DS 5220: Supervised Machine Learning and Learning Theory

Facial Emotion Recognition

Bala Sirisha Sripathi Panditharadhyula

1. SUMMARY

Facial emotion recognition is the process of detecting human emotions from facial expressions. Over the last years, face recognition and automatic analysis of facial expressions has one of the most challenging research areas in the field of Computer vision and has received a special importance. Facial Expression recognition is an important technique and has drawn the attention of many researchers due to its varying applications such in security systems, medical systems, entertainment by utilizing them in Human Computer Interaction (HCI), Emotion analysis, psychological area, virtual reality, videoconferencing, indexing and retrieval of image and video database, image understanding and synthetic face animation.

Face recognition has been around for a long time. Taking a step forward, human emotion, as expressed by the face and felt by the brain, can be mimicked in video, electric signal (EEG), or image form. Modern artificial intelligent systems need to be able to replicate and evaluate reactions from human faces, therefore emotion detection is critical. This can help you make more educated decisions, whether it's about determining intent, promoting deals, or avoiding security threats. Recognizing emotions from photos or video is a simple operation for the human eye, but it's a difficult challenge for machines that necessitates the use of numerous image processing techniques for feature extraction.

Using various models like Decision Trees, Random Forests, KNNs, Gaussian Naïve Bayes and CNNs, we will be developing an application that makes decisions based on the predefined features set in the program and the features that are fed by the input image to predict six types of mood of the given input image- angry, fear, happy, sad, surprise, neutral.

2. DATA

The dataset contains static images sorted into folders pertaining to different expressions of the human face, namely, Surprise, Anger, Happiness, Sad, Neutral, Disgust, Fear.

The dataset is derived from Kaggle https://www.kaggle.com/apollo2506/facial-recognition-dataset.

The folders are split into two super-folders, Training and Testing, so that it can become easier for the end user to configure any model using this data. The dataset consists of 35.3k files in total. The training set consists of 28,273 samples and the testing set consisting of 7,067 samples.

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is centred and occupies about the same amount of space in each image.

2.1 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) aids in the analysis of data sets in order to visualize their features. It enables us to examine what the data can tell us in addition to formal modelling or hypothesis testing.

Exploratory Data Analysis was used to illustrate the distribution of data in relation to each emotion.

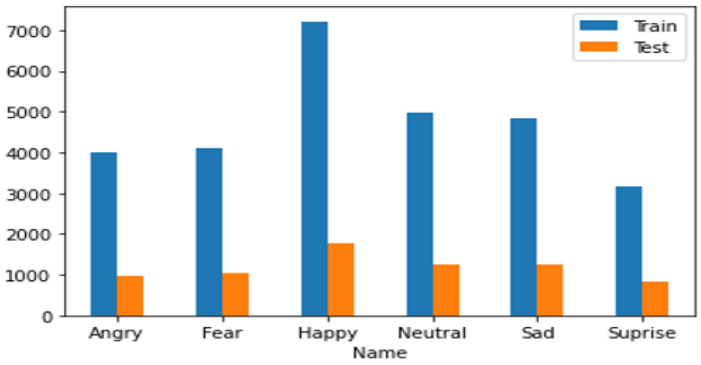

Figure 1: Train-Test Data

- The train data contains:
  7215 Happy images, 4097 Fear images, 4830 Sad images, 3995 Angry images, 4965 Neutral images, 3171 Surprise images.
- The test data contains:
  1774 Happy images, 1024 Fear images, 1247 Sad images, 958 Angry images, 1233 Neutral images, 831 Surprise images.

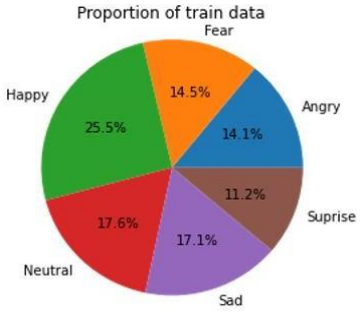

Figure 2: Proportion of Train Data

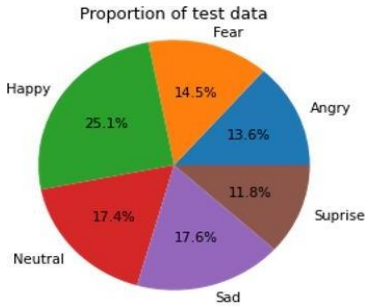

Figure 3: Proportion of Test Data

The above plots show the proportion of test and train data. From the graphs it can be observed that the data is more biased towards the "Happy" emotion which will also be evident in the results.

The sample images that we have retrieved are as follows:

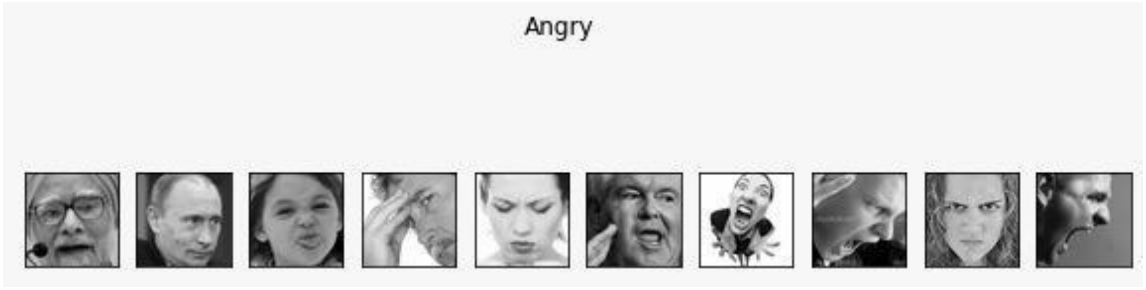

Figure 4: Sample Images per Facial Emotion

3. METHODOLOGY

3.1 DATA PREPROCESSING

We have used the dlib library to determine important facial landmarks in a human face. A pre-trained landmark detector specific to detecting the 68 facial landmarks (a .dat file) in a human face localizing the region around the eyes, eyebrows, nose, mouth, chin and jaw was employed to reach our goal.

As the dataset consisted of faces that were centred and occupied about the same amount of space in each image, we did not see any problem using the dlib function get_frontal_face_detection() for our face (and facial landmark) detection.

An array highlighting the 68 facial landmarks was retrieved and the relevant points on a 48x48 matrix were highlighted. This matrix was then used while fitting the models.

The process of retrieving the 68 facial landmarks was implemented on both, the training and the testing data images.

While implementing the pre-processing, it was noticed we were not able to detect the facial images all images when we were used a simple format detector. While increasing the complexity of the detector was able to solve this problem to quite an extent, the processing time increased drastically.

It was thus decided that these images would be dropped from the training dataset. Our final training dataset (the one the models were fit on) consisted of a total of 19272 images.

The pre-processing for the VGG16 model was completed a little differently. Each pixel was normalized by dividing it with a value of 255

3.2 PROPOSED MODELS

The models we have implemented are as follows:

- K Nearest Neighbours (KNN)
- Decision Tree
- Random Forest
- Gaussian Naïve Bayes
- Convolutional Neural Networks – VGG16

We have also implemented K-fold cross validation on KNN, Decision Tree, Random Forest and Gaussian Naïve Bayes models.

4. IMPLEMENTATION

4.1 K NEAREST NEIGHBOURS

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. KNN involves classifying a data point by looking at the nearest annotated data point, also known as the *nearest neighbour.* It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using KNN algorithm.

To implement KNN, we must load the training as well as test data. Next, we need to choose the value of K i.e., the nearest data points. K can be any integer. For each point in the test data do the following −

- Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- Now, based on the distance value, sort them in ascending order.

- Next, it will choose the top K rows from the sorted array.
- Now, it will assign a class to the test point based on most frequent class of these rows.

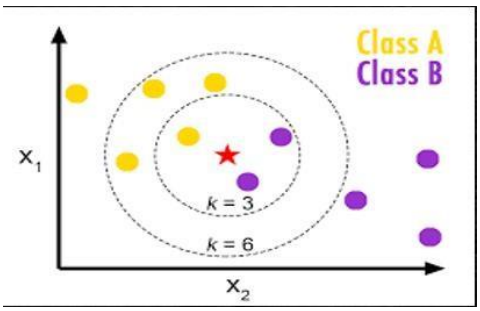

Figure 5: Sample KNN

From the above figure as we can see, when k=3, the star is classified into class B whereas when k=6, the star is classified into class A depending on the neighbours.

We had initially implemented the KNN model with no hyperparameter tuning (the default distance - Euclidean Distance was used). This gave us a very low accuracy and hence we applied hyperparameter tuning for K= 130. We tried K values between 1 to 25 and above 200 which gave a very low accuracy and hence we narrowed it down to 100-150 range where we found the optimal K value of 121 gave us the best accuracy.

The metrics obtained are as follows:

- Precision: 0.3681
- Recall: 0.2886
- F1 score: 0.2606
- Accuracy: 0.3262

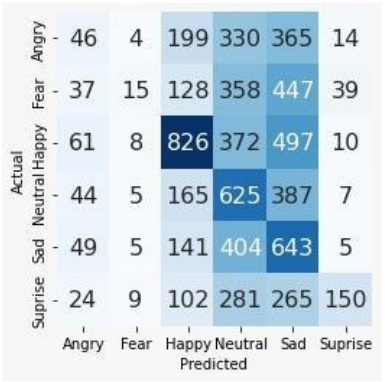

Figure 6: Confusion Matrix of KNN

	precision	recall	f1-score	support
Angry	0.18	0.05	0.08	958
Fear	0.33	0.01	0.03	1024
Happy	0.53	0.47	0.50	1774
Neutral	0.26	0.51	0.35	1233
Sad	0.25	0.52	0.33	1247
Suprise	0.67	0.18	0.28	831
accuracy			0.33	7067
macro avg	0.37	0.29	0.26	7067
weighted avg	0.37	0.33	0.29	7067

Figure 7: Classification Report of KNN

4.2 DECISION TREE CLASSIFIER

In Decision Trees, the data is continuously split according to a certain parameter. It is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches. It creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

The first node is called the *root node*. Starting from the root node, at each node, we use the dataset features to create yes/no questions and continuously split the dataset until we isolate all data points belonging to each class creating a new node. Every time we answer a question, we're also creating branches and

segmenting the feature space into disjoint. All data points on one branch of the tree correspond to answering yes to the inquiry implied by the rule in the preceding node. The remaining data points are in a node on the other branch. The technique attempts to entirely separate the dataset so that all leaf nodes, or those that do not further split the data, belong to a single class. Pure leaf nodes are what they're called. Finally, the algorithm can only assign one class to each leaf node's data points. Because all data points in a pure leaf node have the same class, this is already taken care of. When there are mixed leaf nodes, however, the method assigns the most common class to all data points in thatnode.

To create the next best tree, the algorithm takes a greedy approach. Instead of attempting to make the best overall decision, a greedy strategy makes locally optimum judgments to choose the feature used in each split. Loss functions are used in Decision Trees to evaluate the split depending on the purity of the nodes that result. A loss function, such as Gini Impurity and Entropy, is used to compare the class distribution before and after the split.

The default loss function is the 'Gini Impurity'. It is the one that we have used.

One important task in implementing a decision tree model is parameter tuning. We have used the following parameters for our model:

Random State=7,

Max_Depth=30,

Min_sample_split=50,

Min_samples_leaf=93

Random State: Controls the randomness of the estimator.

Max_Depth: The maximum depth of the tree.

Min_samples_split: Minimum number of samples required to split an internal node

Min_sample_leaf = Minimum number of samples required to be at a leaf node

The evaluation metrics are as follows:

- Precision: 0.2532
- Recall: 0.2356
- F1 score: 0.1818
- Accuracy: 0.2948

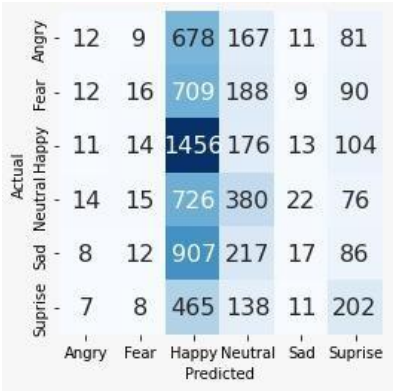

Figure 8: Confusion Matrix for Decision Tree Classifier

	precision	recall	f1-score	support
Angry	0.19	0.01	0.02	958
Fear	0.22	0.02	0.03	1024
Happy	0.29	0.82	0.43	1774
Neutral	0.30	0.31	0.30	1233
Sad	0.20	0.01	0.03	1247
Suprise	0.32	0.24	0.27	831
accuracy			0.29	7067
macro avg	0.25	0.24	0.18	7067
weighted avg	0.26	0.29	0.21	7067

Figure 9: Classification Report for Decision Tree Classifier

4.3 RANDOM FOREST CLASSIFIER

Random Forest Classifier refers to the classification algorithm made up of several decision trees. The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of bagging method is that a combination of learning models increases the overall result.

The hyperparameters of a random forest are quite like those of a decision tree. Fortunately, you may utilize the classifier-class of random forest instead of combining a decision tree with a bagging classifier. You can use the algorithm's regressor to cope with regression tasks with random forest.

While growing the trees, the random forest adds more randomness to the model. When splitting a node, it looks for the best feature from a random subset of features rather than the most essential feature. As a result, there is a lot of variety, which leads to a better model.

The hyperparameter tuning is applied on the following parameters:

N_estimators=100

Random State=9,

Min_Samples_Split = 5,

Min_Samples_Leaf = 1,

Max_Depth = 86

n_estimators: The number of trees in the forest.

Random State: Controls both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split.

Min_samples_split: The minimum number of samples required to split an internal node:

Min_sample_leaf: The minimum number of samples required to be at a leaf node

Max_Depth: The maximum depth of the tree.

The evaluation metrics are as follows:

- Precision: 0.5566
- Recall: 0.2938
- F1 score: 0.2613
- Accuracy: 0.3567

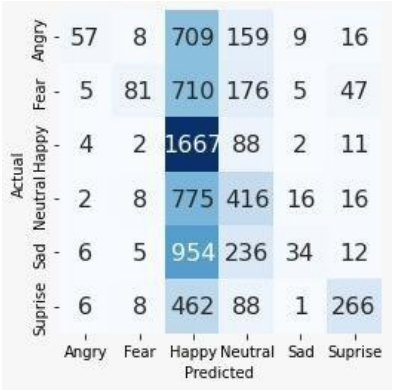

Figure 10: Confusion Matrix for Random Forest Classifier

	precision	recall	f1-score	support
Angry	0.71	0.06	0.11	958
Fear	0.72	0.08	0.14	1024
Happy	0.32	0.94	0.47	1774
Neutral	0.36	0.34	0.35	1233
Sad	0.51	0.03	0.05	1247
Suprise	0.72	0.32	0.44	831
accuracy			0.36	7067
macro avg	0.56	0.29	0.26	7067
weighted avg	0.52	0.36	0.28	7067

Figure 11: Classification Report for Random Forest Classifier

4.4 GAUSSIAN NAÏVE BAYES

Naïve Bayes is basically an algorithm which implements bayes theorem. It assumes that there is conditional independence between each feature pair.

When working with continuous data, one common assumption is that the continuous values associated with each class follow a normal (or Gaussian) distribution. With Gaussian Naïve Bayes, the likelihood of the features is assumed to Gaussian instead of the default Multinomial distribution.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The steps of the algorithm are as follows:

1. Calculate the chance of a given class label being correct.
2. Create a frequency table using the historical data provided.
3. Determine the likelihood probability for each class attribute.
4. Now plug these numbers into the Bayes algorithm to get a posterior probability estimate.
5. The outcome will be determined by the class with the highest probability.

Sometimes it is assumed that the variance is independent of Y (i.e., σi), or independent of Xi (i.e., σk), or independent of both (i.e., σ). This model can be fitted by simply calculating the mean and standard deviation of the points within each label, which is all that is required to construct a distribution of this type.

The evaluation metrics are as follows.

Precision: 0.2962

Recall: 0.2307

F1 score: 0.1708

Accuracy: 0.2308

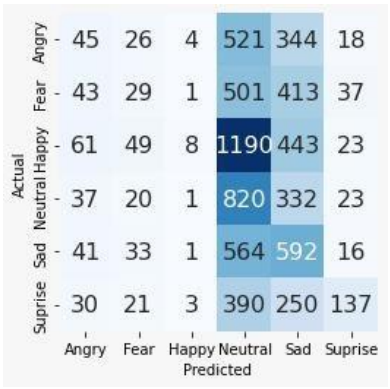

Fig 12: Confusion Matrix for Gaussian Naive Bayes Classifier

	precision	recall	f1-score	support
Angry	0.18	0.05	0.07	958
Fear	0.16	0.03	0.05	1024
Happy	0.44	0.00	0.01	1774
Neutral	0.21	0.67	0.31	1233
Sad	0.25	0.47	0.33	1247
Suprise	0.54	0.16	0.25	831
accuracy			0.23	7067
macro avg	0.30	0.23	0.17	7067
weighted avg	0.30	0.23	0.16	7067

Fig 13: Classification Report for Gaussian I Bayes Classifier

We had tried hyperparameter tuning for Gaussian Naïve Bayes (by using var_smoothing which is the portion of the largest variance of all features that is added to variances for calculation stability. It widens the curve to account for more samples away from the mean). While the accuracy showed little to no difference, we noticed that the model was predicting values primarily as “Happy” or “Neutral”.

4.5 CONVOLUTION NEURAL NETWORKS – VGG16

VGG16 is a deep neural network and like the name suggests it has 16 convolution networks. It is one of the most advanced and best CNN models till date for image processing.

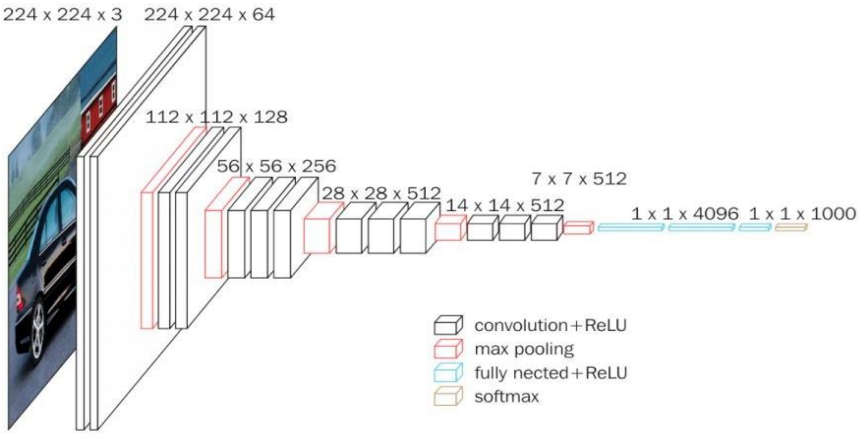

Figure 14: Architecture of VGG16

In a default VGG16, the input to the cov1 layer is a 224x224 RGB picture with a fixed size. The image is passed through a stack of convolutional (conv) layers with an extremely narrow receptive field: 3x3 (the smallest size to capture the notions of left/right, up/down and center). It also uses 1x1 convolution filters in one of the setups, which may be thought of as a linear transformation of the input channels (followed by non-linearity). The convolution stride is set to 1 pixel, and the spatial padding of conv. layer input is set to 1 pixel for 3x3 conv. layers so that the spatial resolution is kept after convolution. Five max-pooling layers, which follow part of the conv. layers, do spatial pooling (not all the conv. layers are followed by max-pooling). Max-pooling is done with stride 2 over a 2x2 pixel window.

Model Implemented:

```
base_model = VGG16(
        weights=None,
        include_top=False,
        input_shape=IMG_SIZE + (3,)
    )
```

```
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(1000, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES, activation="softmax"))
```

The non-linear Rectified linear unit (ReLU) – activation function is present in all hidden layers. The model was trained for 20 epochs. We found the training accuracy to be 0.9521

The metrics obtained are as follows:

- Precision: 0.5783
- Recall: 0.5825
- F1 score: 0.5803
- Accuracy: 0.60

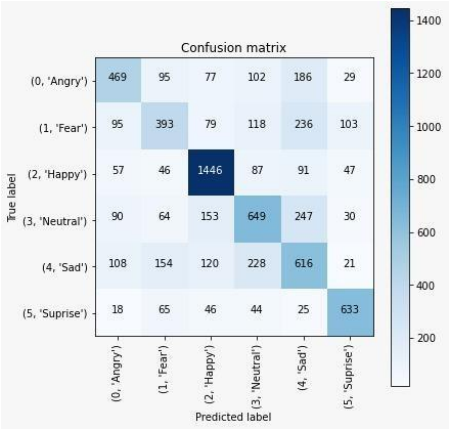

Figure 15: Confusion Matrix for VGG16

5. RESULTS

	KNN	Decision Tree	Random Forest	Gaussian Naïve Bayes	VGG16
Accuracy	0.3262	0.2948	0.3567	0.2308	0.6
Precision	0.3618	0.2532	0.5566	0.2962	0.5783
Recall	0.2886	0.2356	0.2938	0.2307	0.5825
F1	0.2606	0.1818	0.2613	0.1708	0.5803

6. CONCLUSION

- The VGG16 model gave the best test accuracy of 60%
- This model was trained for 20 epochs and had given a train accuracy of 95.21%
- The major drawback of the model is the time it trains to train and process the model.
- The second-best model (one with lower training time) as from the models used was the Random Forest model which had a test accuracy of 35%

7. FUTURE WORK

In this project we have applied basic feature pre-processing with 68 facial landmarks. In the future, we plan to apply more complex models like Viola Jones algorithm and Active Shape model for better fitting of our models. We also intend to balance out the training data so that the models fitted are not biased towards a single facial emotion (as seen in the models fitted on the data here, the models are biased towards the facial emotion 'Happy'). We also intend to analyse in detail the Gaussian Naïve Bayes model to understand that why on applying var_smoothing the images are only classified under the emotions of "Happy" and "Neutral". Another aspect we would like to work on is removing the overfitting from the VGG16 model.

8. REFERENCES

1. https://link.springer.com/article/10.1007/s42979-020-0114-9
2. https://www.analyticsvidhya.com/blog/2021/06/build-vgg-net-from-scratch-with-python/
3. https://iq.opengenus.org/gaussian-naive-bayes/#:~:text=Gaussian%20Naive%20Bayes%20is%20a,distribution%20and%20supports%20continuous%20data.&text=Naive%20Bayes%20are%20a%20group,technique%2C%20but%20has%20high%20functionality.
4. https://machinelearningmastery.com/naive-bayes-for-machine-learning/
5. https://ieeexplore.ieee.org/abstract/document/4409066
6. https://pages.cms.hu-berlin.de/EOL/geo_rs/S09_Image_classification2.html
7. https://towardsdatascience.com/a-visual-guide-to-decision-trees-26606e456cbe
8. https://arxiv.org/abs/1806.06988
9. https://builtin.com/data-science/random-forest-algorithm
10. https://iq.opengenus.org/gaussian-naive-bayes/