Constructing A H.O.G-Feature Based Human Emotion Recognition System

1st Nguyen Nguyen Khoi
Computer Science
Univeristy of Information Technology
Ho Chi Minh City, Viet Nam
21521009@gm.uit.edu.vn

2nd Phi Quang Dat
Computer Science
Univeristy of Information Technology
Ho Chi Minh City, Viet Nam
21520711@gm.uit.edu.vn

***Abstract*—Emotion recognition plays a crucial role in various domains, including healthcare and education. In this paper, we examine Histogram-of-Oriented-Gradient (H.O.G) features and four machine learning algorithms: K-Nearest Neighbors (KNN), Random Forest, Softmax Regression and Support Vector Machines (SVM) which are used to predict human's expression. Experiment is implemented on FER2013 dataset. The results show that SVM (RBF kernel) outperforms other algorithms in terms of accuracy with 50.8% of photos are assigned with true emotion labels.**

I. Introduction

Emotion recognition performs an important role in mental health improvement, education & personalized learning and health monitoring. Firstly, early detection of patients' depression, anxiety and stress could be carried out by emotion recognition system. This information could be analyzed by mental health professionals to tailor interventions, provide personalized therapies and track the progress of patients over time. Secondly, emotion recognition technology has the potential to revolutionize education by enabling personalized and adaptive learning experiences. Emotion-aware tutoring systems can analyze students' emotional responses, adjust their approach accordingly and offer personalized guidance and support. Finally, emotion recognition technology could be utilized for health monitoring, particularly in contexts where emotional well-being plays a crucial role. For instance, in elderly care settings, emotion recognition systems can detect signs of distress, loneliness, or discomfort in individuals. This can prompt caregivers to provide immediate attention and support, improving the quality of care. To build an emotion recognition system, we extract features using Histograms of oriented gradients (HOG) and apply them to some machine learning models. We examine four classification algorithms : K-Nearest Neighbors (KNN), Random Forest, Softmax Regression and Support Vector Machines (SVM) for this research.

Nguyen Nguyen Khoi is responsible for HOG, Random Forest and SVM. Phi Quang Dat is responsible for KNN, Softmax Regression and a demonstration applied to real-world data.

Our problem's input is a front face photo with full eyes, mouth, nose and forehead. The face could be captured at various angles. However, capture device and face have to be ensured to stand in a straight line parallel to the ground. The output of our problem is 1 out of 7 emotion labels: suprised, sad, neutral, happy, fearful, disgusted and angry.

II. Extracting features

A. Idea

Histogram of Oriented Gradients, also known as HOG, is a feature descriptor used in computer vision and image processing. **The technique counts occurrences of gradient orientation in the localized portion of an image**. The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

B. Steps to calculate H.O.G features

- 1. Preprocessing the image: the image might need to be resized. In our project, all input images are resized to 128x128.
- 2. Calculate the horizontal and vertical gradients by filtering the image with the following kernels:

-1	0	1

-1
0
1

Next, we can find the magnitude and direction of gradient using the following formula:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

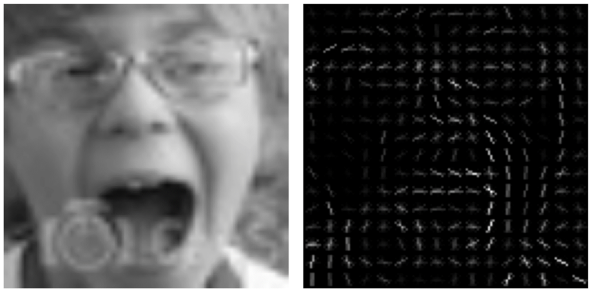
- 3. Calculate Histogram of Gradients in 8×8 cells
- 4. Block normalization: gradients of an image are sensitive to overall lighting. If the image is made darker by

dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half.

Ideally, our descriptor needs to be independent of lighting variations. In other words, normalizing the histogram is necessary.

- 5. Calculate the H.O.G feature vector

Here is an example of extracting H.O.G features. The left image is a front face photo from FER2013 dataset, which is used in our project. The right one illustrates main patterns of the face, relying on H.O.G feature vector.



III. Classifiers

A classifier is a machine learning model that is used to discriminate different objects based on certain features.

A. K-Nearest Neighbors

1) Definition: K-Nearest Neighbors (K-NN) classifies a given sample based on the groups of its surrounding 'k' number of neighbors. It uses feature similarity to predict the cluster that the new point will fall into.

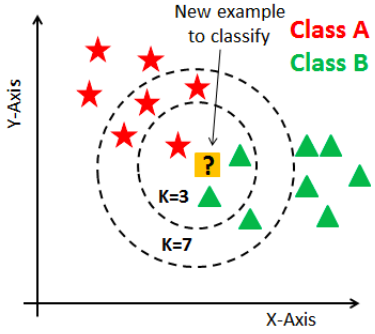


Fig. 1: Example of K-NN.

2) Properties: The properties of KNN are that they are a lazy learning algorithm and a non-parametric method. It is used for classification tasks

a) Lazy learning: Lazy learning means the algorithm takes almost zero time to learn because it only stores the data of the training part (no learning of a function). The stored data will then be used for the evaluation of a new query point.

K-NN belongs to a subcategory of non-parametric models that is described as instance-based learning. Models based on instance-based learning are characterized by memorizing the training dataset, and lazy learning is a special case of instance-based learning that is associated with no (zero) cost during the learning process.

b) Non-parametric: The non-parametric method refers to a method that does not assume any distribution. Therefore, K-NN does not have to find any parameter for the distribution. While in the parametric method, the model finds new parameters, which in turn will be used for the prediction purpose. The only hyperparameter (provided by the user to the model) K-NN has is K, which is the number of points that needs to be considered for comparison purpose.

3) Metric for distance computation: There are many methods to measure distance such as Hamming Distance, Euclidean Distance, Manhattan Distance or Minkowski Distance, but the most popular one is the Euclidean distance (for smaller dimension data) and Manhattan distance.

a) Euclidean Distance: Euclidean Distance is defined as crow flies. It calculates the distance between two real-valued vectors. Euclidean distance is calculated as the square root of the sum of the squared differences between the two vectors.

$$\begin{aligned} d(p,q) &= d(q,p) \\ &= \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\ldots+(q_n-p_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \end{aligned} \tag{1}$$

In (1), q and p are two different points which have n dimensions

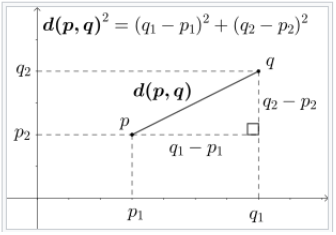


Fig. 2: Example of Euclidean Distance.

b) Manhattan: Manhattan Distance also called the Taxicab distance or the City Block distance, calculates the distance between two real-valued vectors. It is perhaps more useful to vectors that describe objects on a uniform grid, like a chessboard or city blocks. The taxicab name for the measure refers to the shortest path that a taxicab would take between city blocks (coordinates on the grid). Manhattan distance is calculated as the sum of the absolute differences between the two vectors.

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}|q_i-p_i|} \tag{2}$$

In (2), q and p are two different points which have n dimensions

Fig. 3: Example of Manhattan distance and Euclidean distance.

In Fig. 3, the blue line denotes the Manhattan distance between two points, whereas the red poly line represents the Euclidean distance between those.

4) Process: In the training phase, the model will only store the data points.

In the testing phase, all the distance from the query point to the other points from the training phase is calculated to classify each point in the test dataset.

5) Pros and cons:

a) Pros:

- The algorithm is simple and easy to comprehend and implement.
- The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms, as there is no need to train a model for generalization. This is why K-NN is known as the simple and instance-based learning algorithm.
- K-NN does provide a relatively high accuracy compared to other algorithms.
- K-NN could be useful in case of nonlinear data. It could be used with the regression problem. Output value for the object is computed by the average of k closest neighbors value.

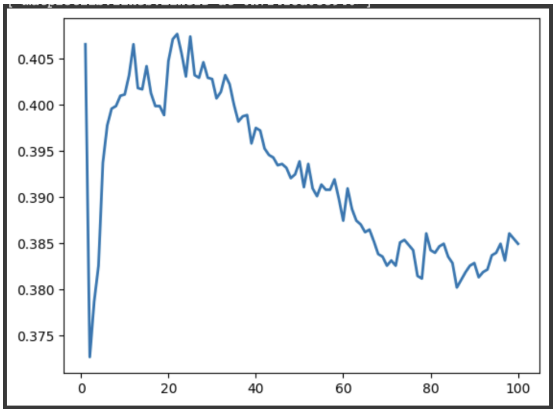
b) Cons:

- 'Time complexity' and 'space complexity' is enormous, which is a major disadvantage of K-NN.
  Time complexity refers to the time the model takes to evaluate the class of the query point, while space complexity refers to the total memory used by the algorithm. If we have n data points in training and each point is of m dimension –¿ Then time complexity is of order O(nm), which will be huge if we have higher dimension data. Therefore, K-NN is not suitable for high dimensional data.
- Another disadvantage is called 'The curse of dimensionality'.
  It is important to mention that K-NN is very susceptible to overfitting due to the curse of dimensionality. The curse of dimensionality describes the phenomenon where the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset. Intuitively, we can think of even the closest neighbors being too far away in a high-dimensional space to give a good estimate.

6) Hyperparameters: K-Nearest Neighbors only has one hyperparameter 'k', which stands for the number of neighbors required for computation purpose. One thing to notice here, if the value of K is even, it might create problems when taking a majority vote because the data has an even number of classes. Therefore, choose K as an odd number when the data has an even number of classes and an even number when the data has an odd number of classes.

7) Choosing k: Choosing the value of k is an important decision as it can significantly impact the performance and accuracy of the model

In this project, we consider a range of values for k from 1 to 100. The line plot below represents the accuracy of KNN model on the testing set of the dataset (FER2013) with different values of k.



The maximum accuracy achieved is 0.40763 at k = 22. Maybe there are unseen values of k that produce higher accuracy but the time is not enough for us to explore more.

B. Softmax Regression

1) Idea: The main idea of Softmax Regression, also known as Multinomial Logistic Regression, is to extend binary logistic regression to handle multi-class classification problems.

In supervised-learning problems, we are given a training set of n labeled samples: D = ($\mathbf{x}_1$, y_1), ($\mathbf{x}_2$, y_2), ... , (x_n, y_n), where $\mathbf{x}_i$ is a m-dimensional vector that contains the features of sample i and y_i represents the label of that sample.

Softmax Regression outputs probability of a sample belonging to each of k classes and assigns it a label based on these probabilities. Therefore, softmax regression is called **probabilistic classifier**

2) Calculating probabilities: Given a sample (**x**, y), the softmax regression model outputs a vector of probabilities $\mathbf{p} = (p_1, \ldots, p_k)$, where p_i represents the probability that the sample belongs to class i:

$$p_i = P(y = y_i|x)$$

The probabilities must sum to 1:

$$\sum_{i=1}^{k} p_i = 1$$

In softmax regression, we choose one of the probabilities as a reference (let's say p_k), and assume that the log-odds ratio between each probability p_i and p_k is a linear combination of the input features. In other words, we can write the lo between p_i and p_k as a dot product of some weight ve and the feature vector **x**:

$$\log \frac{p_i}{p_k} = \mathbf{w}_i{}^T \mathbf{x}$$

Note that in softmax regression we have a separate vector of parameters w_i for each class i. The set of all the parameters of the model is typically stored in a matrix W of size $(m+1)k$, obtained by concatenating $w_1, \ldots, w_k$ into columns:

$$W = \begin{pmatrix} | & | & & | \\ \mathbf{w}_1 & \mathbf{w}_2 & \ldots & \mathbf{w}_k \\ | & | & & | \end{pmatrix}$$

Fig. 4: The matrix of parameters

By taking the exponent of both sides of the log-odds equation we get:

$$\frac{p_i}{p_k} = e^{\mathbf{w}_i^T \mathbf{x}}$$
$$p_i = p_k e^{\mathbf{w}_i^T \mathbf{x}}$$

Since all k probabilities sum to 1, we could write:

$$p_k = 1 - \sum_{i=1}^{k-1} p_i = 1 - \sum_{i=1}^{k-1} \left(p_k e^{\mathbf{w}_i^T \mathbf{x}} \right)$$
$$p_k + p_k \left(\sum_{i=1}^{k-1} e^{\mathbf{w}_i^T \mathbf{x}} \right) = 1$$
$$p_k \left(1 + \sum_{i=1}^{k-1} e^{\mathbf{w}_i^T \mathbf{x}} \right) = 1$$
$$p_k = \frac{1}{1 + \sum_{i=1}^{k-1} e^{\mathbf{w}_i^T \mathbf{x}}}$$

We could now use the expression for p_k to find all the other probabilities:

$$p_i = p_k e^{\mathbf{w}_i^T \mathbf{x}} = \frac{e^{\mathbf{w}_i^T \mathbf{x}}}{1 + \sum_{j=1}^{k-1} e^{\mathbf{w}_j^T \mathbf{x}}}, \qquad 1 \leq i < k$$

Since all k probabilities must sum to 1, the probability p_k is completely determined once all the rest are known. Therefore, we have only k1 separately identifiable vectors of coefficients. This means that we can arbitrarily choose $w_k = 0$ to make sure that $e^{w^t{}_k} = 1$. This in turn allows us to write the equations above more compactly as follows:

$$p_i = \frac{e^{\mathbf{w}_i^T \mathbf{x}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^T \mathbf{x}}}, \qquad 1 \leq i \leq k$$

3) Regularization: Regularization is a technique used to avoid overfitting in machine learning models. It does this by adding a penalty term to the objective function (also called the loss function or error function) that the model is trying to minimize.

The objective function measures the error or difference between the predicted output of the model and the true output. In logistic regression, the objective function is typically the cross-entropy loss, which measures the difference between the predicted probability of the positive class and the true label.

By adding a penalty term to the objective function, regularization helps to reduce the complexity of the model and prevent it from fitting the training data too closely. The penalty term is a hyperparameter that controls the strength of the regularization.

Some types of regularization are:

L1 (Lasso) regularization

- Adds a penalty term to the objective function equal to the absolute value of the coefficients.
- This leads to a sparse model, where many of the coefficients are exactly equal to zero.
- L1 regularization is useful for feature selection because it can automatically identify and remove unnecessary or redundant features from the model.

L2 (Ridge) regularization

- Adds a penalty term to the objective function equal to the square of the coefficients.
- This leads to a model with all coefficients close to zero, but not necessarily equal to zero.
- L2 regularization is less prone to overfitting than L1 regularization and is often used as a default choice.

Elastic Net regularization

- Combines L1 and L2 regularization by adding a penalty term to the objective function that is a combination of the absolute value and square of the coefficients.
- This leads to a model with some coefficients equal to zero and some close to zero.
- Elastic Net could be useful when there are correlated features in the data and Lasso is prone to selecting only one of them.

4) Choosing hyperparameters: In this project, penalty l2 and regularization strength C=1 are chosen to examine the performance of Softmax Regression model on human emotion recognition task.

C. Random Forest

1) Definition: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

2) Decision Trees: Decision trees are among the most popular machine learning algorithms and are also the building blocks of Random Forests. The algorithm of Decision Trees is as follows, For more formal definition look at

- Create a root node t_0 that contains all learning data L
- Set $currentNode = t_0$
- Repeat until stopping criteria is met:
  - Find best split s^* in all variables which maximizes impurity decrease
  - Label the $currentNode$ with best split variable and its value
  - Divide the available learning data L into L_l and L_r
  - Create nodes t_l and t_r which contains data L_l and L_r respectively
  - Repeat with $currentNode = t_l$ and data L_l
  - Repeat with $currentNode = t_r$ and data L_r

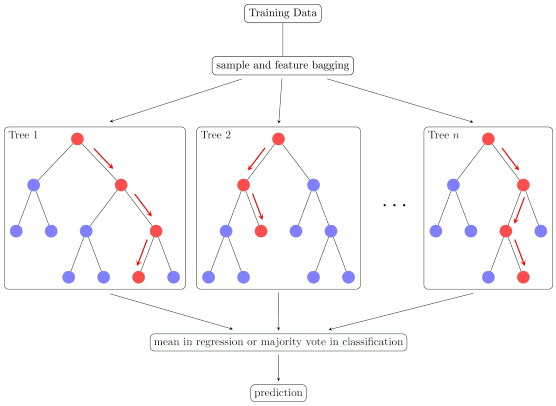


Fig. 5: Diagram of the random forest algorithm (Breiman 2001).

3) Random Forest Algorithm: Random Forests are among the most powerful and successfully used machine learning algorithms. Main idea of Random Forests is to build multiple Decision Trees, where each tree is built with subset of variables and bootstrapped data. This ensemble of trees is called Random Forests, it makes prediction by taking outputs of all trees and then averaging the result if its a regression problem or choosing the most popular answer if its a classification problem. The algorithm for creating a Random Forest is as follows, For more formal definition look at

Set variable $M =$ maximum number of trees
Set variable $p \leq k$, where k is total number of variables
Set variable $\widetilde{N} = 0.632N$ where N is total number of samples
For $1..., M-1, M$:
  Create a root node t_0 that contains bootstraped learning data L
  Randomly choose p variables Set $currentNode = t_0$
  Repeat until stopping criteria is met:
    Find best split s^* in p variables (randomly chosen in previous step) which maximizes impurity decrease
    Label the $currentNode$ with best split variable and its value
    Divide the available learning data L into L_l and L_r
    Create nodes t_l and t_r which contains data L_l and L_r respectively
    Repeat with $currentNode = t_l$ and data L_l
    Repeat with $currentNode = t_r$ and data L_r

4) Complexity Analysis: Random Forests are built upon Decision Trees so their complexity analysis are very similar complexity analysis. The differences are:

- There are M number of trees instead of only one tree
- There are p number of variables in each tree instead of k, where $p \leq k$ and k is total number of variables
- Each tree is built using $\widetilde{N}$ number of samples, where $\widetilde{N}$ is 63.2% of total number of samples N

Considering these differences the complexities of Random Forests turn out to be:

$$\textit{Best Case:} \quad O(Mp\widetilde{N}log^2\widetilde{N}) \tag{3}$$

$$\textit{Worst Case:} \quad O(Mp\widetilde{N}^2log\widetilde{N}) \tag{4}$$

$$\textit{Avg. Case:} \quad O(Mp\widetilde{N}log^2\widetilde{N}) \tag{5}$$

D. Support Vector Machines

1) Idea: The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N - the number of features) that distinctly classifies the data points.

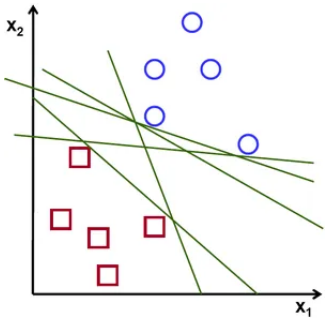


Fig. 6: Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

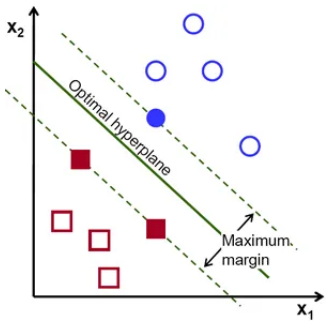


Fig. 7: Optimal hyperplane

2) Hyperplane: Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

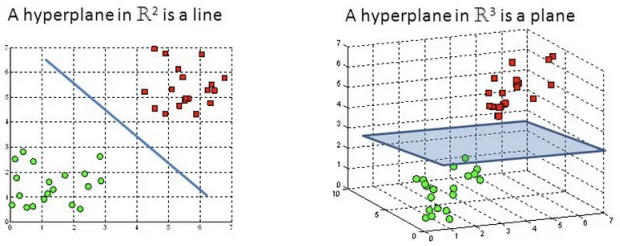


Fig. 8: Hyperplanes in 2D and 3D feature space

3) Support vectors: Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

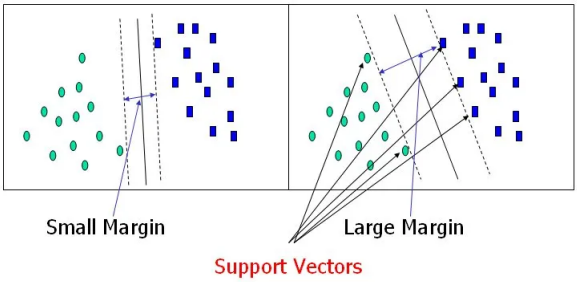


Fig. 9: Support vectors

4) Regularization: Consider the case where a single outlier breaks the linear separability of the data

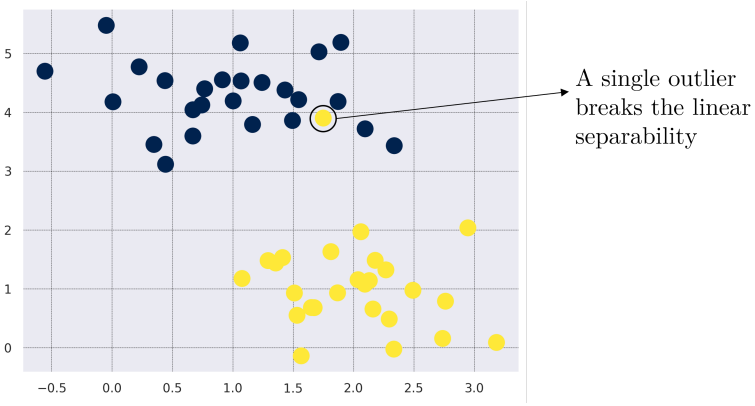
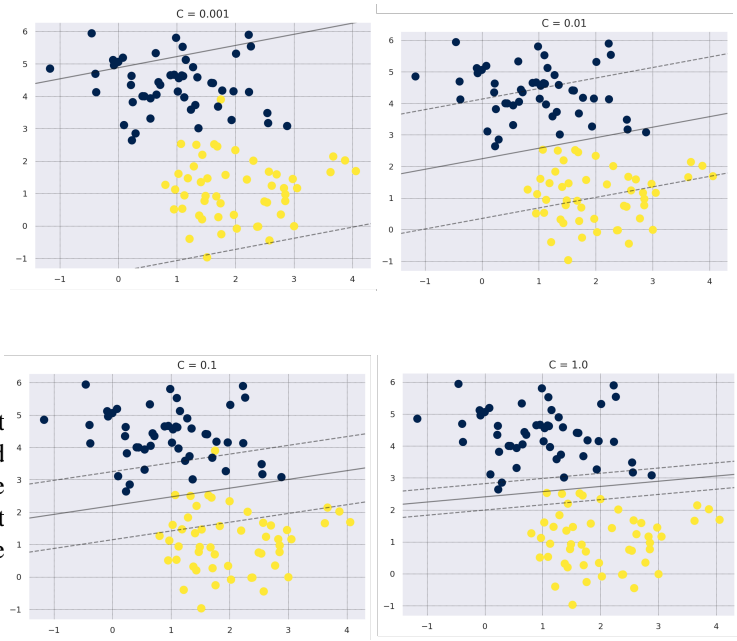


Fig. 10: Linear separability broken by a point

Our goal now is not to make zero classification mistakes, but to make as few mistakes as possible. By adding a **regularization** parameter to the classification task, we could control the width of the ”soft margin” (how SVM handles errors). This parameter is usually denoted in C.



5) Kernels: The data shown in the examples above are (almost)**linearly separable**, which means a line, a plane or a hyperplane could be found to separate the data. Interestingly, SVM could even work with a **linearly inseparable** dataset and for this, we use “Kernel Trick” which makes it easier to classifies the points. Suppose we have a dataset like this:

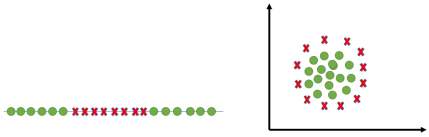


Fig. 11: Examples of linearly inseparable data

Obviously, finding a line or hyperplane which could correctly classify the points seems impossible. Therefore, we could try converting this feature space to a higher dimensional one which will allow us to find a decision boundary that clearly divides the data points. Some quadratic functions could help us do this. These functions are called kernels.

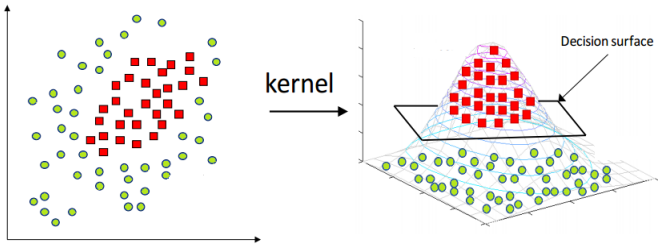


Fig. 12: Kernel trick

6) Different kernels: Some kernel functions are given below:

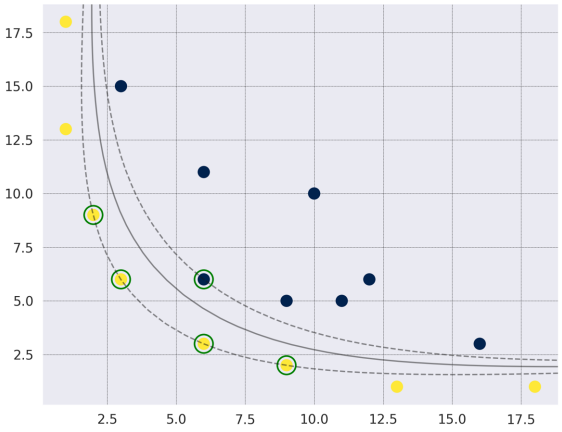


Fig. 13: Polynomial kernel

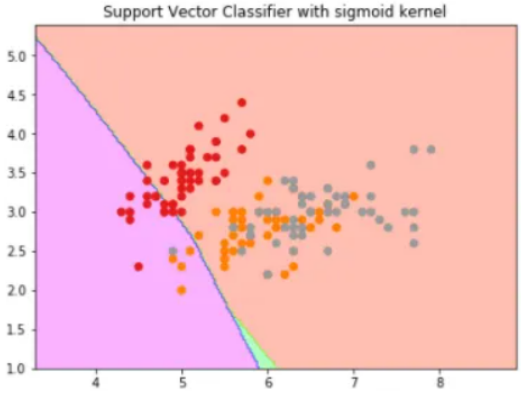


Fig. 14: Sigmoid kernel

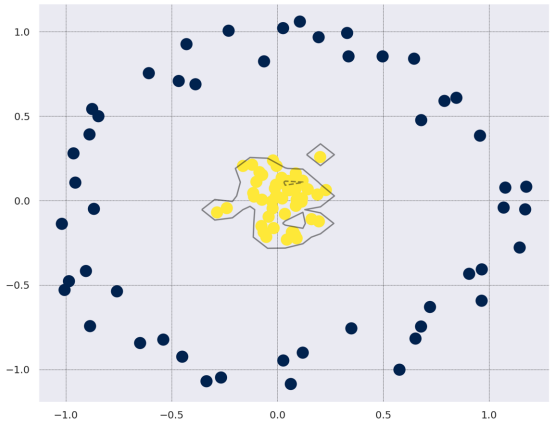


Fig. 15: Radial Basis Function (RBF) kernel

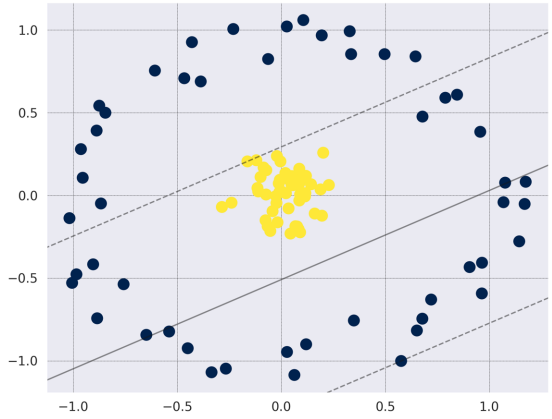


Fig. 16: Linear kernel

7) Choosing hyperparameters: In this project, regularization C and kernel are chosen as 1 and RBF respectively. Maybe there are other values that produce higher accuracy on testing set but we do not have enough time to conduct more experiments.

IV. Dataset

The FER2013 dataset, created by Pierre-Luc Carrier and Aaron Courville, serves as the primary dataset for the human emotion recognition project. This dataset consists of grayscaled front face photos, each with a resolution of 48x48 pixels. The images are distributed among seven distinct classes representing different emotional states: surprise, sad, neutral, happy, fear, disgust, and angry.

The dataset comprises a total of 35,887 images. Specifically, there are 28,709 images allocated for training purposes, while the remaining 7,178 images are designated for testing and evaluation. This division enables researchers to assess the models' generalization ability and performance on unseen data.

Each image in the FER2013 dataset is labeled with one of the seven emotion classes, allowing for supervised training and validation of the emotion recognition models. These labels

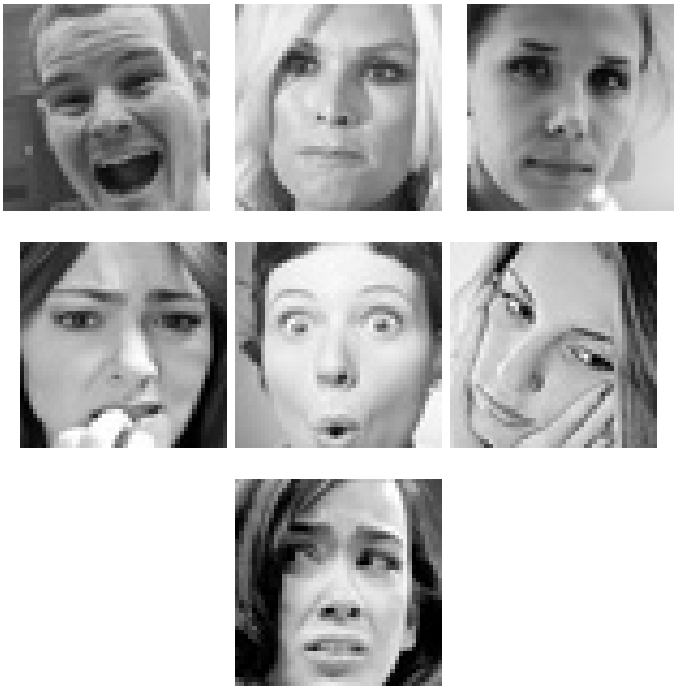


Fig. 17: Some samples in FER2013 dataset

are crucial for training the models to accurately predict the emotional states depicted in the input images.

The table below represents the distribution of dataset over 7 classes of emotion:

TABLE I: Number of photos for each emotion label

	Training	**Testing**
happy	7215	1774
neutral	4965	1233
sad	4830	1247
fear	4097	1024
angry	3995	958
surprise	3171	831
disgust	436	111

V. Experimental settings

A. Environment

The experiment was conducted in a computing environment provided by Google Colab Free. Google Colab is a cloud-based platform that offers a Jupyter Notebook interface along with access to powerful computational resources. The experiment specifically utilized a T4 GPU and 12GB of system RAM.

The T4 GPU, which stands for Tesla T4, is a high-performance graphics processing unit designed for deep learning and machine learning tasks. It is known for its exceptional computational capabilities and efficient memory utilization, making it well-suited for training and inference tasks in computer vision applications.

In addition to GPU, the experimental environment provided 12GB of system RAM. This amount of memory allowed for the loading and processing of the dataset, as well as the training and evaluation of machine learning models used in the project. The available RAM capacity ensured that the experiments could be conducted efficiently without significant memory limitations.

B. Metrics

The accuracy metric was employed to assess the performance and effectiveness of the machine learning models in the human emotion recognition project. Accuracy is a commonly used metric in classification tasks and represents the proportion of correctly classified instances out of the total number of instances in the dataset.

In the context of this project, accuracy was computed by comparing the predicted emotional labels generated by the models with the ground truth labels of the dataset. A model with higher accuracy indicates a higher rate of correctly predicting the emotional states of the input images.

The use of accuracy as the evaluation metric offered a straightforward measure of the overall model performance. It provided an indication of how well the models could generalize to unseen data and accurately classify the emotional states depicted in the front face photos.

However, it is important to note that accuracy alone could not provide a comprehensive evaluation, especially when dealing with imbalanced datasets or when the cost of misclassification varies across different emotional labels. In such cases, additional metrics like precision, recall, F1-score, or ROC AUC can provide more nuanced insights into the model's performance.

Nevertheless, in this project, accuracy is chosen as the primary evaluation metric due to its simplicity and interpretability. It permits a straightforward comparison of the performance of different machine learning algorithms and facilitates the identification of the most accurate model for human emotion recognition.

VI. Experimental Results

TABLE II: Evaluation of classification algorithms(%)

Algorithms	**Accuracy on test set**
Support Vector Machines (RBF)	50.8
Random Forest	39.997
K-Nearest Neighbors (k=22)	40.763
Softmax Regression	37.81

VII. DEMO

A. Steps

The following process involves several steps to classify emotions in photos containing faces:

1) **A photo containing faces**: The process starts with an input photo that contains one or more faces. face. The photo must meet the requirements:
    a) A front face photo with full eyes, mouth, nose and forehead
    b) The face can be captured at various angles
    c) Capture device and face are in a straight line parallel to the ground
    d) The photo must be grayscaled
2) **RetinaFace**: The RetinaFace algorithm is used for face detection. It analyzes the input photo and identifies the locations and boundaries of the faces present in the image. The algorithm provides information about the facial area, including the coordinates of the bounding box around each detected face.
3) **Face photos**: Based on the face detection results from RetinaFace, the input photo is cropped to obtain individual facial photos. These cropped photos isolate each face, making it easier to analyze and classify emotions specifically for each face.
4) **HOG (Histogram of Oriented Gradients)**: The HOG algorithm is applied to each cropped facial photo. HOG computes the gradients of pixel intensities in local image regions and generates histograms of oriented gradients as features. These features capture shape and texture information in the image.
5) **SVM (Support Vector Machine) with RBF kernel**: The model is trained on a labeled dataset to learn patterns and relationships between the extracted HOG features and corresponding emotion labels. The RBF kernel allows the SVM to handle non-linear relationships between features and labels.
6) **Emotion label**: Once the HOG features are extracted from a cropped facial photo, they are passed to the trained SVM model for prediction. The SVM model classifies the features into one of several emotion labels. The predicted label represents the emotion inferred from the facial expression in the cropped photo.

B. Demonstration

Google Colab: The demonstration's result could be found in FinalProject - colab.ipynb file in the same folder with this report.

REFERENCES

[1] FER2013 dataset
https://www.kaggle.com/datasets/msambare/fer2013
[2] RetinaFace library
https://github.com/serengil/retinaface
[3] K-Nearest Neighbors
https://scikit-learn.org/stable/modules/neighbors.html
[4] Understanding Random Forest
https://towardsdatascience.com/understanding-random-forest-58381e0602d2
[5] Deep dive into Softmax Regression
https://towardsdatascience.com/deep-dive-into-softmax-regression-62deea103cb8
[6] Regularization in Softmax Regression
https://medium.com/@rithpansanga/logistic-regression-and-regularization-avoiding-overfitting-and-improving-generalization-e9afdcddd09d
[7] Support Vector Machines
https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
[8] HOG (Histogram of Oriented Gradients)
https://learnopencv.com/histogram-of-oriented-gradients/
[9] HOG (Histogram of Oriented Gradients)
https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f