 common objects where it can detect like cars, people, and chairs. By setting this up, we enable YOLO to recognize these objects in images or videos.

Building a CNN Model for Emotion Recognition

Next, we define a Convolutional Neural Network (CNN) for recognizing emotions from facial images. The network includes 4 layers namely convolutional layers to find image patterns, pooling layers to reduce the size of these patterns, and fully connected (dense) layers for classification of emotions. The last layer outputs one of seven emotions, such as happy, sad, or angry. This is the core notion of our emotion detection system.

Loading and Preprocessing the Dataset

We use a file named icml_face_data.csv as our dataset. This file has pixel values of 48x48 grayscale facial images and their corresponding emotion labels. We preprocess this data by converting pixel values from strings into numerical arrays, reshaping them to fit the input of our CNN, and normalizing them to make the training process faster and more accurate. The emotion labels are also one-hot encoded, which means we represent each emotion as a binary vector.

Splitting Data for Training and Validation

To train and evaluate model, we split the dataset into two classic parts namely training (80%) and validation (20%). The training data is used to train the model which is of 80% of data in the dataset, while the validation data helps us to check how well the model performs on unseen images. This ensures that our model has learned and is in a state to yield correctly.

Data Augmentation

To make the model more robust, we apply data augmentation. This means we artificially increase the size of our training dataset by applying random transformations, like rotating, zooming, or shifting images. This helps the model learn much better and generalize well to new, unseen data.

Training the Model

We train the CNN model using the augmented training data while validating its performance with the validation set. The training runs for 20 epochs, during which the model learns to improve its accuracy gradually. After training, the validation accuracy reaches about 54%, showing that our model can moderately recognize emotions.

Saving the Trained Model

Once model is trained, we are allowing it to save it as a file named emotion_model.h5. This allows us to use the model later without having to retrain it from scratch, saving both time and computational resources.

Testing the Model

We test our saved model by loading it and using it to predict emotions for new images. For example, we display a random image from the validation set and use the model to predict its emotion. The model outputs a label, like "Happy" showing how well it can recognize emotions in unseen images.

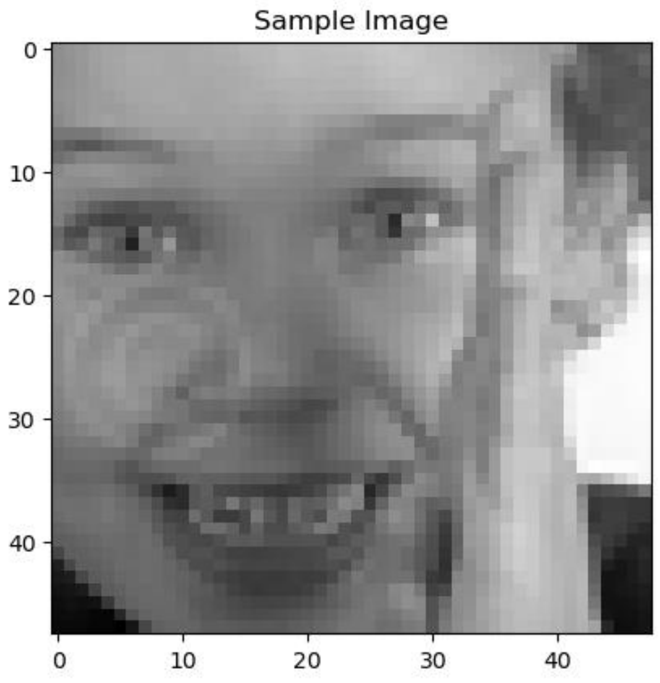

Figure 4: YOLO V4 Prediction Image

Hyperparameter for Complex Models

Hyperparameter Tuning in OpenCV using Neural Networks

When we use OpenCV for tasks like image recognition or classification, we can adjust some hyperparameters to improve performance.

1. **Learning Rate:**

   The learning curve rate decides how well the model changes with each learning step. If it's too high, model might skip the best solutions. If it's too low, learning takes a longer time. Tried values like 0.01, 0.001, or 0.0001 to see what works the best.

2. **Batch Size:**

   Batch size is how many images the model processes at a time. Smaller batches use less memory but can make training less stable. Bigger batches are smoother but need more resources. We test sizes like 16, 32, or 64 to find the right balance.

3. **No. of Epochs:**

   Epochs are how many times the model sees the entire dataset. If we use too few, the model might not learn enough. Too many can make the model focus too much on training data. We adjust this to avoid underfitting or overfitting**.**

4. **Optimizer:**

   The optimizer controls how the model updates its weights. We can use options like Adam, SGD (Stochastic Gradient Descent), or RMSprop and see which one works best for our problem.

5. **Model Architecture:**

   We can experiment with the number of layers, the size of each layer, and the activation functions (like Relu or Sigmoid) to improve how well the model performs.

Hyperparameter Tuning in YOLOv4

YOLO v4 is great for object detection, and tuning its hyperparameters helps us make it more accurate and faster.

1. **Input Image Size:**

   YOLOv4 needs images to be a specific size, like 416x416 or 608x608. Larger sizes give better accuracy but need more processing power. We choose a size based on whether we need speed or precision**.**

2. **Confidence Threshold:**

   This decides how confident YOLOv4 must be to mark an object as detected. For example, if the threshold is 0.5, it only keeps detections with at least 50% confidence. We adjust this to reduce false positives or missed detections.

3. **Non-Maximum Suppression (NMS):**

   NMS makes sure overlapping boxes for the same object are merged into one. We can tune the overlap limit (e.g., 0.3 or 0.5) to improve the results**.**

4. **Anchor Boxes:**

   Anchor boxes help YOLOv4 detect objects of different sizes. By matching the box sizes to the sizes of objects in our data, we make the detection better.

5. **Learning Rate and Warm-Up:**

   Just like with OpenCV models, the learning rate controls how fast YOLOv4 learns. It also has a warm-up period to stabilize training. We can adjust these settings for better results.

6. **Batch Size:**

   The batch size decides how many images are used in one step of training. Bigger sizes give smoother updates but need more memory. We test different sizes to find the best option for our system.

7. **Data Augmentation:**

   YOLOv4 uses tricks like flipping, scaling, or changing colors in images to make the model learn better. We can tune how often these tricks are applied to improve the model's robustness.

8. **Loss Functions:**

   YOLOv4 uses multiple loss functions to decide how well it predicts objects, their locations, and their confidence scores. We can adjust the importance of each type of loss to make the model focus on what we care about most.

Benchmarking:

In this code, comparison of four models' performance is taken place namely KNN Classifier, Logistic Regression, OpenCV using Neural Network, and YOLO. Each model's results were evaluated using common metrics such as RMSE, MAE, Accuracy, Precision, Recall, and F1 Score. We have analyzed which model performed the best and why.

We are using following Evaluation Metrics:

Model	**RMSE**	**MAE**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
KNN	0.7071	0.5000	50%	0.6600	0.5000	0.4798
Logistic Regression	0.6325	0.4000	60%	0.5373	0.6000	0.5467
Open-CV using Neural Networks	0.7071	0.5000	50%	0.5000	0.5000	0.5000
YOLO V4 Algorithm	0.6325	0.4000	60%	0.8133	0.6000	0.5838

Table 1: Comparison of Algorithms using Evaluation Metrics

Analysis:

1. KNN Classifier

KNN performed moderately but struggled with precision and recall. It worked well for one class (label 0) but missed many samples from the other class (label 1). The accuracy is only 50%, which means it didn't classify many test samples correctly.

2. Logistic Regression

Logistic Regression performed slightly better than KNN. It achieved 60% accuracy, showing it could classify more samples correctly. However, its precision and recall were unbalanced. For label 0, the performance was low, which reduced the overall effectiveness of the model.

3.OpenCV using Neural Network

The neural network's performance was average, with only 50% accuracy. While it managed to balance precision and recall equally, its scores were not significantly better than KNN or Logistic Regression. This indicates the model may need more training epochs, better architecture, or hyperparameter tuning for improvement.

4. **YOLO V4**

YOLO achieved the highest precision (0.8133) and performed well overall. Its accuracy (60%) matched Logistic Regression, but its ability to identify objects more confidently gave it an edge. This makes YOLO suitable for applications requiring higher precision and fewer false positives.

Plots for Evaluation Metrics:

From the plots, we can draw the following conclusions about which algorithm performs the best based on the evaluation metrics:

RMSE (Root Mean Square Error) and MAE Comparison:

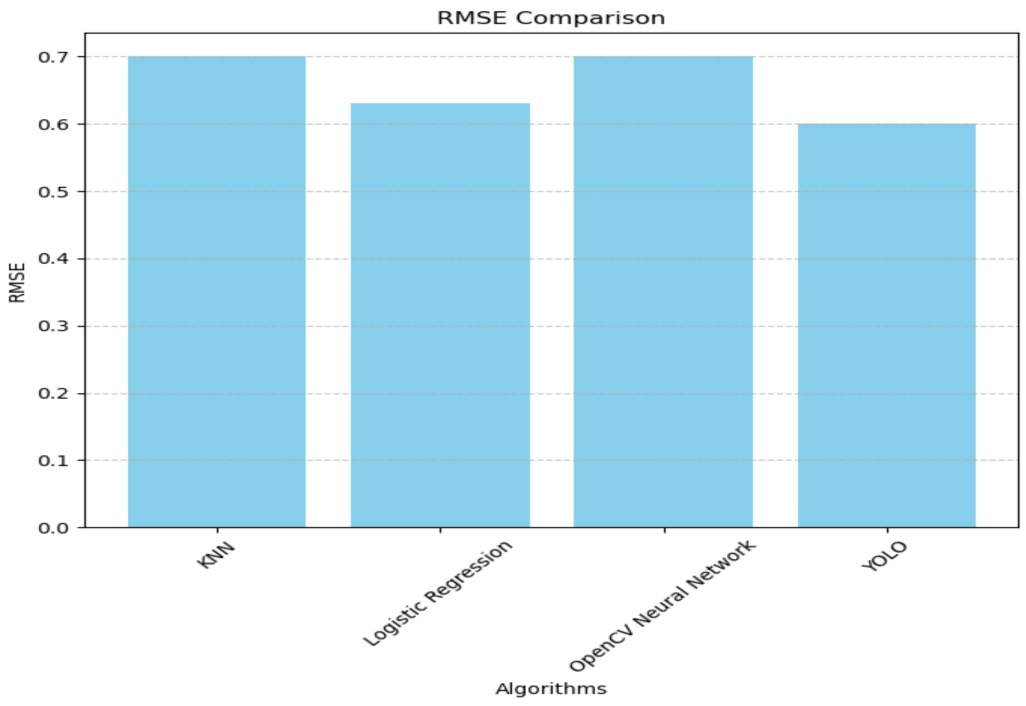

Figure 5: RMSE (Root Mean Square Error Plot)

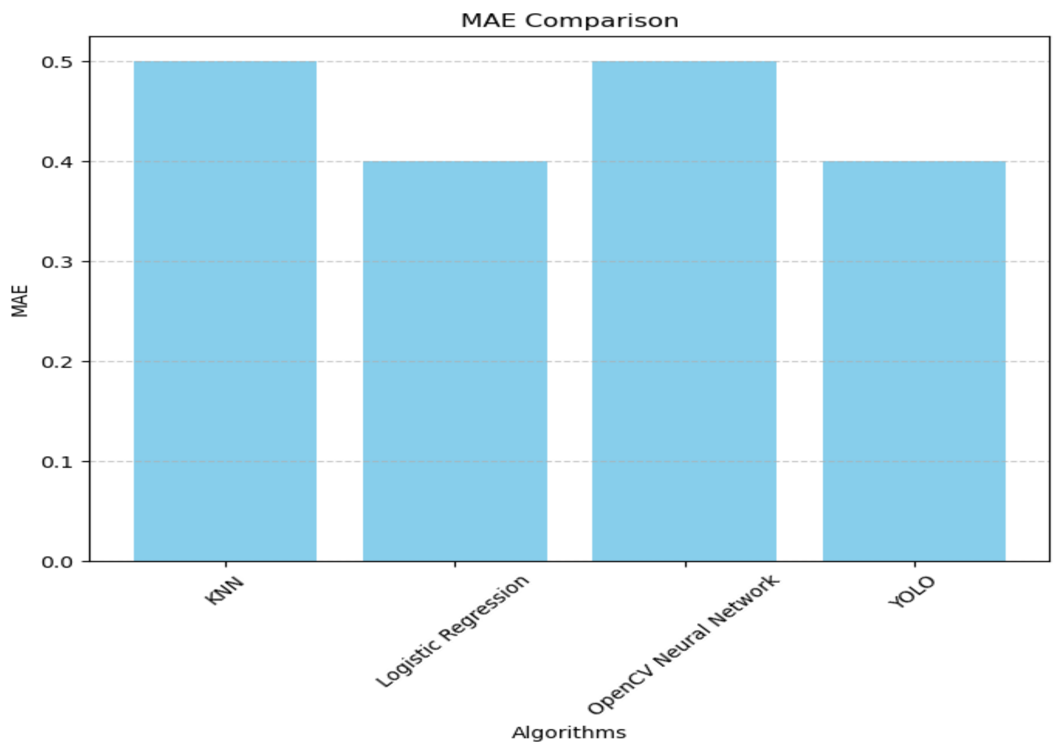

Figure 6: MAE (Mean Absolute Error) plot

YOLO has lower values for RMSE (Root Mean Square Error) and MAE (Mean Absolute Error), which suggests that YOLO's predictions are generally more accurate with fewer errors in comparison to other algorithms.

Precision

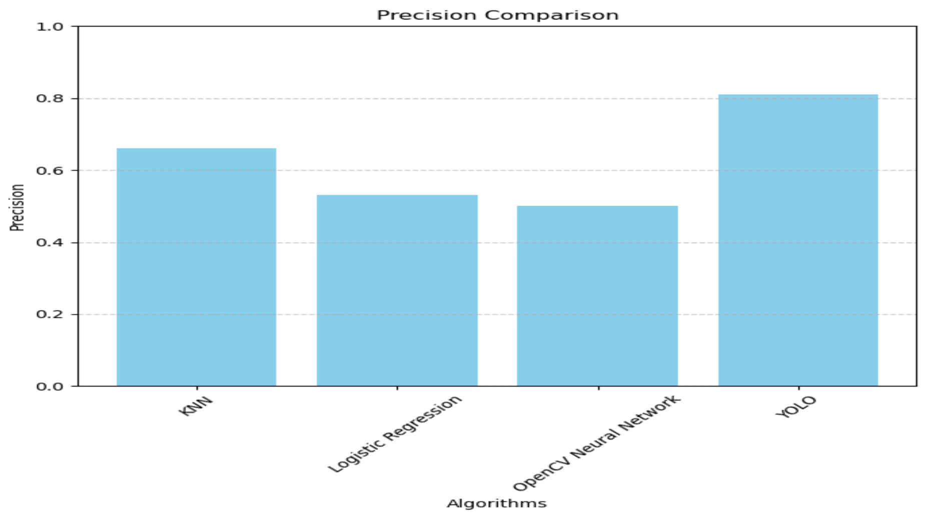

Figure 7: Precision plot

YOLO has the highest precision among all the algorithms. This means it made fewer false positive predictions, correctly identifying more of the positive instances without mistakenly labeling too many negatives as positives.

Recall

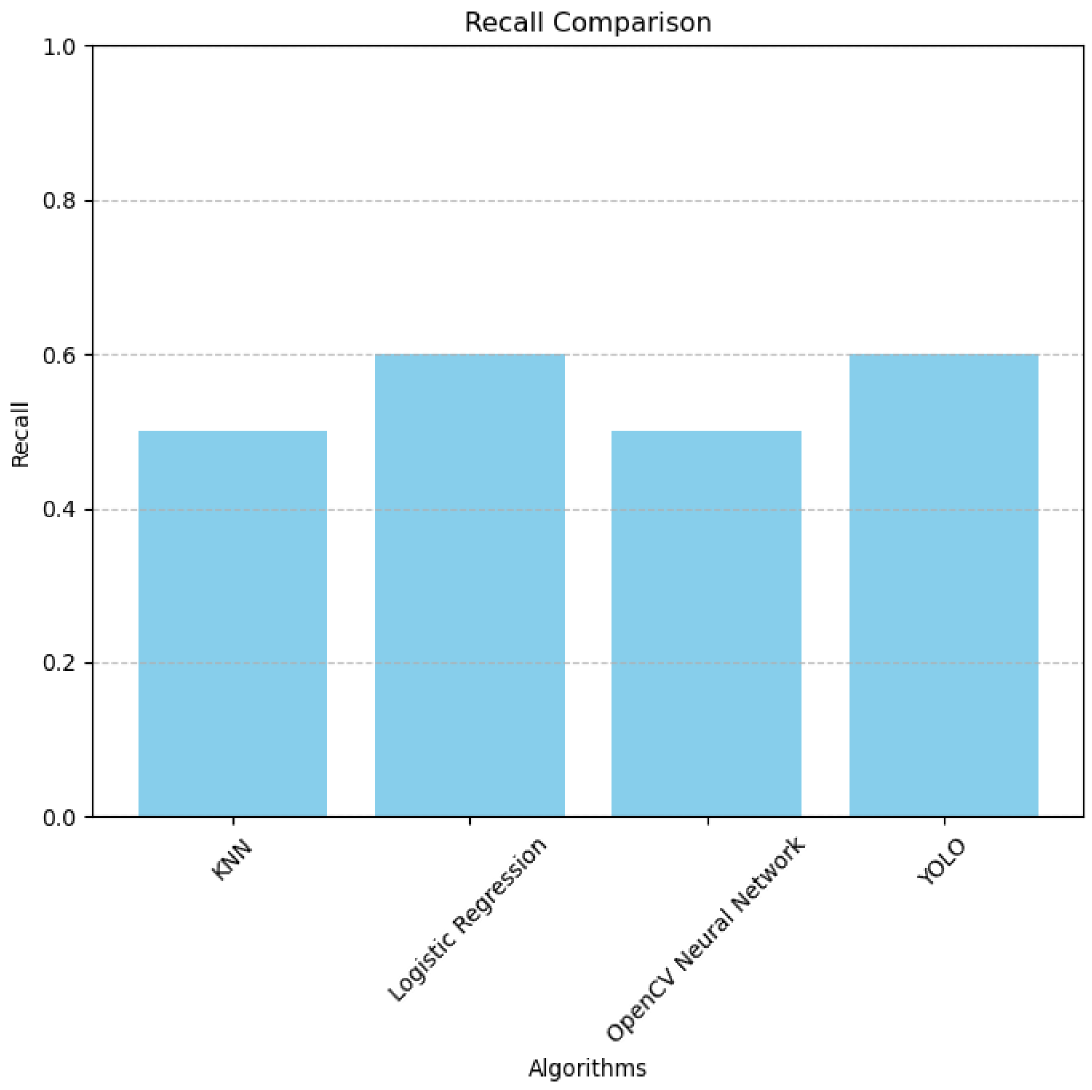

Figure 8: Recall Plot

YOLO and Logistic Regression had the highest recall, indicating they were better at identifying positive instances. YOLO, however, showed better precision, balancing the two metrics effectively.

Accuracy

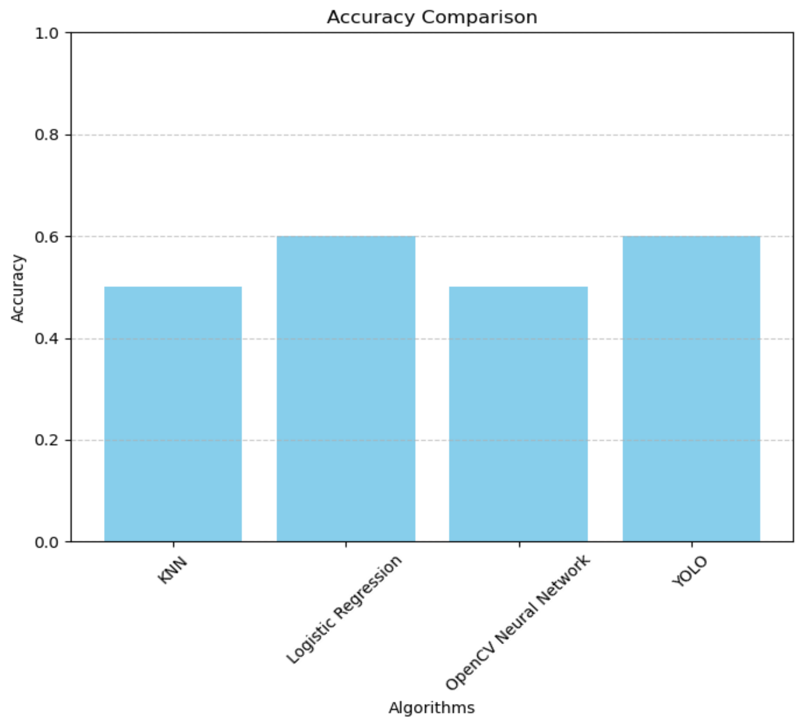

Figure 9: Accuracy Plot

YOLO matches Logistic Regression in accuracy. This shows it correctly classified the same number of samples as Logistic Regression, but with a better balance in its other metrics like precision and recall.

F1 Score

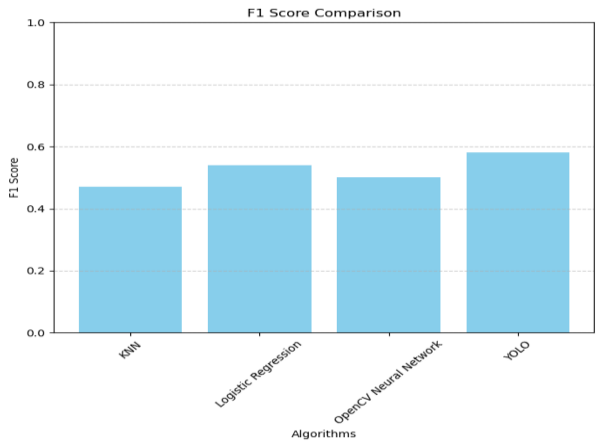

Figure 10: F1 Score Plot

YOLO also performs better in terms of the F1 Score. The F1 Score is a balance between precision and recall. A higher F1 Score means YOLO performed better at identifying both positive and negative instances, without favoring one over the other too much.

Best Performance:

We found that YOLO performed the best among the four models. It achieved the highest precision (0.8133), meaning it made fewer incorrect positive predictions. While its accuracy was similar to Logistic Regression, YOLO outperformed the others in terms of F1 Score, showing a better balance between precision and recall. YOLO's design, which is optimized for object detection tasks, likely helped it perform better in this scenario, making it more effective for identifying patterns in the dataset compared to the other models**.**

Conclusion

From our analysis, we found that YOLO outperformed the other algorithms in several key metrics, including RMSE, MAE, precision, recall, and F1 score. YOLO's ability to make accurate predictions with fewer errors, especially in precision, makes it a strong choice for our task. While Logistic Regression showed competitive accuracy and recall, YOLO's overall performance indicates it handles both false positives and false negatives more effectively. This shows that YOLO might be a better option when we need a model that is both precise and balanced.

For future research, we could explore improving the performance of YOLO even further by fine-tuning its hyperparameters or by using more advanced versions like YOLOv5 or YOLOv7. Additionally, we could experiment with larger datasets to see if YOLO's performance holds up across different kinds of data. It would also be beneficial to investigate how these models perform in real-time applications or in more complex environments to understand their practical usability.

References

[1] FER-YOLO: Detection and Classification Based on Facial Expressions Conference paper. Link: https://link.springer.com/chapter/10.1007/978-3-030-87355-4_3

[2] Facial Emotion Recognition (FER) Through Custom Lightweight CNN Model: Performance Evaluation in Public Datasets by Authors Mustafa Can Gursesli; Sara Lombardi; Mirko Duradoni; Leonardo Bocchi; Andrea Guazzini; Antonio Lanata.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10477992

[3] FER 2013 https://paperswithcode.com/dataset/fer2013

[4] Facial Emotion Recognition using YOLO based Deep Learning Classifier by Sri Vaibhav Mohan Dev Vanamoju; Moturi Veda Vineetha; Hitesh Tekchandani; Pallavi Joshi; Praveen Kumar Shukla; https://ieeexplore.ieee.org/document/10698173

[5] J. R. Barr, L. A. Cament, K. W. Bowyer, and P. J. Flynn. Active clustering with ensembles for social structure extraction. In Winter Conference on Applications of Computer Vision, pages 969–976, 2014

[6] B. R. Beveridge, P. J. Phillips, D. S. Bolme, B. A. Draper, G. H. Givens, Y. M. Lui, M. N. Teli, H. Zhang, W. T. Scruggs, K. W. Bowyer, P. J. Flynn, and S. Cheng. The challenge of face recognition from digital point-and-shoot cameras. In Biometrics: Theory Applications and Systems, pages 1–8, 2013

[7] FACIAL EMOTION DETECTION USING CONVOLUTIONAL NEURAL NETWORKS by Mohammed Adnan Adil https://dspace.library.uvic.ca/bitstream/handle/1828/13388/Adil_Mohammed_Adnan_MEng_2021.pdf?sequence=3

Contributions

This project is completed by –

1. Anushka Kumari
    - Simple Model: K-Nearest Neighbors (KNN).
    - Complex Model: OpenCV using Neural Networks.
    - Data cleaning and preprocessing.
    - Comparative analysis.
    - Overall reporting.
2. Miriyala Sree Harish Kumar
    - Simple Model: Logistic Regression.
    - Complex Model: YOLO v4.
    - Data augmentation.
    - Comparative analysis of results.
    - Visualizing evaluation metrics.
    - Presentation Slides.