*Facial emotion detection from images*

*Vaibhav Chaudhari*

*Rajat Chelani*

*Nicholas Pelletier*

*Shubh Mody*

**Project Objective**

The aim of project is two-fold. Firstly, we intend to devise an analytical framework that categorizes emotions into three distinct categories: comfortable, neutral, and uncomfortable. This framework will aid in the understanding and interpretation of emotions in various contexts. Secondly, we aim to develop a modeling framework that helps individuals with autism to identify emotions from facial expressions. This framework will be designed to assist those who struggle with recognizing and comprehending emotions in social interactions, thereby promoting better communication and socialization skills.

**The Dataset**

The dataset contains 48x48 pixel grayscale images of facial expressions of people belonging to one of seven categories: anger, disgust, fear, happiness, sadness, surprise, and neutral. The dataset is in a CSV (Comma Separated Values) format and contains two columns: "emotion" and "pixels."

The "emotion" column has integers from 0 to 6, representing the seven categories mentioned earlier. The "pixels" column contains a space-separated pixel values of the image, in row-major order. Thus, each image is represented as a one-dimensional array of 2,304 integers.

The dataset has a total of 35,887 examples, divided into a training set of 28,709 examples and a test set of 3,589 examples. The dataset was initially created for a Kaggle competition, where participants were tasked with building a model to classify the facial expressions in the test set.

The top 5 rows of the dataset:

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, number, plot

Description automatically generated

**Data Processing**

**-- Data Categorization:**

First, we categorized did our dataset by grouping the 7 emotion types into 3 classes viz. Comfortable, Uncomfortable and Neutral which became our new categorical column.

The following was the frequency distribution of the number of rows in dataset based on these 3 classes:

Chart, bar chart

Description automatically generatedUncomfortable Comfortable Neutral

**-- Class Balancing:**

From the above graph it was clear that all three classes are imbalanced. In the case of imbalanced classes, the model may achieve a high accuracy score simply by predicting the majority class all the time. This is because the majority class dominates the training data, and the

model learns to optimize for overall accuracy rather than class-specific performance. To tackle this issue, we used under sampling and took 3000 images from all 3 classes to make sure that our predictive model is unbiased which eventually will benefit and improve the model performance as well.

Chart, bar chart

Description automatically generated A close-up of a color chart

Description automatically generated with low confidence

**-- Splitting the Data**

We have split the data into training, validation, and test sets and as there was a label name ‘Usage’ attached for each entity in the dataset itself, we went ahead to create three sets performing data filtering and resetting the index of the subsets.

The Training data had 7217 rows. (~80%)

The validation set had 909 rows. (~10%)

The testing data had 874 rows. (~10%)

A picture containing screenshot, text, rectangle, display

Description automatically generated

The graph above shows the split of each class type (Uncomfortable, Comfortable, Neutral) int the training set which looks almost equally divided. This is a good sign because the training data represents all 3 classes equally and hence won’t be biased towards predicting a particular class.

**-- Data conversion:**

For neural networks, we performed data conversion by devising a function which accepts a dataset as input and processes the pixel values of the images in the dataset. It converts the pixels string of each image into a numpy array of integers and reshapes it to a 48 x 48 matrix. The function returns a numpy array which contains the processed pixel values of all the images in the dataset, and second array containing the integer labels for each image.

We used the devised function to process the training, validation, and testing datasets. It also reshapes the processed pixel values of the datasets into a 4D shape with dimensions (n, 48, 48, 1) where n represents the number of images. The last dimension is set to 1 to indicate that the images are grayscale. Finally, the pixel values are normalized by dividing them by 255 to scale the values between 0 and 1.

For naïve classifiers, we split the pixels string for each image into individual pixel values and arranged them in a 48x48 columnar format using the "split" and "expand" functions. This created a 3 new data frames for training data, testing data, and validation data. The pixel values in the data frames are also normalized by dividing them by 255 to scale the values between 0 and 1.

We extracted the corresponding emotion labels for each image in the dataset and stored them in the data frames for training, testing, and validation data, respectively. These labels are used to train and evaluate the machine learning model.

**Assigning class weights:**

We calculated the class weights for the emotion labels. The value\_counts() method is used to count the number of occurrences of each unique emotion label in the training data. The sort\_index() method is used to sort the labels in ascending order. The resulting counts are then divided by the total number of labels to obtain the proportion of each class in the dataset. These proportions are then converted into a dictionary format using the zip() method and assigned to the variable. This variable will further be passed to the model as a parameter during training to give more weight to underrepresented classes.

**Methods of Analysis**

**K Nearest Neighbor:**

We implemented K-Nearest Neighbors (KNN) algorithm for emotionrecognition. The dataset is split into training and testing sets, consisting of 9000 and 900 samples respectively. Initially, the model is trained on the training set using the default value of K. The accuracy achieved on the test set is 39.05%. To find the optimal value of K, a for loop is used to evaluate the training and testing accuracy for different K values ranging from 1 to 30. The performance of the model is plotted against the number of neighbors, and the best value of K is selected as 14, which provides an accuracy of 42.57%. Finally, the confusion matrix is visualized using a heatmap to analyze the classification results. The results show that the KNN algorithm can be used for emotion recognition with moderate accuracy.

**Decision tree:**

The report describes the implementation of Decision Tree Classifier for emotion recognition. The dataset is split into training and testing sets, consisting of 9000 and 900 samples respectively. The Decision Tree Classifier is trained on the training set and tested on the testing set. The accuracy achieved on the test set is 42.77%, which is not great. The confusion matrix is visualized using a heatmap to analyze the classification results. The results show that the Decision Tree Classifier can be used for emotion recognition with moderate accuracy. However, further optimization and feature engineering techniques can be applied to improve the accuracy of the model.

A screenshot of a graph

Description automatically generated with low confidence

**Support Vector Machine:**

The code performs various operations related to the Support Vector Machine (SVM) algorithm. It first creates an SVM model with a large C value and trains it with data, predicting labels for train data and checking the accuracy of the predicted labels. The data is then checked for linearity and a linear SVM model is selected. An untuned SVM classifier model is created and tested for accuracy, which is found to be low(51.04%). The code then performs a grid search to find the best hyperparameters for the SVM model and declares a new SVM model with these hyperparameters. Finally, the accuracy of the model with the best parameters is checked, which is 53%, and a confusion matrix is plotted. Overall, the code demonstrates the process of selecting and tuning an SVM model for classification.

A screenshot of a graph

Description automatically generated with low confidence

**Random Forest:**

The above code snippet is from an experiment that involves using a Random Forest Classifier to predict the emotions of a given sentence. The code initializes a Random Forest Classifier model, fits it to the training data, and calculates the accuracy of the model. The accuracy of the Random Forest model was found to be 54.56%, which indicates that the model performs decently on the dataset. Further experimentation may be required to improve the accuracy of the model.

A screenshot of a graph

Description automatically generated with low confidence

**Gradient Boosting Classifier:**

Gradient Boosting Classifier is one that our group stumbled upon while searching the SKLearn documentation for possible classifying algorithms to implement. This algorithm uses gradient descent technique to minimize the cost function of the classification equation. After implementing this new classifier, the best accuracy it performed was on standardized pixel data, achieving 51.94% accuracy.

A screenshot of a graph

Description automatically generated with medium confidence

**Neural Network 1:**

The provided code implements a convolutional neural network (CNN) using the TensorFlow library. The network consists of four convolutional layers followed by max-pooling layers and three dense layers. The input shape for the network is (48, 48, 1), which suggests that the network is designed for image classification tasks. The model is trained using the Adam optimizer with a learning rate of 1e-3 for 50 epochs, and the training progress is monitored using both the accuracy and the loss function. The validation data is also used during the training phase to prevent overfitting. Furthermore, the model is

saved at each epoch where the validation accuracy improves using the EarlyStopping and ModelCheckpoint callback functions. The classification accuracy of the model on the validation set reaches up to 59.50% and the testing accuracy is at 55.22%. The loss and accuracy plot showing us how our model is performing and with this model the validation loss is increasing while the training loss decreases. The training accuracy shoots drastically from epoch 15, while the validation accuracy remains stagnant, and the model has overfitting.

A picture containing text, plot, line, diagram

Description automatically generated

A picture containing text, line, plot, diagram

Description automatically generated

From the confusion matrix and the classification report we can observe that the predictions for neutral and comfortable class is significant but for uncomfortable class the proportion of correct predictions are low.

A picture containing screenshot, text, rectangle, diagram

Description automatically generated

**Neural Network 2:**

The third model is a deep convolutional neural network with three convolutional layers, two dense layers, and a softmax output layer. The convolutional layers have 64, 128, and 256 filters of size 1x1, 3x3, and 5x5, respectively, followed by batch normalization, ReLU activation, and dropout. The first dense layer has 128 neurons, followed by batch normalization, ReLU activation, and dropout. The second dense layer has 256 neurons, followed by batch normalization, ReLU activation, and dropout. The output layer has three neurons corresponding to the three classes of emotions, with a softmax activation function. We use Batch normalization in this model to normalize the inputs through layers and use a higher learning rate to efficiently train the model.

The model is trained using the Adam optimizer with a learning rate of 1e-3 and a batch size of 64. The class weights are used to handle class imbalance. The model is trained for 50 epochs and monitored using accuracy as the performance metric. The best model checkpoint is saved based on the validation accuracy.

The classification accuracy of the model on the validation set reaches up to 60.64% and the testing accuracy is at 58.41%. The loss and accuracy plot showing us how our model is performing and with this model the validation loss is increasing while the training loss decreases. The training accuracy shoots drastically from epoch 10, while the validation accuracy remains stagnant, and the model has overfitting.

**Neural Network 3:**

For this model we tried some good practices for kernel size selection for convolution layer. We used odd kernel sizes, which is more efficient because an odd kernel matrix can narrow in on the central pixel of the matrix. We also used max pooling in alternate layers of convolution to down sample the feature map. For training the model we used ImageDataGenerator to generate training data to accompany image rotation, flip, shift, and zoom. This was done to have a better learning for the model. We also introduced an additional callback for training the model. ReduceLROnPlateau was used to give the model a variable learning rate based on validation accuracy trend.

The model has 4 convolution layers, 3 dense layers, batch normalization and maxpolling. The layer activation used is ‘Relu’, and the output activation is ‘softmax’. We have used the adam optimizer with initial learning rate of 1e-3. The model has an early stopping at epoch 40. The validation accuracy achieved was at 75.40%, while the testing accuracy was at 72.16%. from the accuracy and loss plot we can observe that the validation and training loss have a nice decreasing curve, while the training and validation accuracy goes hand in hand. The fluctuations in the validation accuracy are replication of good noise tolerance in model and the similar accuracies show that the model is learning key features from the images and has considerable weight to randomness as well.

A picture containing text, plot, line, diagram

Description automatically generated

A picture containing text, line, plot, diagram

Description automatically generated

From the confusion matrix and the classification report we can observe that the predictions for neutral and comfortable class is quite high and for uncomfortable class the proportion of correct predictions are significant.

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

We tried another CNN model with a few changes. The performance of the model was similar to this model; hence we selected this particular model due to better testing accuracy.

**Best Model Analysis:**

The best model achieved from naïve classifiers was the Tuned Random Forest. The accuracy of the Random Forest model was found to be 54.56%, which compared to our base accuracy of 33.33% is a decent accuracy for a naïve classifier. From the convolution neural networks our model “Neural Network 3” was the best performing model. With a testing accuracy of 72.16%, our model was able to drastically increase the accuracy from the tuned random forest results. The convolution neural network has a high recall for neutral and comfortable emotion predictions. While it faced more difficulties to identify uncomfortable emotions. The wrong predictions had a significant number of limitations that lead to more misclassified classes.

A close-up of a person's face

Description automatically generated

A picture containing text, screenshot

Description automatically generated

**Limitations:**

There are three limitations in the dataset that restricted improvement in many of the machine learning models. These limitations include the presence of watermarks in some of the images, some images being at severe tilts, and bad labeling in the original dataset.

**– Watermarks on Image:**

Some images included watermarks overlapping some pixels in the image. Because there was a watermark, some of the pixels of the image were darkened but did not represent anything within the image. These extra dark, random pixels were extra noise in the dataset that made classification of those images more uncertain.

A close-up of a child's face

Description automatically generated with medium confidence

A close-up of a child's face

Description automatically generated with medium confidence

**-– Angle of Images:**

Images being presented at different angles also posed an issue for standard machine learning models. Although this was able to be corrected partially within the deep learning phase of the project, some pictures that were taken at extreme tilts were still difficult challenges for even neural networks to properly classify.

A close-up of a baby's face

Description automatically generated with medium confidence

A close-up of a person's face

Description automatically generated with medium confidence

**– Wrong Labeling:**

Within the original dataset, some images were inconsistently classified by the original creators. We know this by the inability to classify images into single classes that look almost identical in expression. When classifying images with the same expression, but different class label, the algorithm would predict, with high confidence, the wrong label.

**A close-up of a person's face

Description automatically generated with medium confidence**

**A close-up of a person's face

Description automatically generated**

**References:**

* <https://github.com/rrajj/kNN-for-Facial-Emotion-Recognition/blob/master/ApplyingKNN.ipynb>
* <https://towardsdatascience.com/emotion-detection-a-machine-learning-project-f7431f652b1f>
* <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data?select=icml_face_data.csv>
* <https://www.kaggle.com/code/drcapa/facial-expression-eda-cnn>
* <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>
* <https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363>