University Cadi Ayyad
Faculty Polydisciplinaire
Of Safi

**End of the year Project**

To obtain the Fundamental Licence diploma in

**Mathematical and Computer Sciences**

# Automatic emotion detection through facial expressions

***Directed by :***
Mirrou Houssam
Zriouille Aymen

***Supervised by :***
Mrs. Alioua Nawal
Ms. Aaniba Imane

**Academic Year**
2024/2025

# Special Thanks

*At the end of our training cycle, we would first like to thank God, who gave us the courage, will, and strength to accomplish our work.*

*It is with pleasure that we dedicate these few lines as a sign of gratitude and deep recognition to all those who, near or far, have contributed to the completion of this work.*

*First and foremost, we cannot conclude this project without expressing our heartfelt gratitude to our supervisors,* ***Mrs. ALIOUA NAWAL*** *and* ***Ms. Imane Aaniba****, who accompanied us throughout the project. Their expertise, support, and rigorous supervision were essential in enabling us to successfully complete our work.*

*We extend our thanks to our teachers for their contribution to our undergraduate training, for their generosity, and for the great patience with which they supported us, despite their academic and professional responsibilities.*

*We are also deeply grateful to our families and friends for their unwavering support and understanding during this intensive period of work.*

*Finally, we would like to thank all the members of the jury who kindly agreed to evaluate our work, and we hope that our project will be beneficial to the community.*

# Summary

Our project focuses on facial emotion recognition using machine learning and deep learning algorithms. The goal is to implement and compare multiple models for classifying human emotions based on facial expressions. We developed a system that uses Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Convolutional Neural Networks (CNN) to analyze images and identify the corresponding emotion.

Our work concentrated on training, evaluating, and integrating these models into a unified interface. Each model was trained using facial expression datasets, and their performance was assessed based on accuracy, inference speed, and robustness to noise. For traditional models (SVM, KNN), we used Histogram of Oriented Gradients (HOG) features to extract relevant information from facial regions and Principal Component Analysis (PCA) to reduce the high dimensionality of face image data. For the CNN model, we built and trained a deep neural network capable of learning directly from pixel values.

The system supports three types of input: image upload, real-time emotion detection using a webcam, and camera capture through the web interface. The main purpose of the interface is to demonstrate and compare the models' performance in a user-friendly way, but the core focus of the project remains the implementation and evaluation of the underlying algorithms.

Our project aims to demonstrate how different machine learning approaches can be applied to facial expression analysis and highlights the strengths and limitations of each model in real-world scenarios.

# Contents

# List of Tables



# List of Figures

# General Introduction

Human emotions can be important in many areas in real life, such as healthcare, security, customer service , and human-robot interaction. For example, in hospitals, emotion recognition can help doctors better understand how patients feel. In a customer service, it will help them respond in a better way to the clients. In robotics it will also help machines to interact with humans in a normal and natural way.

That's why facial emotion recognition has become an important field in artificial intelligence. The goal is to automatically detect and classify emotions to their specific label (happy,sad,angry,surprise...) just by analyzing a person's facial expression.

In our final project, we created a web based system that can recognize facial emotions using AI. The user can upload an image, take a picture or use the webcam for real-time emotion detection. The system then uses trained machine learning models to analyze the face and then identify the emotion.

We focused on building and comparing three different algorithms: SVM (Support Vector Machine), KNN(K-Nearest Neighbors), CNN (Convolutional Neural Network). These models were trained on Ck+ and Fer2013 datasets, and we studied their performance based on each algorithm and dataset in terms of accuracy, speed, and robustness.

This report is organized into several parts. First, we explain the purpose of the project and why emotion recognition is useful. Then we describe the design and development of our solution, including the algorithms we used and how we trained them. We also present the tools and technologies involved and finally we show the result and conclusion.

Our project shows how AI can help machines better understand human emotions, which can be useful in many real life situations.

# 1 Project Presentation

## 1.1 Introduction

In this chapter we begin by presenting the overall context of the project and giving an overview of the whole topic. This includes explaining the problem we want to solve, reviewing similar projects that have already been done, and starting the main objective of our work. We will also describe the solution we plan to implement, along with the functional and non-functional requirement related to the project.

## 1.2 Context

As we mentioned earlier healthcare, security, customer service, and human robotics can can benefit from the understanding human emotions which plays a very important role. With the growth of artificial intelligence in many fields, we chose to develop a project based on machine and deep learning to provide an effective solution for recognizing human emotions through facial expressions. Our goal is to contribute meaningfully by creating models that cover the needs for these departments. To do this, we studied the challenges involved in emotion recognition and explored ideas to meet these specific needs

## 1.3 Problem Statement

We people rely on various clues to understand emotions and respond appropriately. For example, we use visual information such as facial expressions to identify feelings plus other clues like the tone of the voice or body language. Recognizing emotions accurately can differ from person to person due to the differences in expressions and the change of lighting or camera angles.

Humans are generally good at interpreting emotions; however, automated systems face multiple challenges in detecting and classifying facial emotions reliably.Some emotions can be confused with others (e.g., Fear , surprise) and pictures quality can also affect the detection performance. Moreover, the complexity of facial features requires advanced techniques to extract and analyze relevant informations.

In many fields like healthcare, security, customer service, and human-robot interaction, automatic emotion recognition is important to improve interacting between the system and the client for more natural and adaptive responses. It is therefore important to develop and compare different machine learning models to find solutions that can accurately interpret facial emotions in diverse conditions.

This project focuses on addressing these challenges by implementing and evaluating three different models K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Convolutional Neural Networks (CNN). Each model is trained separately on two common facial emotion datasets, CK+ and FER2013, to assess their strengths and limitations in emotion detection tasks

## 1.4 Study on the existing System

Numerous solutions are already developed for detecting facial emotion expression, designed for various domains like healthcare, security. Here are some well-known systems that work on it:

1. Affectiva

    a. Fonctionnality :

    Affectiva is an emotion recognition technology developed by a company originating from the MIT Media lab. It analyzes facial expressions from videos or images to detect emotions such as joy, anger, surprise, or sadness. It is used in marketing, market research, smart vehicles, and mental health applications.

    b. Advantages :

    - Real-time emotion recognition from simple webcams.
    - Large training database containing millions of faces from diverse cultures
    - Easy integration into other applications via API.
    - Well-established commercial use.

    c. Disadvantages :

    - Proprietary technology, not open-source.
    - Sometimes less accurate in low-light conditions.
    - May require an internet connection depending on use case.

2. Microsoft Azure Face API :

    a. Fonctionnality :

    Microsoft's Azure Face service enables face detection and analysis in images, including emotion recognition. It detects emotions such as happiness, sadness, anger, fear, contempt, among others.

    b. Advantages :

    - Cloud API accessible remotely.
    - Fast recognition with comprehensive documentation.

- High accuracy under normal conditions.
- Integration with other Azure services.

c. Disadvantages :

- Requires a stable internet connection.
- Paid usage after exceeding a certain quota.
- Dependent on cloud infrastructure.

3. OpenFace :

a. Fonctionnality :

OpenFace is an open-source tool for facial recognition and emotion analysis based on machine learning models. It can be used locally for real-time or offline video analysis. It extracts features such as facial movements, eye position, and gaze points to estimate emotions.

b. Advantages :

- Free and open-source.
- Does not require an internet connection.
- Customizable and suitable for research.
- Well-suited for academic environments.

c. Disadvantages :

- Less user-friendly for beginners.
- Less accurate than commercial solutions if not properly configured.
- Requires technical knowledge for effective integration.

### 1.4.1 Review

| Application | Functionality | Advantages | Disadvantages |
|---|---|---|---|
| **Affectiva** Developer Toolkit | Analyzes facial expressions in real-time to detect emotions such as joy, anger, and surprise. Primarily used in marketing and driver monitoring. | • Simple to use in real time.<br>• Large database containing diverse faces.<br>• Easy integration in applications.<br>• Well-established commercial use. | • Not open-source.<br>• May misclassify low-light pictures.<br>• Might require internet connection. |
| **Microsoft Azure Face API** Cloud-based service | Offers facial recognition including emotion detection, age, and gender estimation. Accessible via cloud services. | • Easy integration via REST API.<br>• Includes face detection, verification.<br>• Scalable with good documentation. | • Requires internet connection.<br>• Usage costs increase with scale.<br>• Limited customization of models. |
| **OpenFace** Desktop application | Open-source toolkit for facial behavior analysis, emotion recognition, and facial landmark detection. Suitable for research and development. | • Free and open-source.<br>• Works offline.<br>• Research-friendly and customizable. | • Requires technical setup.<br>• Lower accuracy than commercial tools.<br>• Limited real-time performance on weak hardware. |

Table 1: Study of the Existing Systems

## 1.5 Objective and solution

The objective of this project is to develop a facial recognition system that accurately classifies and outputs the label of the predicted emotion from facial images using datasets like FER2013 and CK+. To built this project we made three models (SVM, KNN, CNN) and compared them based on accuracy, inference speed, and robustness to noise to find the best model for practical emotion labeling in real-world scenarios.

## 1.6 Requirements Specification

### 1.6.1 Functional Requirements

These are the main functions of the system. They specify the expected input-output behavior. The system must allow detection of facial emotions and should:

1. Detect faces from images accurately.
2. Classify each detected face into one of the predefined emotion categories (e.g., angry, happy, sad).
3. Provide the predicted emotion label as output for each detected face.
4. Operate efficiently to deliver real-time or near real-time emotion recognition results.

### 1.6.2 Non-Functional Requirements

These requirements makes our model respect some rules to achieve optimal response time for example. Our model should:

1. Be robust and handle varying lighting conditions and face orientations.
2. Ensure fast processing times to allow near real-time emotion detection.
3. Maintain accuracy and reliability across different users and environments.
4. Be user-friendly and easy to integrate into other applications or workflows.

## 1.7 Conclusion

In conclusion, our project tries to provide an efficient and accurate facial emotion recognition tool Using algorithms such as KNN with HOG features offers us accurate labeling of emotions.The models are designed to be robust, scalable, and user-friendly, aiming to address limitations in existing solutions.

# 2 Artificial intelligence (AI)

## 2.1 Definition

Artificial intelligence (AI) is a branch of computer science that focuses on creating machines capable of performing tasks that normally require human intelligence, such as speech recognition, understanding natural language, learning, and problem-solving. AI technologies use complex algorithms to analyze data, identify patterns, and learn to perform specific tasks autonomously. AI is widely used in many fields, such as medicine, manufacturing, finance, transportation, energy, agriculture. There are several subfields of artificial intelligence (AI), each focusing on a specific aspect of creating intelligent machines. here are some of the main subfields of AI :

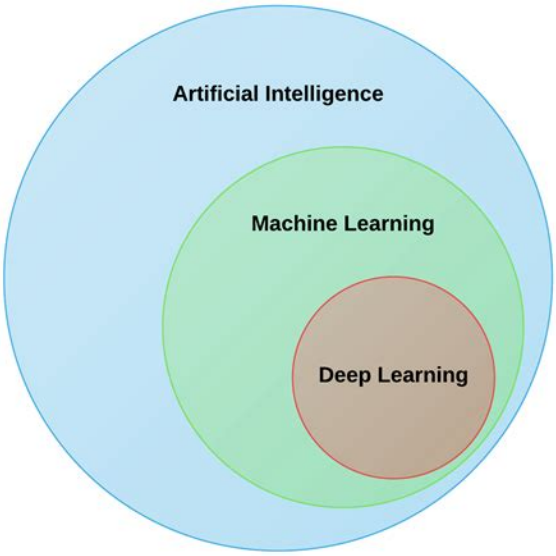



Figure 1: Subfields of AI

The figure represents a general diagram of artificial intelligence and its subfields, including machine learning and deep learning.

## 2.2 Machine Learning

Machine learning is an AI technique that allows machines to learn from data without being explicitly programmed. Machine learning can be divided into two broad categories: supervised learning and unsupervised learning.

**Supervised learning** is a learning method where the algorithm is trained on a labeled data set. This means that each data example in the training set is associated with a known label or answer, and the algorithm learns to predict that label based on the characteristics of the example. The goal of supervised learning is to generalize the knowledge acquired during training to correctly predict labels for previously unseen data examples.

**Unsupervised learning** is a learning method where the algorithm is trained on a set of unlabeled data. It involves analyzing data without a predefined answer to find patterns or structures. The goal of unsupervised learning is to discover interesting structures in the data, such as groupings, patterns, and relationships. There are several unsupervised learning methods, including:

- **Clustering:** This method involves grouping similar data together based on their proximity to each other.
- **Dimensionality Reduction:** This method involves reducing the dimensionality of the data by eliminating variables that do not contribute significantly to the variability of the data.
- **Association Rules:** This method involves discovering relationships between variables in the data.

### 2.2.1 KNN (K-Nearest Neighbors)

#### 2.2.1.1 Definition

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm that is mainly used for classification. It relies on the 'k' closest data points, considered as neighbors, to classify a new input and makes a prediction based on the most frequent class among its neighbors. Since KNN makes no assumptions, we can say that it is a non-parametric and instance-based learning method.[KNN]

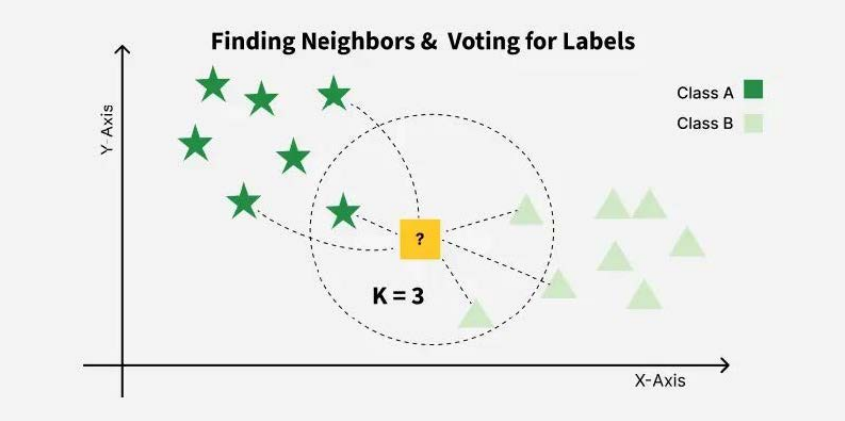



Figure 2: Classifying new data based on k parameter (KNN)

#### 2.2.1.2 Statistical Methods For Selecting k

- **Cross-Validation:** One of the most effective ways of finding the ideal value of k is by using k-fold cross-validation. This requires dividing the dataset into k subsets. Some subsets are used to train the model, and others are used to test it. This is repeated for all subsets. The k value that gives the best average accuracy for the tests is usually the best one to utilize. [KNN]
- **Elbow Method:** In the Elbow Method, we graph a curve showing the error rate or accuracy against different values of k. As error normally decreases with an increase in k. But after some point error does not decrease as much. The point where the direction of the curve turns and the curve looks like an ”elbow” is generally the best value for k. [KNN]
- **Odd Values for k:** It is a good idea to keep k as odd, particularly in classification problems. This avoids ties when one has to decide which class is most dominant among the neighbors. [KNN]

#### 2.2.1.3 Distance Metrics Used in KNN Algorithm

KNN employs distance metrics in finding nearest neighbor, the neighbors are utilized for classification and regression task. In order to find nearest neighbor we use following distance metrics:

- **Euclidean Distance :** :Euclidean distance is that distance which is the straight-line distance between two points in a plane or space. You can think of it like the shortest way you would walk if you walk directly from one point to another.[KNN]

$$\text{distance}(x,X) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Manhattan Distance :** This is the distance you would travel if you could only move horizontally and vertically along streets or a grid. It is also called ”taxicab distance” because a taxi can only move along the grid-like city streets. [KNN]

$$\text{distance}(x,X) = |x_1 - x_2| + |y_1 - y_2|$$

- **Minkowski Distance :** Minkowski distance is a group of distances, having both Euclidean and Manhattan distances as special cases.[KNN]

$$D(x,y) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- **Cosine distance :** measures the angle between two vectors, focusing on their direction rather than magnitude.[KNN]

$$\text{Cosine Distance} = 1 - \frac{A \cdot B}{\|A\| \, \|B\|}$$

#### 2.2.1.4 Advantages and Disadvantages

**Advantages:**

- **Simple to use :** Simple to comprehend and apply.
- **No training step :** Since it only saves the data and uses it for prediction, there is no requirement for training.
- **Few parameters :** Just the distance method and the number of neighbors (k) need to be set.
- **Versatile :** works for issues involving both regression and classification.

**Disadvantages:**

- **Slow with large data :** Every point must be compared when making a prediction.
- **No training step :** When data includes too many features, accuracy decreases.
- **Few parameters :** It may overfit, particularly if the data is dirty or high-dimensional.

#### 2.2.1.5 Conclusion (KNN)

For our model (on ck+ dataset) we chose k as an odd number we followed the rule of sqrt(number of testing images) and we went with the weight distance which means favoring the points that are closer to it and also we went with the Euclidean distance in the metrics to calculate the closest point. And in case of fer dataset (a large dataset) we chose also k as an odd number and distance for weight but we chose cosine for the metrics because it gave us more accuracy than the other metrics in case of a large dataset.

### 2.2.2 SVM (Support Vector Machine)

#### 2.2.2.1 Definition

Support Vector Machine (SVM) is a supervised machine learning algorithm primarily used for classification tasks but can also be used for regression. The main idea behind SVM is to find a hyperplane that best separates the data into different classes.svm creates a decision boundary that maximizes the margin between classes. The data points closest to this hyperplane are called support vectors, which are pivotal in defining the decision boundary. [SVM]

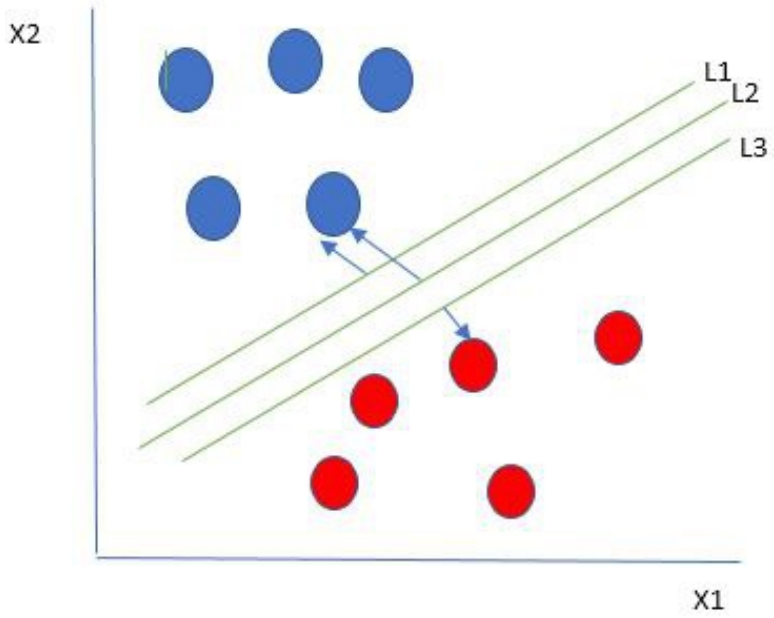



Figure 3: SVM classification showing decision boundaries separating two classes

#### 2.2.2.2 Statistical Methods for Selecting the Best Model in svm

- **Cross-Validation:** Similar to KNN, cross-validation can be used to determine the best parameters for the SVM model. This helps in selecting the hyperparameters like the regularization parameter (C) and kernel type (linear, polynomial, or radial basis function). [SVM]
- **Grid Search:** A grid search method is often used in SVM to find the optimal values for parameters such as C and kernel type. The search space is defined by different values of these parameters, and the best set of parameters is selected based on cross-validation results. [SVM]
- **Kernel Trick:** The SVM algorithm can use the kernel trick to handle non-linearly separable data by transforming the data into a higher-dimensional space where a linear decision boundary is possible. The most commonly used kernels are: [SVM]
  - → **Linear kernel:** Used when the data is linearly separable.
  - → **Polynomial kernel:**

#### 2.2.2.3 Types of Kernels Used in SVM

SVM uses different types of kernels to transform the data, allowing for better separation of classes in higher dimensions: [SVM]

- **Linear Kernel:** The simplest kernel, which is used when data is linearly separable. The decision boundary is a straight line (or hyperplane in higher dimensions). [SVM]

$$K(\mathbf{x},\mathbf{y}) = \mathbf{x}\cdot\mathbf{y}$$

- **Polynomial Kernel:** This kernel maps the data into higher-dimensional spaces using polynomial functions. It is suitable when there is a non-linear relationship between data points. [SVM]

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+c)^d$$

Where c is a constant, and d is the degree of the polynomial.

- **RBF Kernel (Radial Basis Function):** Popular for non-linear classification tasks, the RBF kernel uses a Gaussian function to transform data into a higher-dimensional space. [SVM]

$$K(x,y) = \exp\left(\frac{-\|x-y\|^2}{2\gamma^2}\right)$$

Where $\gamma$ is a parameter that controls the width of the Gaussian.

- **Sigmoid Kernel:** Similar to a neural network, this kernel uses a sigmoid function to map data points. It is less commonly used compared to other kernels [SVM].

$$K(x,y) = \tanh(\alpha x \cdot y + c)$$

#### 2.2.2.4 Advantages and Disadvantages (SVM)

**Advantages :**

- **Slow with large data:** Needs to compare every point during prediction.
- **Versatile:** SVM can be used for both classification and regression tasks.
- **Robust to Overfitting:** When used with the right kernel, SVM can generalize well to unseen data and is less prone to overfitting, especially in high-dimensional spaces.
- **Works Well with Non-linear Data:** With non-linear kernels like RBF, SVM can handle complex relationships between data points.

**Disadvantages :**

- **Computationally Intensive:** SVM can be slow to train, especially with large datasets, as it involves finding the optimal hyperplane in a high-dimensional space.
- **Sensitive to Choice of Kernel:** The performance of SVM heavily depends on the choice of kernel and its parameters. Selecting the wrong kernel can lead to poor performance.
- **Requires Proper Tuning:** SVM requires careful parameter tuning (e.g., choosing the right kernel, regularization parameter, and margin) to achieve optimal performance.
- **Memory Usage:** SVM can be memory-intensive, particularly when dealing with large datasets, as it stores the entire dataset in memory during training.

#### 2.2.2.5 Conclusion (SVM)

For our model (on the CK+ dataset), we chose a linear kernel, as the data was linearly separable. We utilized the standard Euclidean distance to calculate the distance between data points. In this case, SVM was efficient at finding the optimal hyperplane to separate the classes.

For the FER dataset (a larger dataset), we chose a radial basis function (RBF) kernel due to the non-linearity of the data. We used the Gaussian function to map the data to a higherdimensional space. This kernel provided better performance in terms of classification accuracy compared to the linear kernel, especially for complex and non-linearly separable data.

## 2.3 Deep Learning

### 2.3.1 Definition

Convolutional neural networks (CNNs) are used in the machine learning subfield of deep learning to extract knowledge from unprocessed data.

### 2.3.2 Convolutional Neural Network (CNN)

In general, an artificial neural network is a mathematical model inspired by the functioning of the human brain. It is composed of a collection of interconnected nodes, or artificial neurons, that work together to learn patterns in data.

A Convolutional Neural Network (CNN) is a specific type of deep neural network that consists of multiple layers of interconnected nodes. The term ”deep” refers to the fact that these networks have many layers, which allows them to learn complex patterns in data. Figure 4 provides an overview of a deep neural network (CNN):

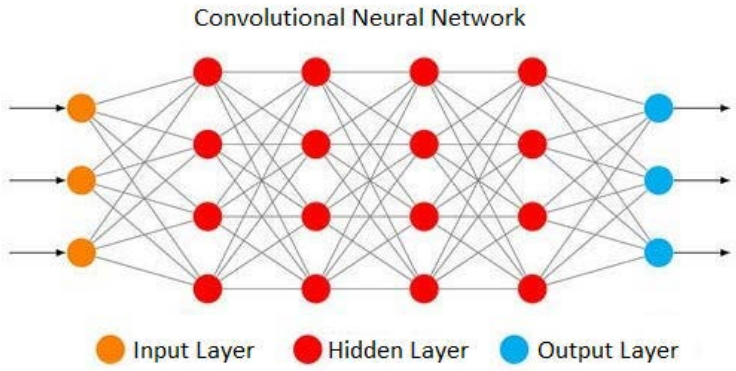



Figure 4: Example of a deep neural network (CNN)

### 2.3.3 How a Convolutional Neural Network works

To understand how a deep neural network works, let’s take the example of processing the image of an x shown in Figure 5.

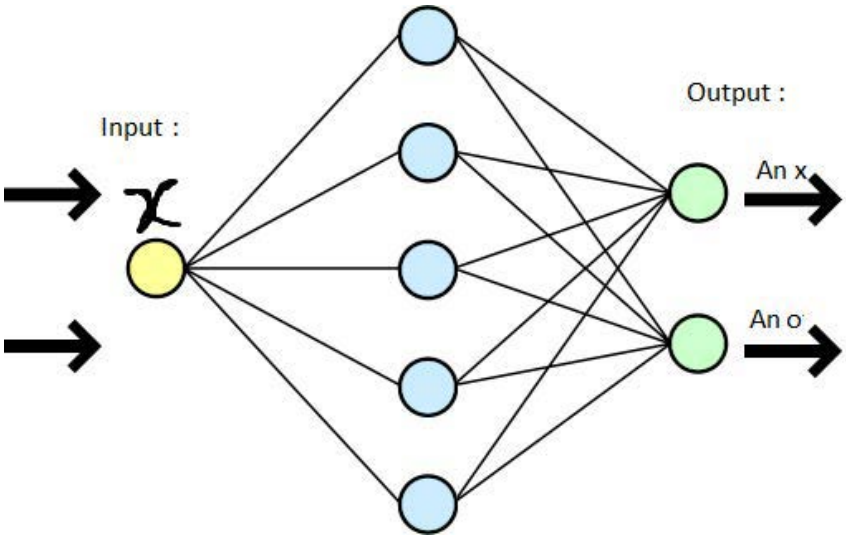



Figure 5: x image processing using a Convolutional Neural Network

What a computer see :

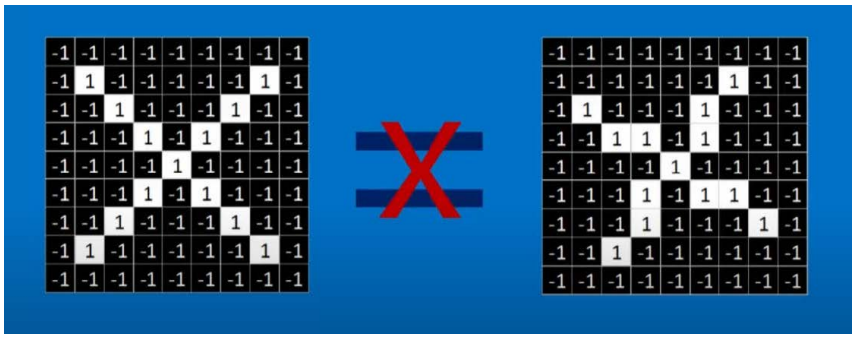


Figure 6: How a computer see the x

So as we can see in the picture above the x that is written by a machine and the one that it's ideal ,the one written by a human doesn't look the same, CNN relies on layers to detect features and determine whether the image corresponds to an 'x' or not. Here are the layers through which it processes the data:

- **Convolutional layer:**

  Before having a convolutional layer, we should go through a filtering process that breaks down the image into smaller parts like 3 by 3 pixels. Then we align the feature and image patch, multiply each image pixel by the corresponding feature pixel, add them up, and finally divide them by the total number of values (in this case, 3×3 = 9).

  **Conclusion:** This layer helps us see some patterns that resemble an "x".

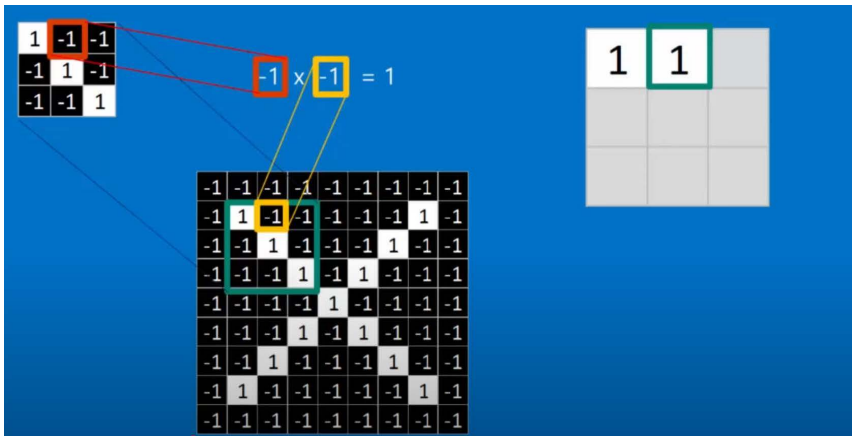



Figure 7: Filtering the x image

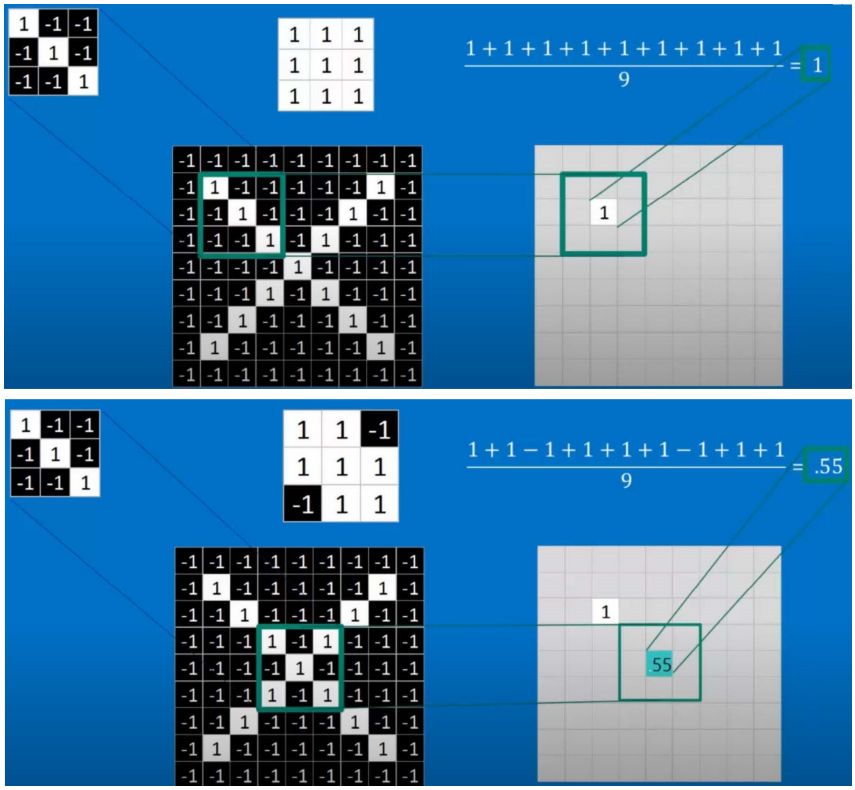



Figure 8: Addition and division (two examples)

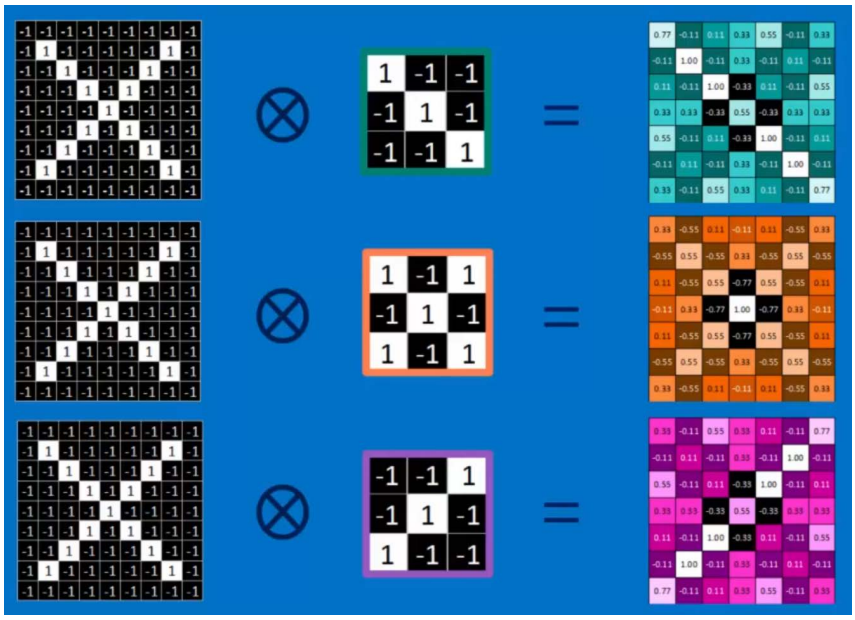


Figure 9: Trying every possible match (Convolution Layer)

In these figures, we see the ”x” shape divided into three smaller patches. Each patch contributes to a new image channel, effectively creating a stack of filtered images.

- **Pooling Layer:**

  After choosing a window size (often 2 or 3) and a stride (typically 2), we travel the window across our filtered photos before selecting the greatest possible value.

  This layer aids in image reduction, allowing us to preserve crucial information while reducing image size.

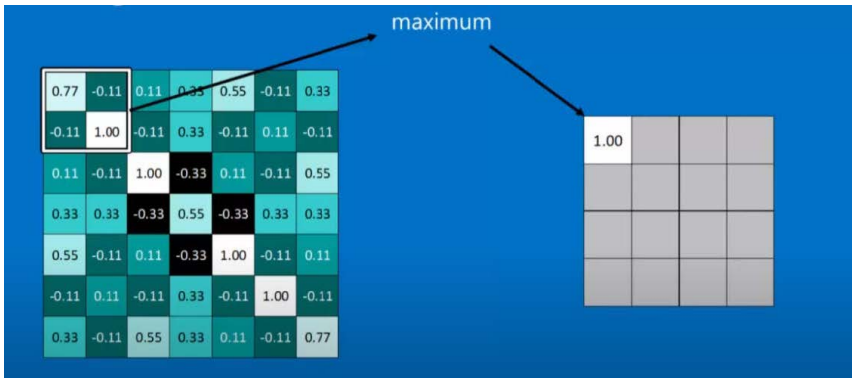



Figure 10: Keeping the important information (Pooling layer)

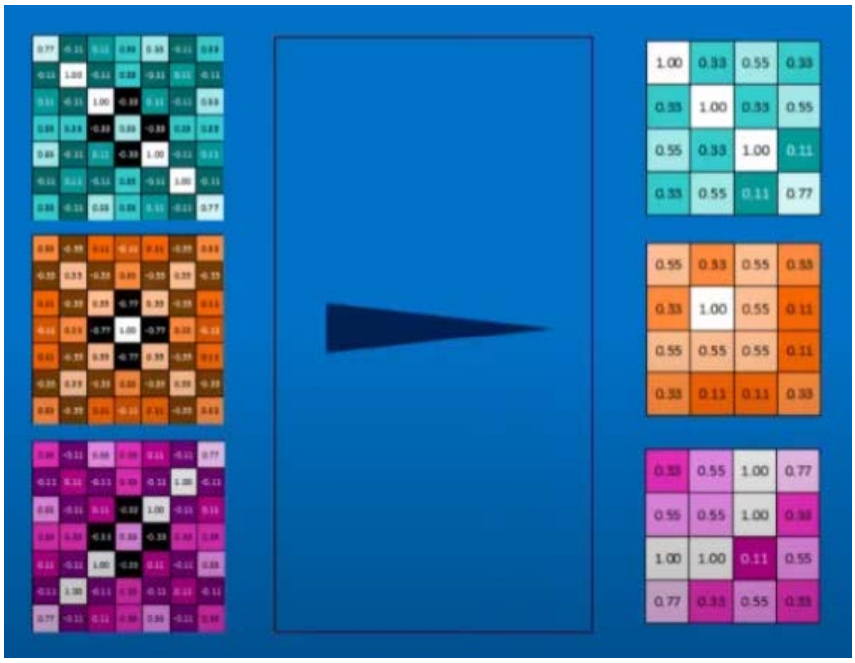


Figure 11: Stacks of images (from convolutional to pooling)

- **Normalization layer (Relu, comes before Pooling layer):**

  So what this layer does is take the negative values and sets them to 0.

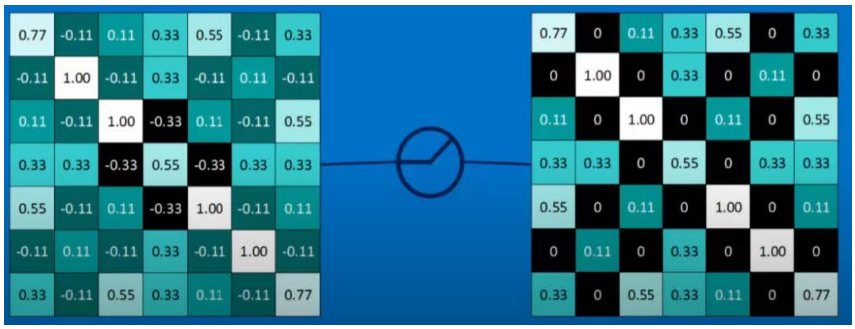



Figure 12: Eliminate the negative values (ReLu)

We can repeat layers several (or many) times so we get result like this :

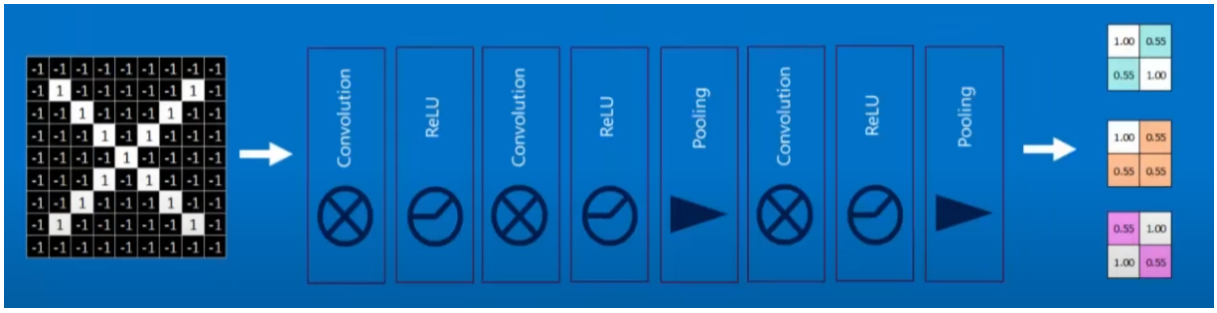



Figure 13: After going through layers many time

- **Fully connected layer:**

  We take all the important information (features) from the last layer, flatten them meaning put them in a single vector, and then use that to make our final prediction.

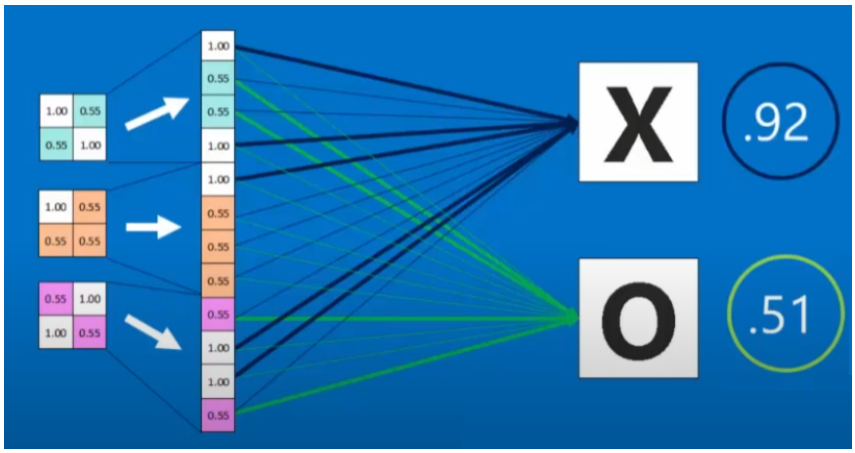



Figure 14: Predictions of fully connected layers

We can also repeat this layer multiple times so we can get a better predictions, and finally underneath we will show how the full layers connect to each others creating a CNN model.

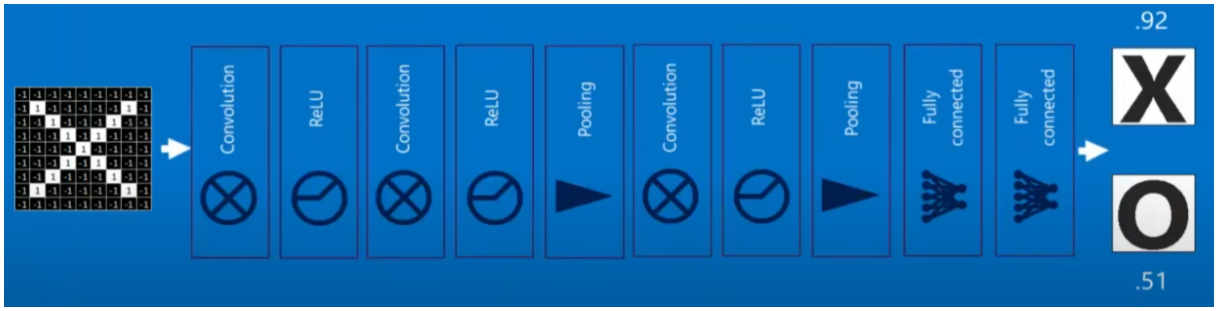



Figure 15: Full Convolutional Neural Network figure

# 3 Emotion Detection

## 3.1 Introduction

In this chapter, We will discus why emotion detection is important across various fields.We will also present the datasets used for training, and why we chose these algorithms to make these models.

## 3.2 Objective

In many places like we mentioned earlier can be highly valuable in various domains, for example the costumers department we can know how the clients react to interactions. However, some existing systems may rely on significant memory and CPU usage, incur costs, and some may not be open-source.

So we can try to improve the weak point with simple models as you will see next.

## 3.3 Datasets

### 3.3.1 Ck+ Dataset

The Extended Cohn-Kanade (CK+) dataset is one of the most widely used datasets for facial expression and emotion recognition research. It provides a reliable and well annotated collection of facial images that capture a range of basic human emotions. [Ck+]

#### 3.3.1.1 Overview

This dataset contains adapted data consisting of 920 images from the original CK+ dataset. The data is reshaped to 48x48 pixels in grayscale format, and faces are cropped using faces are cropped using haarcascade_frontalface_default .

Noisy (based on room light/hair format/skin color) images were adapted to be clearly identified using Haar classifier. Columns from file are defined as `emotion`, `pixels`, and `Usage`. [Ck+]

#### 3.3.1.2 Emotion Labels

Out of the 593 sequences, 327 sequences have been labeled with one of seven emotion categories:

- 0 : Anger (45 samples)
- 1 : Disgust (59 samples)
- 2 : Fear (25 samples)
- 3 : Happiness (69 samples)

- 4 : Sadness (28 samples)
- 5 : Surprise (83 samples)
- 6 : Neutral (593 samples)
- 7 : Contempt (18 samples)

#### 3.3.1.3 Advantages

- **Well-labeled:** The dataset is carefully annotated with emotions and Facial Action Units (FAUs).
- **Temporal information:** It includes full sequences, not just static images, allowing dynamic analysis.

### 3.3.2 Fer2013 Dataset

The FER2013 (Facial Expression Recognition 2013) dataset is a widely used dataset for facial emotion recognition. It is popular for training deep learning models due to its size, diversity, and availability in a simple CSV format.[Fer]

#### 3.3.2.1 Overview

The data has 48x48 pixel grayscale images of faces. The faces are already cropped and centered in the image and they are all about the same.

The training set contains 28,709 images and the test set contains 3,589 images.

- 0 : Angry (958 samples)
- 1 : Disgust (59 samples)
- 2 : Fear (1024 samples)
- 3 : Happy (1774 samples)
- 4 : Sad (1233 samples)
- 5 : Surprise (1247 samples)
- 6 : Neutral (593 samples)

#### 3.3.2.2 Advantages

- **Large size:** Much larger than CK+, which helps reduce overfitting in deep learning models.[Fer]
- **Public availability:** Freely available on Kaggle and widely supported in the research community.[Fer]
- **Varied data:** Includes diverse subjects with varying facial angles and lighting conditions.[Fer]

### 3.3.3 The use of CK+ and Fer2013 in our project and limitations

For CK+ it has some limitation like lacking some images in the fear folder for example is having just 25 samples which will pose some problems during training the models but it's efficient for both simple and complex models.

For fer2013 it also has the same problem as ck+ data set but only affect the simple models (KNN,SVM) not the cnn model which will help us to achieve our goal to train a lightweight models.

## 3.4 The choice of the algorithm

In this project, we selected two different types of machine learning algorithm KNN and SVM and a deep learning algorithm CNN each one of them has unique strengths between simplicity, performance, and computational cost.

### 3.4.1 K-Nearest Neighbors (KNN)

We chose KNN because it's simple and easy to implement, requires minimal training, and stores all the training data, making predictions based on the closest neighbors. It works well when the number of samples is not very large, which makes it ideal for small datasets (KNN).

### 3.4.2 Support Vector Machine (SVM)

SVMs are effective in high-dimensional spaces, making them strong performers in classification tasks. It's also useful for CK+, where the dataset is relatively small. SVMs are robust

for small to medium-sized datasets and are effective with both linear and non-linear boundaries using kernel functions.

### 3.4.3 Convolutional Neural Networks (CNN)

It's widely known that CNNs are effective in image classification tasks, including facial recognition and emotion detection. They learn directly from image pixels, reducing the need for manual preprocessing.

### 3.4.4 Conclusion

| Algorithm | Simplicity | Feature Extraction | Best for | Weakness |
|---|---|---|---|---|
| **KNN** | Very High | Manual | Small datasets, easy setup | Slow Prediction, poor with noise |
| **SVM** | Moderate | Manual | Medium datasets, strong classifiers | Not scalable to large data |
| **CNN** | Low | Automatic | Large datasets, high accuracy | Needs more training data and compute |

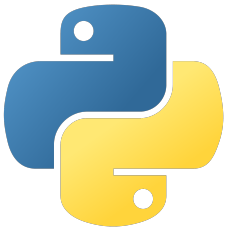
Figure 16: Conclusion of algorithm choices

# 4 Development tools and technologies

## 4.1 Introduction

In this chapter, we will discuss which programming languages we used for our project, the libraries we used, and why they were chosen. We will also highlight the integrated development environment (IDE) in which we tested and wrote our code.

## 4.2 Python



Python is a multi-paradigm, interpreted, and cross-platform programming language. It can be used for structured, functional, and object-oriented imperative programming. Python has dynamic typing with

strong type checking, automatic memory management, and an exception handling system; it is similar to Perl, Ruby, Scheme, Smalltalk, and Tcl. The Python programming language has a free and open-source BSD5-like license and runs on most platforms for computing from smart phones to mainframes6, from Windows to Unix, such as GNU Linux, macOS, Android, iOS, and can be translated to Java or .NET. It is built to optimize increases programmer productivity with high-level facilities and straightforward syntax. It is also appreciated by some teachers who find it a language where the syntax, clearly separated from low-level mechanisms, enables simple introduction to fundamental programming principles. [Pyt]

## 4.3 HTML


Hypertext Markup Language (HTML) is the governing markup language of web documents to be rendered on a web browser. HTML indicates the structure and content of web pages. It is augmented by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript, a scripting language. Web browsers retrieve an HTML document from a web server or from local storage and display the documents in the form of multimedia web pages.HTML specifies the structure of a web page semantically and originally contained suggestions for presentation.HTML elements are the building blocks of HTML documents. With the help of HTML elements, pictures and other objects such as interactive forms can be integrated into the resulting page. HTML provides a means to represent structured documents by marking up structural semantics for document such as headings, paragraphs, lists, links, quotes, etc. HTML elements are declared by tags, expressed in angle brackets. [HTM]

## 4.4 CSS


Cascading Style Sheets (CSS) is a style sheet language for describing the presentation and rendering of a document written in a markup language such as HTML or XML (and its variants, such as SVG, MathML or XHTML). CSS is one of the building blocks of the World Wide Web, along with HTML and JavaScript. CSS is designed to enable authors and presenters to disentangle presentation and content, such as layout, colors, and fonts.

This separation can improve content readability, because the content can be defined independently without considering how it will be presented; provide more

flexibility and ease of use in specifying presentation attributes which makes things easier and less repetitive in structural content. [CSS]

## 4.5 JavaScript


JavaScript (often abbreviated as JS) is a programming language for computers and a World Wide Web standard technology, along with CSS and HTML. Almost all websites utilize JavaScript on the client-side for behavior of a page. Web browsers use a certain JavaScript engine which executes client code. These engines are also utilized in some servers and applications. The most common runtime system outside of browsers is Node.js.

JavaScript is a high-level, often just-in-time-compiled language that adheres to the ECMAScript standard. It is dynamically typed, uses prototype-based object-oriented programming, and first class functions. It is multi-paradigm and accommodates event-driven, functional, and imperative programming.[Jav]

## 4.6 Visual Studio Code

Visual Studio Code, or VS Code, is a source-code editor developed by Microsoft for Windows, Linux, macOS and web browsers. It features debugging support, syntax highlighting, intelligent code completion, snippets, code refactoring, and integrated version control with Git. The user can change the theme, keyboard shortcuts and settings, and install extensions to add functionality.[Vis]

**Why Python ?**

We chose Python for many reasons.

Python is known for its readability and easier syntax, whats makes it easy to write and understandable, especially useful when building and testing machine learning models. It provides a lot of libraries that are designed for computer vision and deep learning, such as OpenCV, TensorFlow, Keras, PyTorch, Dlib and `face_recognition`. These libraries provides prebuilt functions and tools that reduce development time significantly and makes it easier to implement complex models. Also Python can be used in many operating systems and open source which makes it run in various operating systems. And it's community is pretty large so they make sure strong support for debugging, learning. For these reason we chose that we go with python

because it becomes the get go language for both academic researches and industry applications in facial recognition.

## 4.7 The Python libraries used

Python is a programming language that can be used in many contexts and adapts to any type of application thanks to specialized libraries.

Among these libraries are those specifically for artificial intelligence:

### 4.7.1 OpenCV

OpenCV (Open Computer Vision) is a free library, initially developed by Intel, specializing in real-time image processing. The OpenCV library provides a wide range of features for creating programs from raw data to basic graphical interfaces. [Ope]

It offers most of the classic low-level image processing operations:

- reading, writing, and displaying an image;
- calculation of the gray level histogram or color histograms;
- lissage, filtrage ;
- image thresholding (Otsu method, adaptive thresholding)
- segmentation (related components, GrabCut);
- mathematical morphology.

This library has established itself as a standard in the field of research because it offers a significant number of tools from the state of the art in computer vision such as:

1. Reading, writing, and displaying a video (from a file or a camera)
2. Detection of lines, segments, and circles using the Hough Transform
3. Face detection using the Viola and Jones method
4. Face detection using the Viola and Jones method
5. Motion detection, motion history

6. Object tracking using mean-shift or Camshift
7. Detection of points of interest
8. Optical flow estimation (Lucas-Kanade method)
9. Delaunay triangulation
10. Voronoi diagram
11. Convex hull
12. Fitting an ellipse to a set of points using the least-squares method

Since OpenCV version 2.1, the focus has been on matrices and operations on them. Indeed, the basic structure is the matrix. An image can be thought of as a matrix of pixels. Thus, all basic matrix operations are available, including:

- transpose.
- determinant calculation.
- inversion.
- multiplication (by a matrix or a scalar).
- eigenvalue calculation.

### 4.7.2 Numpy

The NumPy library allows you to perform numerical calculations with Python. It introduces easier management of arrays of numbers. The term Numpy is actually an abbreviation for ”Numerical Python.” It is an open-source library in the Python language. This tool is used for scientific programming in Python. [Num]

Pure Python is not very efficient for calculations. Lists are not efficient objects for representing large numerical arrays. Numpy was created at the initiative of developers who wanted to combine the flexibility of the Python language with powerful algebraic calculation tools.

NumPy is based on:

The ndarray: a multidimensional array; derived objects such as masked arrays and ufuncs matrices : mathematical operations optimized for arrays; methods for performing fast operations on arrays: shape manipulation; sorting; input/output; FFT; linear algebra; statistics; random calculations; and much more!

### 4.7.3 scikit-learn(sklearn)

Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings. Source code, binaries, and documentation can be downloaded from http://scikit-learn.sourceforge.net. Keywords: Python, supervised learning, unsupervised learning, model selection. [sci]

### 4.7.4 matplotlib

matplotlib is a library for making 2D plots of arrays in python. Although it has its origins in emulating theMATLABTM graphics commands, it does not require MATLABTM, and can be used in a pythonic, object oriented way. Although matplotlib is written primarily in pure python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays. [mat]

matplotlib is designed with the philosophy that you should be able to create simple plots with just a few commands, or just one! If you want to see a histogram of your data, you shouldn't need to instantiate objects, call methods, set properties, and so it; it should just work.

### 4.7.5 joblib

Joblib is a library designed to provide lightweight pipelining in Python. It is particularly effective for tasks involving large arrays or datasets, which are common in scientific computing and data analysis. Joblib's caching mechanism helps avoid recomputation, making it highly efficient for iterative algorithms. It also supports parallel processing through multiprocessing, enabling the distribution of computational tasks across multiple CPU cores.[job]

### 4.7.6 TensorFlow

TensorFlow is an open-source computational library developed by the Google Brain Team for building deep learning models and neural networks. It is widely used for data analysis, speech recognition, machine translation, image recognition, time series prediction, and much more. The TensorFlow library is based on data flow graphs, where data is represented as multidimensional

tensors, and operations are represented as nodes in a graph. It allows you to create deep learning models using artificial neural networks. [Ten]

### 4.7.7 sckit-image (skimage)

scikit-image is an image processing library that implements algorithms and utilities for use in research, education and industry applications. It is released under the liberal Modified BSD open source license, provides a well-documented API in the Python programming language, and is developed by an active, international team of collaborators. In this paper we highlight the advantages of open source to achieve the goals of the scikit-image library, and we showcase several real-world image processing applications that use scikit-image. More information can be found on the project homepage, http://scikit-image.org. [sck]

### 4.7.8 dlib

In order to overcome the problems of OpenCV in face detection, such as missing detection, false detection and poor recognition effect, a new method of Dlib face recognition based on ERT algorithm is proposed. This method can realize face recognition and feature calibration by Python, which calls a large number of trained face model interfaces, and it has good robustness for occlusion. By testing the process of face detection, feature point calibration, feature vector extraction and comparison in small deflections and positive faces images and videos, the experimental results show that the proposed method is superior to OpenCV method, it can effectively improve detection sensitivity, recognition precision and recognition effect. It can effectively solve the problem of poor real-time performance in dynamic image recognition. [dli, a]

### 4.7.9 seaborn

Seaborn is a Python package for creating statistical visualizations. It strongly interacts with pandas data structures and offers a high-level interface to matplotlib. The seaborn library's functions provide a declarative, dataset-oriented API that facilitates the conversion of data-related queries into graphics that can provide answers. Seaborn automatically maps data values to visual properties like color, size, or style when given a dataset and a plot specification. It then computes statistical transformations internally and embellishes the plot with a legend and relevant axis labels. Numerous seaborn functions are capable of producing multi-panel graphics that prompt comparisons between conditional subsets of data or between various variable pairs within a dataset. seaborn is designed to be useful throughout the lifecycle of a scientific project. By producing complete graphics from a single function call with minimal arguments, seaborn

facilitates rapid prototyping and exploratory data analysis. And by offering extensive options for customization, along with exposing the underlying matplotlib objects, it can be used to create polished, publication-quality figures.[sea]

#### 4.7.10 keras

Keras is a high-level Python library for deep learning that is small and simple to learn. It may be used with TensorFlow (also known as Theano or CNTK). It lets developers handle the finer points of tensors, their shapes, and their mathematical subtleties while concentrating on the core ideas of deep learning, such building layers for neural networks. Keras's back end must be TensorFlow (or Theano or CNTK). Without having to deal with the somewhat complicated TensorFlow (or Theano or CNTK), you may utilize Keras for deep learning applications. Sequential and functional APIs are the two main categories of frameworks. The most popular application of Keras is the sequential API, which is predicated on the notion of a series of layers. The sequential API is based on the idea of a sequence of layers; this is the most common usage of Keras and the easiest part of Keras. The sequential model can be considered as a linear stack of layers.[ker]

### 4.8 Conclusion

In conclusion this chapter contains all the development tools and technologies in detail that we need to make this project. We identified and presented the various key technologies, such as programming languages, libraries, and platforms, used in the development process.

## 5 Realization

### 5.1 Introduction

In this chapter, we will focus on how we developed our models. We describe the different steps in the process, starting with data collection and preparing it for our models, then explaining how we trained our model and the problems we encountered.

## 5.2 Data collection

### 5.2.1 Ck+

#### 5.2.1.1 KNN and SVM

Fortunately, we found the CK+ data as a CSV file that contains emotion labels from 0 to 7, each corresponding to its respective emotion (e.g., 0: Anger). We wrote a script that converts the CSV file. Here's the code below:

```python
# Define emotion labels and their corresponding folders
emotion_labels = {
    0: "Anger",
    1: "Contempt",
    2: "Disgust",
    3: "Fear",
    4: "Happiness",
    5: "Neutral",
    6: "Sadness",
    7: "Surprise"
}

# Set paths
csv_path = "data/ckextended.csv"  # Change to your CSV file path
output_folder = "data/"

# Create main output directory
os.makedirs(output_folder, exist_ok=True)

# Create subdirectories for each emotion
for emotion in emotion_labels.values():
    os.makedirs(os.path.join(output_folder, emotion), exist_ok=True)

# Load CSV
df = pd.read_csv(csv_path)

# Define columns (Modify based on your CSV structure)
emotion_col = "emotion"  # Emotion label column
pixels_col = "pixels"  # Column containing pixel values

# Prepare list to store image paths
image_paths = []

# Convert each row into an image
for idx, row in tqdm(df.iterrows(), total=len(df)):
    emotion = int(row[emotion_col])  # Get emotion label
    pixels = np.array([int(p) for p in row[pixels_col].split()])  # Convert string to array
    img_size = int(np.sqrt(len(pixels)))  # Assume square images
    image = pixels.reshape((img_size, img_size))  # Reshape to image

    # Determine folder and filename
    emotion_folder = os.path.join(output_folder, emotion_labels[emotion])
    img_filename = f"{emotion}_{idx}.png"
    img_path = os.path.join(emotion_folder, img_filename)

    # Save image
    cv2.imwrite(img_path, image)

    # Store image path
    image_paths.append([img_path, emotion_labels[emotion]])

# Save new CSV with image paths
df_new = pd.DataFrame(image_paths, columns=["image_path", "emotion"])
df_new.to_csv("image_paths.csv", index=False)

print("Conversion completed! Images saved in folders and CSV updated.")
```

Figure 17: Convert csv Snippet

As you can see in the snippet above, we used the tqdm library to monitor the progress bar while converting each row of pixel values into an image. We also used the numpy library to reshape the flat pixel array into a 2D image. Then, we saved each image into a corresponding folder based on the emotion label using the cv2 library (OpenCV).

Traditional machine learning algorithms cannot perform well with imbalanced datasets, which makes it difficult for the model to predict the correct emotion. For example, in the case of KNN with k=5, if there is an angry person in the picture, the model may incorrectly label the image as 'neutral' because there are 593 neutral samples in the dataset. This often leads to the model incorrectly labeling the image as 'neutral' even though the person is angry. To address this, we applied the data augmentation technique, as shown in the snippet below:

```python
# Define paths
input_folder = "data/"  # The folder where images are currently stored
output_folder = "split_data/"  # Folder for train/test split

# Define split ratio
train_ratio = 0.8  # 80% training, 20% testing

# Create output directories
train_folder = os.path.join(output_folder, "train")
test_folder = os.path.join(output_folder, "test")
os.makedirs(train_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

# Iterate through emotion folders
for emotion in os.listdir(input_folder):
    emotion_path = os.path.join(input_folder, emotion)
    if not os.path.isdir(emotion_path):
        continue  # Skip non-folder files

    images = os.listdir(emotion_path)
    random.shuffle(images)  # Shuffle images for randomness

    # Split dataset
    train_count = int(len(images) * train_ratio)
    train_images = images[:train_count]
    test_images = images[train_count:]

    # Create subdirectories
    os.makedirs(os.path.join(train_folder, emotion), exist_ok=True)
    os.makedirs(os.path.join(test_folder, emotion), exist_ok=True)

    # Move images to train folder
    for img in tqdm(train_images, desc=f"Moving {emotion} images to train"):
        src = os.path.join(emotion_path, img)
        dest = os.path.join(train_folder, emotion, img)
        shutil.copy(src, dest)

    # Move images to test folder
    for img in tqdm(test_images, desc=f"Moving {emotion} images to test"):
        src = os.path.join(emotion_path, img)
        dest = os.path.join(test_folder, emotion, img)
        shutil.copy(src, dest)
```

Figure 18: Split data snippet

In the snippet above we split our balanced data to a training and a testing set with a 80 to 20 percent ratio, using the help of os and shutil and random libraries to make sure that the data get split and copy it randomly to train and test folder while keeping track with the help of tqdm.

#### 5.2.1.2 CNN

For cnn it doesn't need augmentation technique since it does it at it's own but only on the training set however in this case the problem we have encountered a problem that which ck+ dataset it only has 5 samples in fear emotion after we split it in the test folder so we didn't have enough pictures to test with so we followed the augmentation technique as we did with knn and svm models.

### 5.2.2 Fer2013 dataset

#### 5.2.2.1 KNN and SVM

Since knn and svm are both traditional machine learning algorithms, we will need to augment and balance the data before training our models. For these reasons we will have to use the same technique we did with ck+ dataset.

#### 5.2.2.2 CNN

As for cnn model we don't need to augment or balence the data manually since there's enough samples in the test folder plus it also automatically augment the training data through data generators.

### 5.2.3 The structure of the data folder

Here's the final look on the data folder for both ck+ and fer2013 folders :

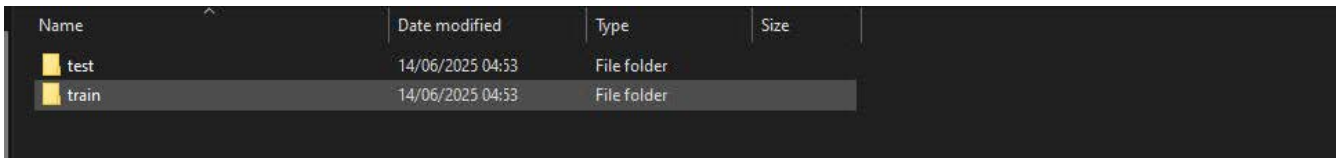



Figure 19: Screenshot from data folder

As we can there is two folder a test and train folder:

- **test folder:** This folder contains 7 subfolders (depending on the dataset ck+ or fer2013) each one labeled depending on the emotion and within the subfolder the images that the model will be tested.
- **train folder:** This folder contains 7 subfolders (depending on the dataset ck+ or fer2013) each one labeled depending on the emotion and within the subfolder the images that the model will be trained on.

## 5.3 Face feature extraction

### 5.3.1 Face extraction

#### 5.3.1.1 Haar-cascade

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, ”Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.[cas]

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.[cas]

#### 5.3.1.2 dlib

Dlib is a modern C++ (we also can use it in python) toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib’s open source licensing allows you to use it in any application, free of charge.[dli, b]

### 5.3.2 Feature extraction

For feature extraction we went for Histogram of Oriented Gradients (HOG) as it known for it's helping for recognizing different pattern in images mainly used for object detection or emotion recognition and what does it do it's basically takes the image and convert it to two separate images one with gradient color and the other with orientation color and it makes a table from the minimum orient color number to the highest one with a scale that u make then see each pixel in the gradient picture takes the value and see where it falls in the scale there where we put it in the table if already one exist we add it to it which gives us the final pictures, bellow we'll see how it works and the final picture :

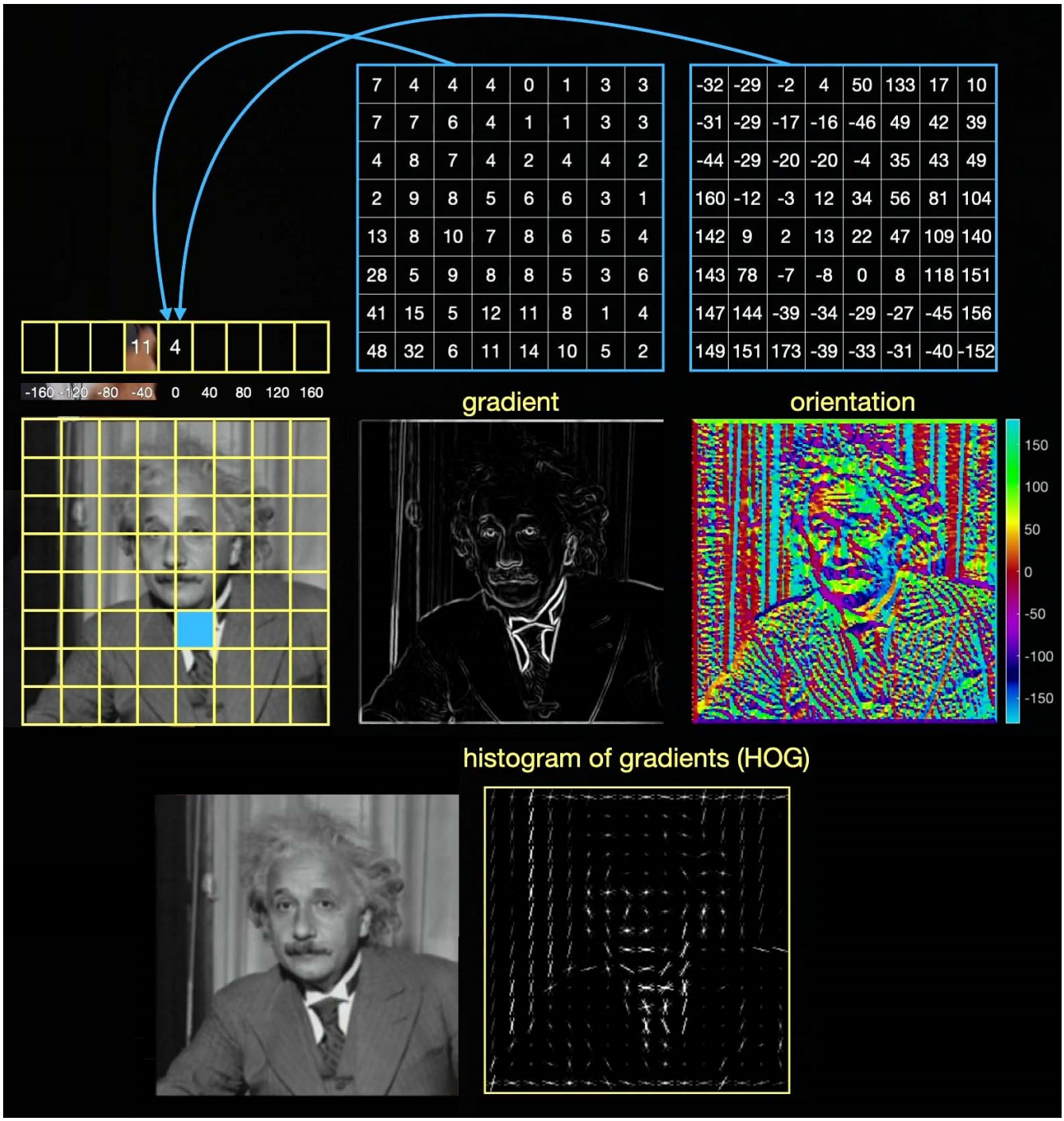



Figure 20: Histogram features extracted (HOG)

## 5.4 Models Training and results

Once we finished leaning the things mentioned above we need to setup all these libraries by inserting them in command prompt or the powershell then we go to our integrated development environment Visual Code

```
pip install numpy
pip install opencv-python
pip install tqdm
pip install scikit-learn
pip install matplotlib
pip install joblib
pip install dlib
pip install seaborn
pip install scikit-image
pip install tensorflow
pip install keras
```

Figure 21: Install libraries command

Now that we have all the requirement for our models we'll go through each model and the steps needed to finish our project.

In the following section we will show the models and the dataset that we will train it on.

### 5.4.1 KNN (Ck+dataset)

#### 5.4.1.1 Introduction

In this chapter, we will present the model's performance, accuracy, classification metrics, confusion matrix, robustness to noise, and complexity analysis, alongside the process we used to train the model.

#### 5.4.1.2 Data Preprocessing

```python
def extract_face(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        return None
    dets = face_detector(img, 1)
    if len(dets) == 0:
        return None
    x, y, x2, y2 = dets[0].left(), dets[0].top(), dets[0].right(), dets[0].bottom()
    x, y = max(0, x), max(0, y)
    x2, y2 = min(img.shape[1], x2), min(img.shape[0], y2)
    face = img[y:y2, x:x2]
    try:
        face = cv2.resize(face, IMG_SIZE)
    except:
        return None
    return hog(face, pixels_per_cell=(8,8), cells_per_block=(2,2), orientations=9)
```

Figure 22: face extraction function (KNN / ck+)

This snippet shows how we extracted the face from each image using dlib. Once the first face is detected, we get the coordinates and assign them to their respective variables. We then make sure it's within the boundaries, crop the face, resize it, and finally, we apply HOG to return the image as a 2D array that captures edges and texture information.

#### 5.4.1.3 Model Training

```python
knn = KNeighborsClassifier(n_neighbors=31, weights="distance")
```

Figure 23: Training the model (KNN / ck+)

KNN requires a parameter $k$ (`n_neighbors`) that signifies the number of neighbors we should take into account to classify a new sample. For our model, we selected 31, which follows the rule that suggests using the square root of the number of test samples to achieve a good balance. We also used distance weighting, which means that closer points have more influence than the others. We didn't use the metric parameter, which by default leads to the Euclidean distance, helping us improve our model.

#### 5.4.1.4 Performance Metrics

```
Training time: 9.78 s
Inference time per sample: 0.009892 s
Training data memory: 12.72 MB
```

Figure 24: Training the model (KNN / ck+)

The model takes 9.78 seconds to train, which is really fast. We have an average inference time of 0.009892 per sample, which means the model is really fast when used in real-world situations (for detecting emotions in an image). Since KNN stores the entire training dataset, 12.72 MB of RAM usage is quite low, and it holds all the features and labels, enabling fast predictions.

#### 5.4.1.5 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:

Classification Report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 1.00 | 1.00 | 1.00 | 474 |
| Disgust | 1.00 | 1.00 | 1.00 | 474 |
| Fear | 1.00 | 1.00 | 1.00 | 474 |
| Happiness | 1.00 | 1.00 | 1.00 | 474 |
| Sadness | 1.00 | 1.00 | 1.00 | 473 |
| Surprise | 1.00 | 1.00 | 1.00 | 474 |
| Neutral | 1.00 | 1.00 | 1.00 | 379 |
| Contempt | 1.00 | 1.00 | 1.00 | 474 |
| accuracy | | | 1.00 | 3696 |
| macro avg | 1.00 | 1.00 | 1.00 | 3696 |
| weighted avg | 1.00 | 1.00 | 1.00 | 3696 |

Figure 25: Classification report train (KNN / ck+)

All the emotions (precision, recall, score f1, accuracy) while testing:

Classification Report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 0.99 | 0.97 | 0.98 | 119 |
| Disgust | 1.00 | 0.99 | 1.00 | 119 |
| Fear | 0.99 | 0.98 | 0.99 | 119 |
| Happiness | 0.99 | 1.00 | 1.00 | 119 |
| Sadness | 0.95 | 0.98 | 0.97 | 119 |
| Surprise | 1.00 | 0.98 | 0.99 | 119 |
| Neutral | 0.88 | 0.85 | 0.87 | 95 |
| Contempt | 0.96 | 1.00 | 0.98 | 119 |
| accuracy | | | 0.97 | 928 |
| macro avg | 0.97 | 0.97 | 0.97 | 928 |
| weighted avg | 0.97 | 0.97 | 0.97 | 928 |

Figure 26: Classification report train (KNN / ck+)

In both of these figures, we can see how the model has been trained and tested, along with the outcomes for each emotion. We can also see its accuracy and the F1-score (a metric that balances precision and recall, providing a single score for evaluating the performance of a classification model).

#### 5.4.1.6 Confusion Matrix Figure

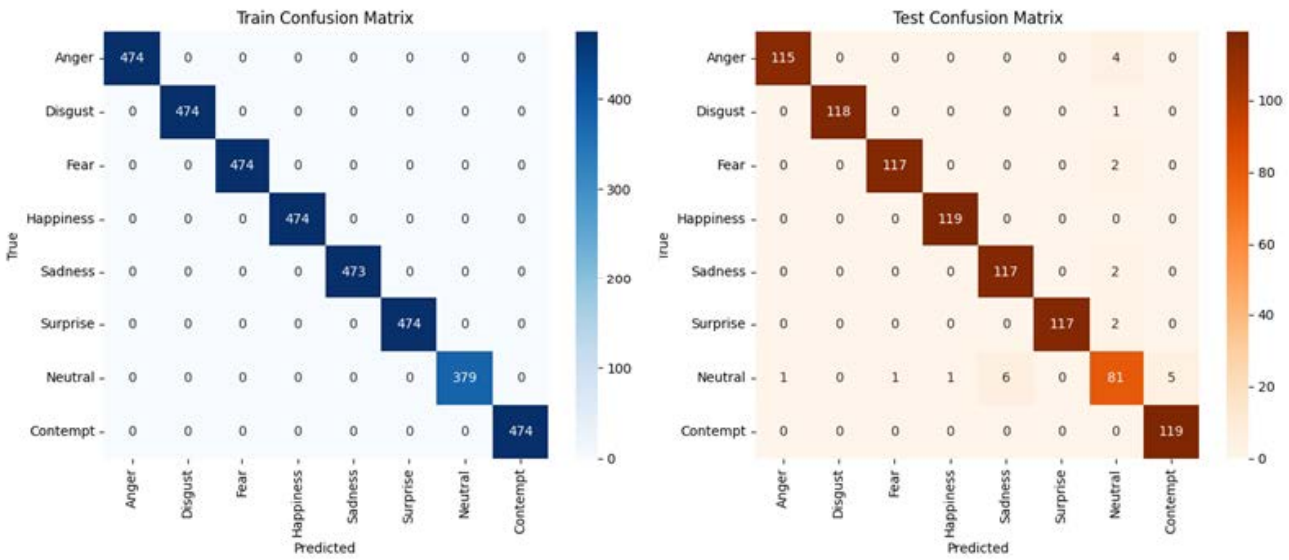



Figure 27: Confusion Matrix (KNN / ck+)

This figure provides a confusion matrix and will interpretaion.

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.
  - → Sadness ↔ Neutral
  - → Anger ↔ Disgust
  - → Neutral ↔ Anger

**Observation for knn:**

These above happened due to similar facial features in the data. If certain emotions have low precision/recall:

→ For neutral we had 100 images or so lower from the other sets of data so we can see the low recall of the data.

→ The emotion visually resembles another like common confusion patterns mentioned above.

#### 5.4.1.7 Accuracy, Complexity, Computational cost (prediction phase) and Robustness on noisy data

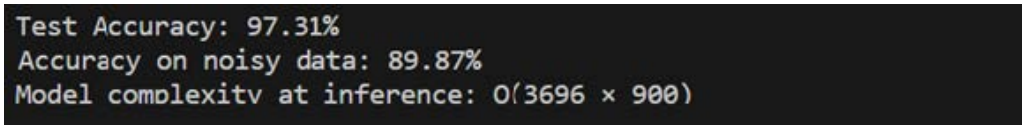



Figure 28: Accuracy, noisy data, complexity (KNN / ck+)

The accuracy achieved is 97.31%, which means the model performs well in classifying the data with the correct label.

Our model relies on KNN, and the complexity of KNN is mostly referred to as O(1) because the model only stores the data during the training phase. This means that the time required to train the model is constant, regardless of the number of training samples, since KNN models do not involve any learning or optimization steps.

We had a total of 3,685 training samples and 900 features per sample (taken from the images in the training set), which gives us approximately 3.3 million operations. Thus, for each new image we provide to the model, it must process these 3.3 million operations to make a single prediction.

In our case, we achieved a 89.87% accuracy on noisy data, which means reliable performance even when the input image contains noise. This indicates that the model will handle real-world situations where images might be noisy.

#### 5.4.1.8 Learning curve

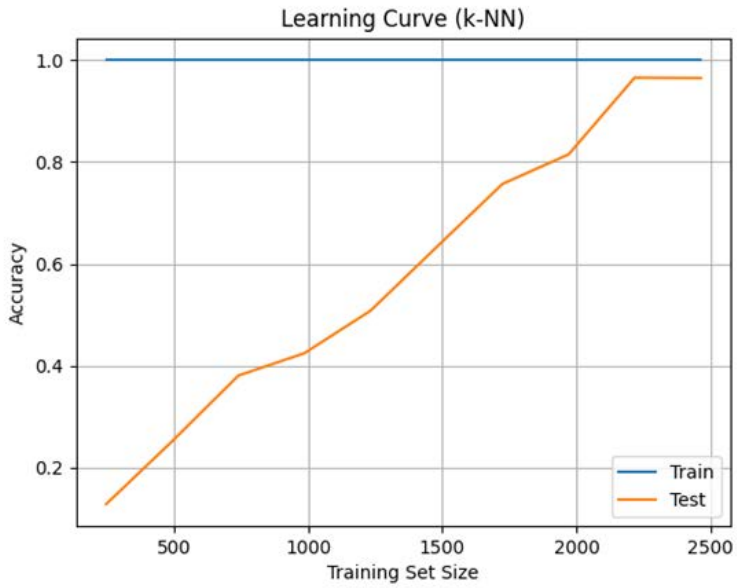



Figure 29: Learning curve (KNN / ck+)

As we can see in the figure above, the model learns more and more as more training data is provided. Its accuracy went up to 97.31% and then stopped, indicating that the model has captured the patterns in the training set.

### 5.4.2 SVM (Ck+dataset)

#### 5.4.2.1 Introduction

This is a presentation of the model's performance, accuracy, classification metrics, confusion matrix, robustness to noise, and complexity analysis.

#### 5.4.2.2 Model Training

```
svm = SVC(C=0.1, kernel='linear', gamma='scale')
```

Figure 30: Train model (SVM / ck+)

A linear kernel was used with a low C value (0.1), which is beneficial when the data is nearly linearly separable. The C value controls the trade-off between maximizing the margin and minimizing classification errors. A lower C value allows for a softer margin, where some misclassification is tolerated to improve generalization. This approach is often preferred for simpler datasets with clear boundaries between classes, such as CK+.

#### 5.4.2.3 Model performance

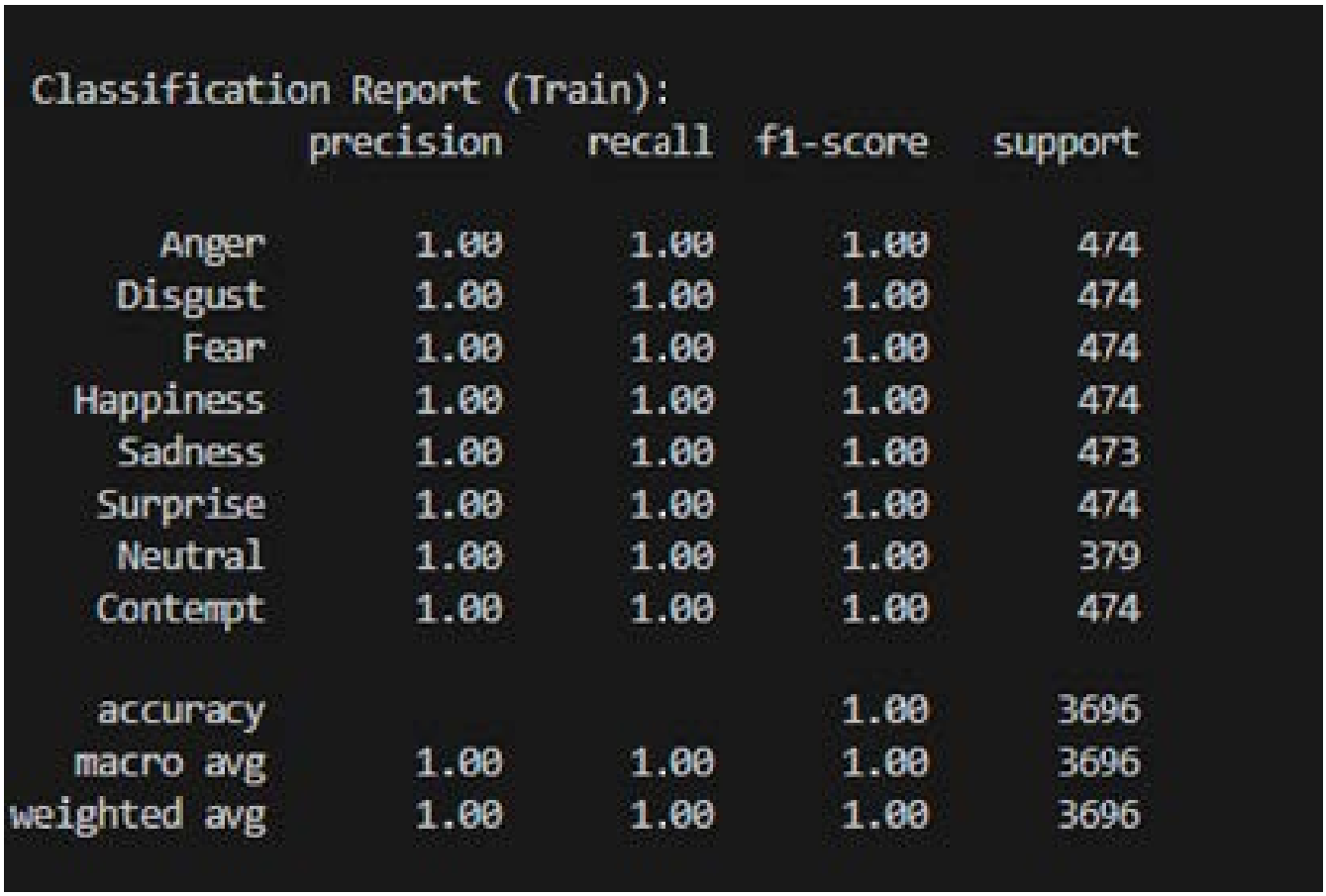
```
Training time: 1.07 s
Test Accuracy: 98.60%
Inference time per sample: 0.000790 s
Model size: 0.05 KB
Training data memory: 25.41 MB
Accuracy on noisy data: 83.08%
```

Figure 31: performance Metrics (SVM / ck+)

This image demonstrates how well the machine learning model performed after training and testing. It took just over a second (1.07 seconds) to train the model. When tested, the model achieved 98.6% accuracy, which is impressive. It can also make predictions extremely fast—less than a millisecond per sample. The model itself is compact, only 0.05 KB in size, making it suitable for use on devices with limited memory. The training data required approximately 25 MB of memory. The model was also tested with noisy data, and it still maintained over 83% accuracy, indicating its robustness in handling imperfect or noisy inputs.

#### 5.4.2.4 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:



Classification Report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 1.00 | 1.00 | 1.00 | 474 |
| Disgust | 1.00 | 1.00 | 1.00 | 474 |
| Fear | 1.00 | 1.00 | 1.00 | 474 |
| Happiness | 1.00 | 1.00 | 1.00 | 474 |
| Sadness | 1.00 | 1.00 | 1.00 | 473 |
| Surprise | 1.00 | 1.00 | 1.00 | 474 |
| Neutral | 1.00 | 1.00 | 1.00 | 379 |
| Contempt | 1.00 | 1.00 | 1.00 | 474 |
| accuracy | | | 1.00 | 3696 |
| macro avg | 1.00 | 1.00 | 1.00 | 3696 |
| weighted avg | 1.00 | 1.00 | 1.00 | 3696 |

Figure 32: Classification report train (SVM / ck+)

All the emotions (precision, recall, score f1, accuracy) while testing:

Classification Report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 0.96 | 0.98 | 0.97 | 119 |
| Disgust | 0.99 | 0.99 | 0.99 | 119 |
| Fear | 1.00 | 0.97 | 0.98 | 119 |
| Happiness | 1.00 | 1.00 | 1.00 | 119 |
| Sadness | 1.00 | 0.99 | 1.00 | 119 |
| Surprise | 1.00 | 1.00 | 1.00 | 119 |
| Neutral | 0.94 | 0.95 | 0.94 | 95 |
| Contempt | 0.99 | 1.00 | 1.00 | 119 |
| accuracy | | | 0.99 | 928 |
| macro avg | 0.98 | 0.99 | 0.98 | 928 |
| weighted avg | 0.99 | 0.99 | 0.99 | 928 |

Figure 33: Classification report test (SVM / ck+)

#### 5.4.2.5 Confusion Matrix

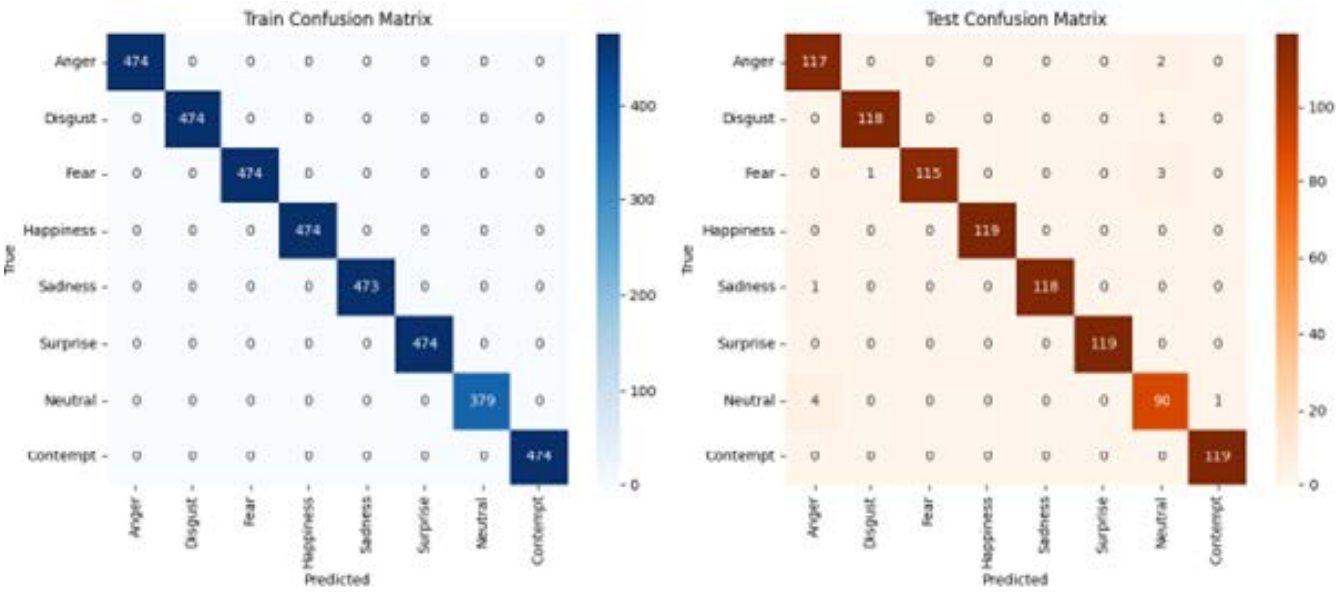



Figure 34: Confusion Matrix (SVM / ck+)

This figure provides a confusion matrix and its interpretation.

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.
  - → Anger ↔ Neutral
  - → Neutral ↔ Fear

**Observation for SVM:**

The problem with the 'neutral' class is not due to the SVM or KNN, but rather in the CK+ dataset itself. The neutral set has approximately 100 fewer images than the other sets, which explains the low recall for this class.

#### 5.4.2.6 Learning curve

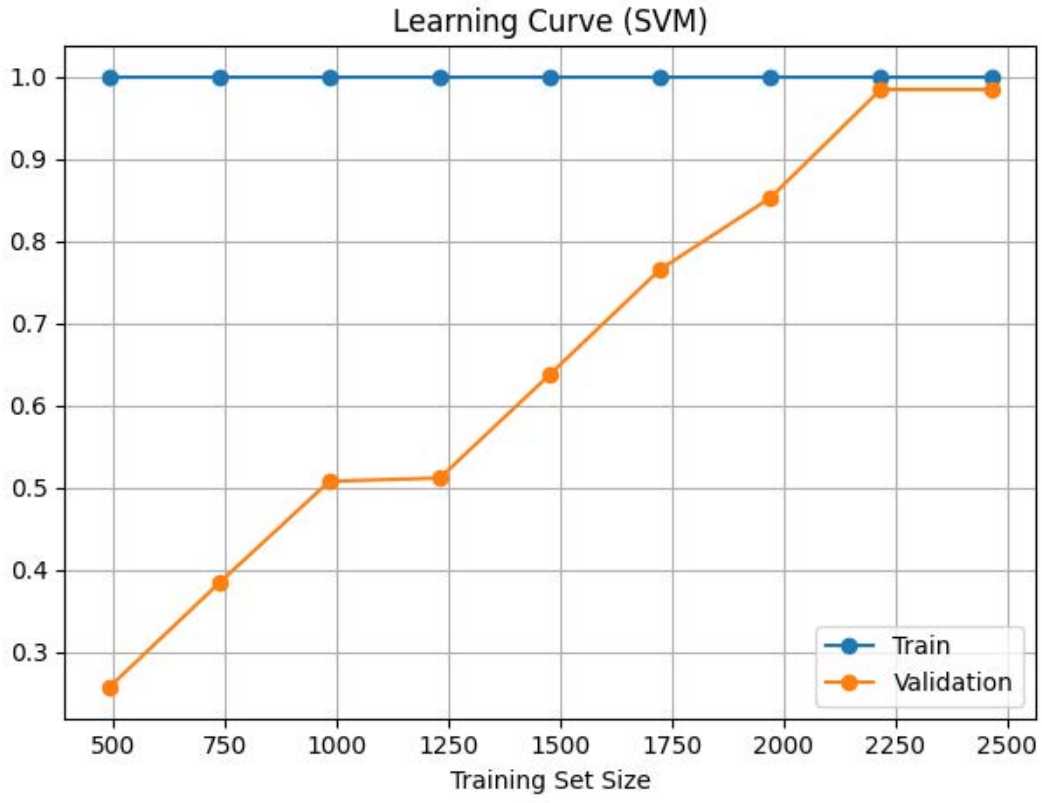



Figure 35: Learning curve (SVM / ck+)

We can see that the model learns more as more training data is provided. Its accuracy increased to 98.6% and then stopped, indicating that the model has captured the patterns in the training set.

### 5.4.3 CNN (Ck+ dataset)

#### 5.4.3.1 Introduction

As we know, CNNs go through multiple layers, and we can repeat this process multiple times. We chose 100 epochs, with the condition that the model should stop if it stops learning without any improvement (using callbacks).

#### 5.4.3.2 Model construction

```
# model construction
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1), padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 36: Model construction snipset (CNN / ck+)

The snippet above describes how we trained the model and how many layers the image goes through before predicting its correct label.

The model goes through four different layers, as explained in the CNN chapter: the convolu-

tional layer (Conv2D), followed by the pooling layer (MaxPooling2D), then the fully connected layer (Dense). There is also the ReLU activation function, which is used as a parameter in the convolutional layer.

In the first Conv2D layer, we applied 32 filters with 3x3 block size and ReLU activation, along with the same padding to preserve the input dimension of 48x48 pixels. This helps the model retain as much information as possible from the input sample in the early stages.

Next, we applied BatchNormalization, a technique used to stabilize and accelerate the training process.

After that, the MaxPooling2D layer uses a 2x2 block to reduce the dimension of the images while retaining the important information.

In the second Conv2D layer, we applied 64 filters to capture more features from the image using the same 3x3 block as in the previous layer. However, the dimensions of the image are now reduced. Following this, we apply BatchNormalization and MaxPooling2D layers to achieve the same effect.

In the final Conv2D layer, we applied 128 filters to further deepen the feature extraction process and capture more complex patterns in the image.

Then, the Flatten layer is used to convert the 3D output from the last pooling layer into a 1D vector, which is then passed to the Dense (fully connected) layer.

As mentioned earlier, in the Dense layer, we used 256 units, which is a common choice for most systems depending on hardware capabilities. We combined this with the ReLU activation function to contribute to the final decision.

To prevent overfitting, we applied dropout, with a rate of 0.5, meaning that half of the units are randomly dropped during training.

Finally, in the output Dense layer, we used the parameter num_class to ensure that we have 8 units (neurons) corresponding to the 8 possible classes. The softmax activation function is applied to make sure that the sum of the 8 neurons equals 1, producing a valid probability distribution for the final prediction.

#### 5.4.3.3 Accuracy curve while training and testing

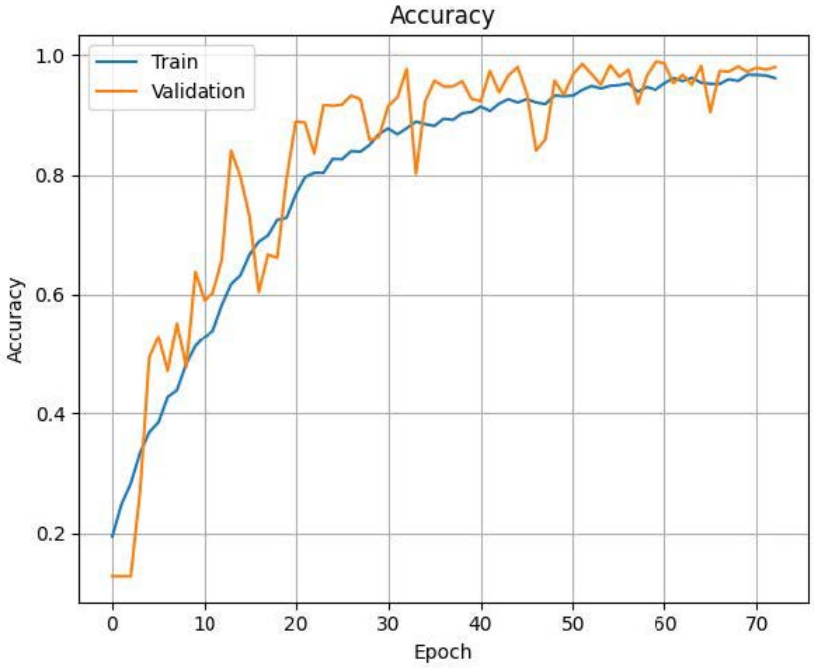



Figure 37: CNN accuracy curve (CNN/ ck+)

We can infer a lot of information from this figure. We can see that the model's training accuracy continues to improve steadily until it stabilizes around 0.98 after approximately 70 epochs. Additionally, we observe that the testing accuracy follows a similar trend, stabilizing around 0.98 as well.

We can conclude that the model is learning effectively and generalizing well due to its high accuracy. Furthermore, the small gap between the training and testing accuracy indicates that the model is not overfitting.

#### 5.4.3.4 Loss curve while training and testing

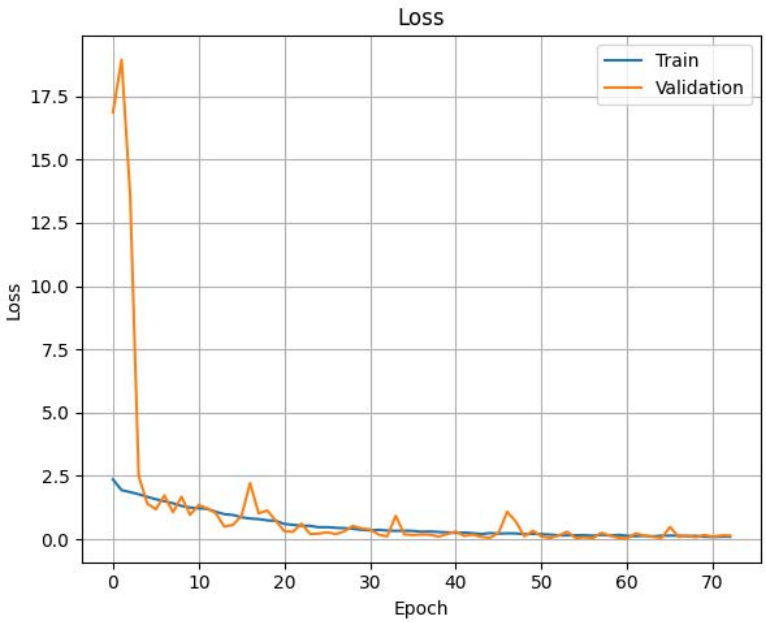



Figure 38: Loss accuracy (CNN / ck+)

The training loss decreases sharply and reaches nearly 0.01, which is excellent, meaning that the model is training the data very well. We can also see that the testing loss follows a similar pattern, which suggests no overfitting, indicating that the model is achieving better generalization and classification.

#### 5.4.3.5 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:

[RESULT] Classification report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 0.97 | 0.96 | 0.97 | 474 |
| Contempt | 1.00 | 1.00 | 1.00 | 474 |
| Disgust | 1.00 | 1.00 | 1.00 | 474 |
| Fear | 1.00 | 1.00 | 1.00 | 474 |
| Happiness | 1.00 | 1.00 | 1.00 | 474 |
| Neutral | 0.96 | 0.99 | 0.97 | 379 |
| Sadness | 1.00 | 0.97 | 0.98 | 473 |
| Surprise | 1.00 | 1.00 | 1.00 | 474 |
| accuracy | | | 0.99 | 3696 |
| macro avg | 0.99 | 0.99 | 0.99 | 3696 |
| weighted avg | 0.99 | 0.99 | 0.99 | 3696 |

Figure 39: Classification report train (CNN / ck+)

All the emotions (precision, recall, score f1, accuracy) while testing:

[RESULT] Classification report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 0.94 | 0.98 | 0.96 | 119 |
| Contempt | 1.00 | 1.00 | 1.00 | 119 |
| Disgust | 1.00 | 1.00 | 1.00 | 119 |
| Fear | 0.97 | 1.00 | 0.98 | 119 |
| Happiness | 1.00 | 1.00 | 1.00 | 119 |
| Neutral | 0.98 | 0.94 | 0.96 | 95 |
| Sadness | 1.00 | 0.96 | 0.98 | 119 |
| Surprise | 1.00 | 1.00 | 1.00 | 119 |
| accuracy | | | 0.99 | 928 |
| macro avg | 0.99 | 0.98 | 0.99 | 928 |
| weighted avg | 0.99 | 0.99 | 0.99 | 928 |

Figure 40: Classification report test (CNN / ck+)

#### 5.4.3.6 Confusion Matrix

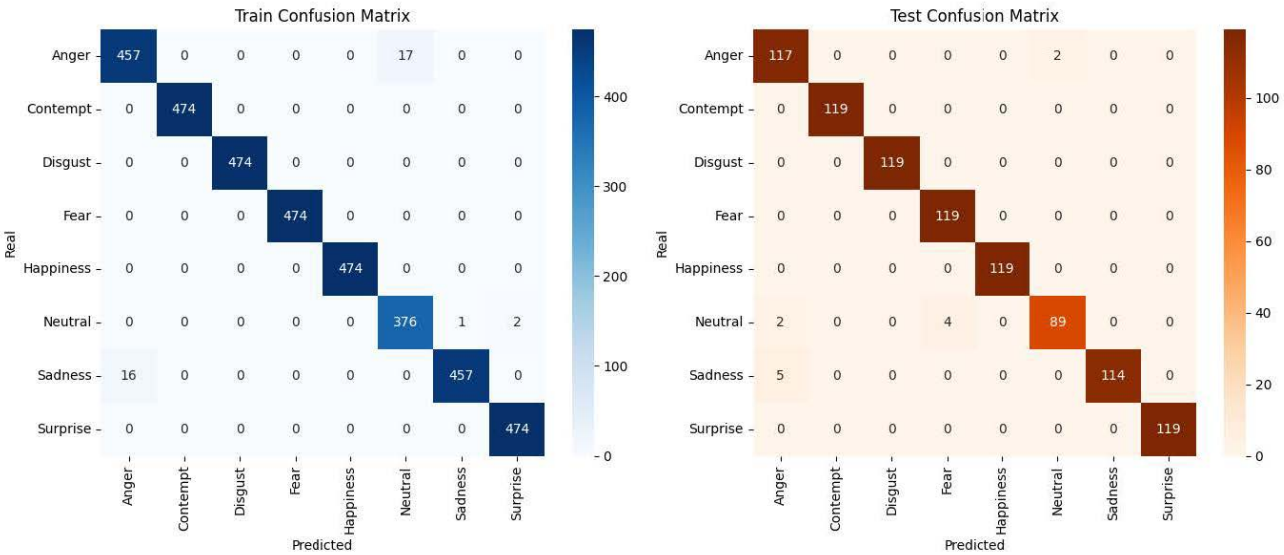



Figure 41: Confusion Matrix (CNN / ck+)

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.

  → Sadness ↔ Neutral

  → Anger ↔ Disgust

  → Neutral ↔ Anger

**Observation for cnn:**

We can observe from the matrix that the model trained and tested almost perfectly. The issue with the 'neutral' class being slightly off is due to the neutral folder having fewer samples than the other classes, which makes it slightly underrepresented. Other than that, the model is working perfectly well at identifying emotions

#### 5.4.3.7 Performance Metric

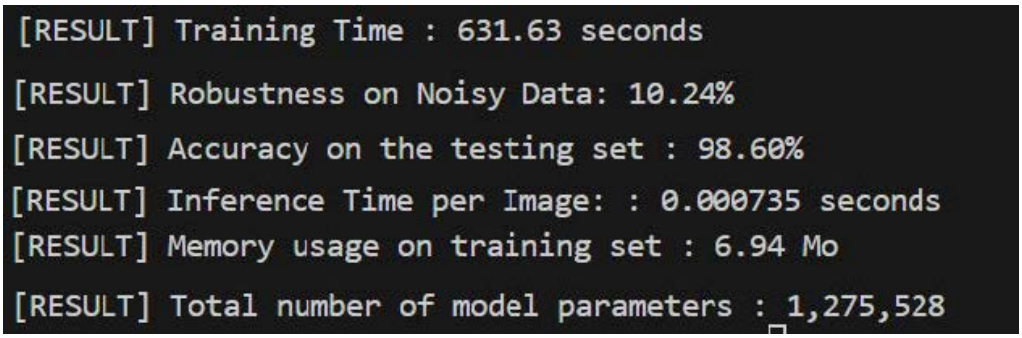



Figure 42: Performance metrics (CNN / ck+)

The training time for CNN is always important because it passes through many layers. Keep in mind that we are working with the CK+ dataset, which is relatively smaller than most other datasets. Therefore, 10 minutes (631.63 seconds) can be considered a low training time for a CNN model.

We can see that the CNN performs poorly on noisy data, with just 10.24% accuracy, which could be concerning for real-time situations.

The model accuracy is 98.60%, indicating that the model is learning very well and performing accurately on the testing set.

In terms of inference speed, it takes a very small amount of time to process an image—just 0.0007 seconds—which is extremely fast, even though the model still requires 10 minutes for training.

As for memory usage, it is only 6.94 MB. By today's standards, this is not a large memory footprint and is relatively low when compared to the KNN model.

### 5.4.4 Ck+ dataset summary

| Criteria | Advantages of CNN | Disadvantages of CNN | Advantages of KNN | Disadvantages of KNN | Advantages of SVM | Disadvantages of SVM |
|---|---|---|---|---|---|---|
| Overall accuracy | Very good (98.60%) | Sensitive to disturbances | Good (97.31%) | Less accurate than CNN/SVM | Best (98.60%) | Slight drop in some classes |
| Noise robustness | Weak (10.24%) | Sensitive to variations | Very robust (89.87%) | No control over generalization | Good (83.08%) | Less resistant than KNN |
| Training time | 631.63s | Slow | 9.78s | slower than SVM | 1.07s Ultra-fast | |
| Inference time | Ultra-fast (0.000735s) | Requires GPU for optimal speed | Correct (0.009892s) | Much slower than CNN and SVM | Fast (0.000790s) | A little slower than CNN |
| Occupied memory | Weak (6.94 MB) | Consumes a lot of resources for training | Moyenne (12.72 MB) | Stores all data → high memory | Compact (0.05 KB) | May be unstable with unbalanced data |
| Complexity | Fixed after training | High training cost (672s) | Simple to understand | O(3696 ×900) | Fixed and optimized | Can be difficult to interpret |
| Scalability | Very good | Requires a lot of computing | Weak | Becomes very slow with large dataset | Good | May be limited by the number of classes |

Figure 43: Ck+ summary

**CNN, SVM, and KNN demonstrate high performance, but their strengths differ:**

- SVM and CNN have the highest accuracy (98.60%), making them ideal for pure classification systems.
- CNN is the fastest in inference (0.000564s), making it optimal for real-time applications, such as emotion tracking in video.
- KNN is the most robust to perturbations (89.87%), making it preferable for noisy environments or degraded images.

Comparison of each model with each emotion:

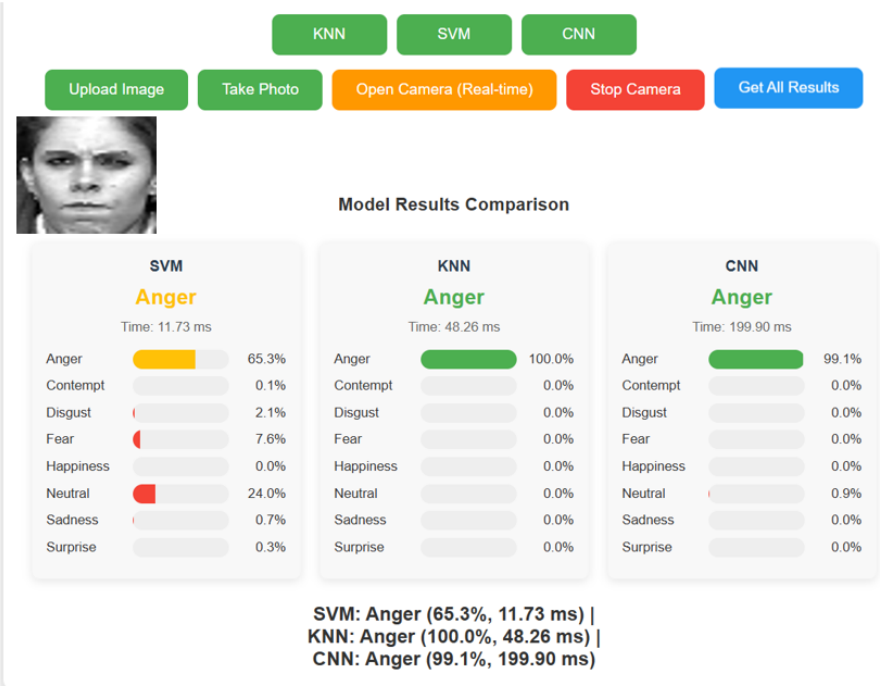



Figure 44: Anger (Ck+ dataset)

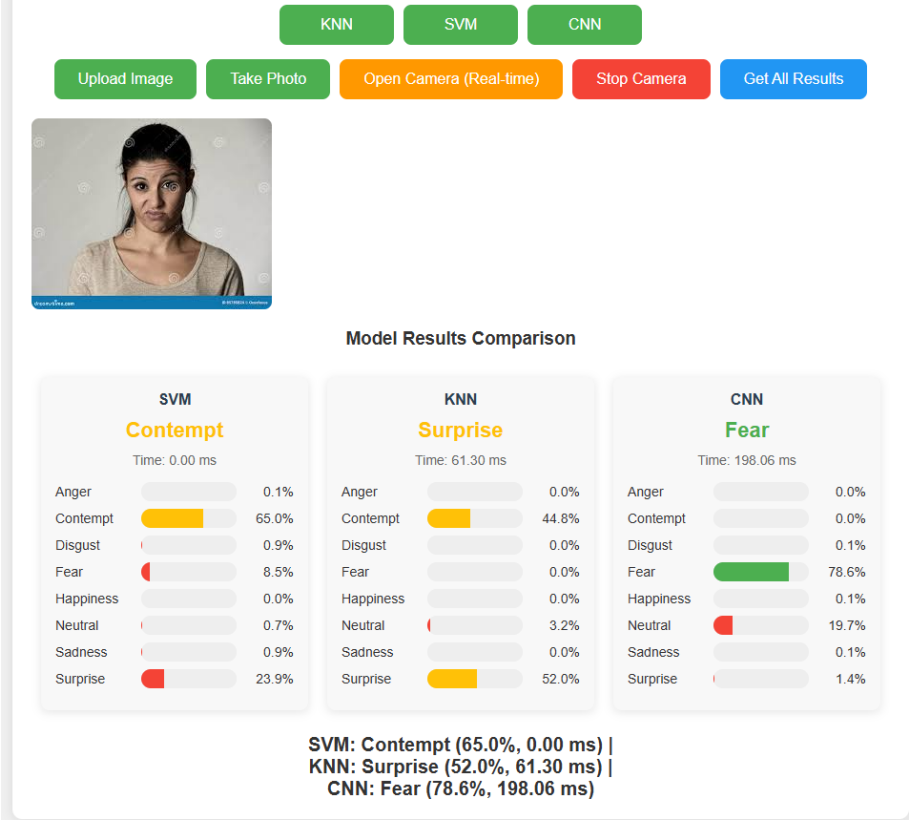



Figure 45: Contempt (Ck+ dataset)

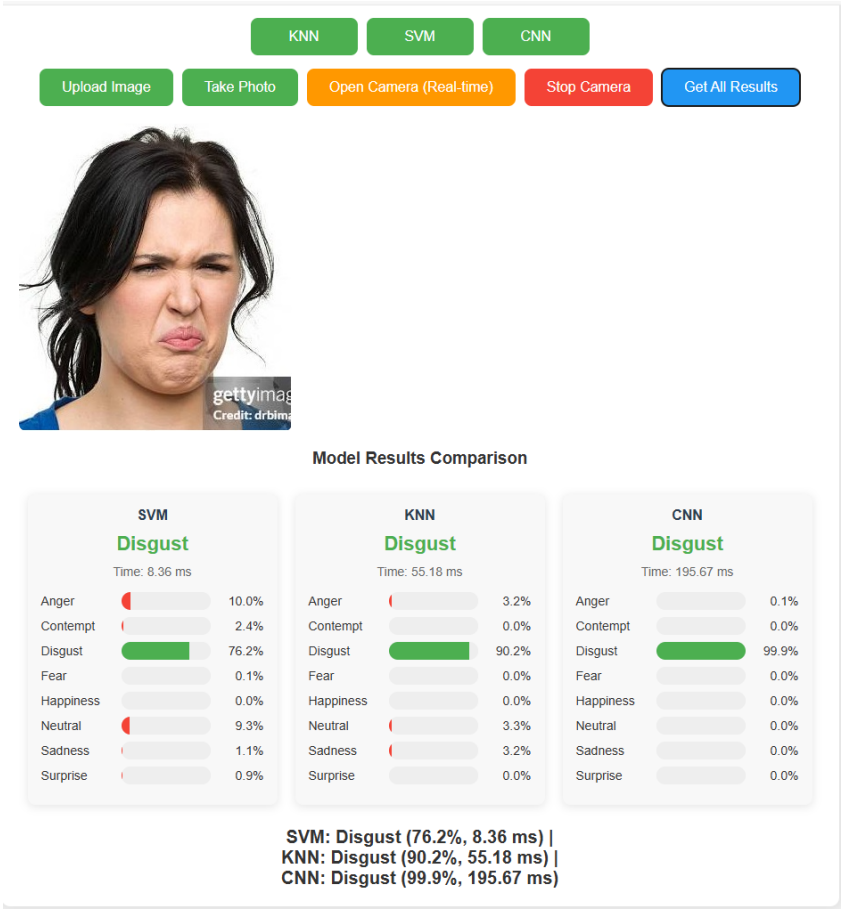



Figure 46: Disgust (Ck+ dataset)

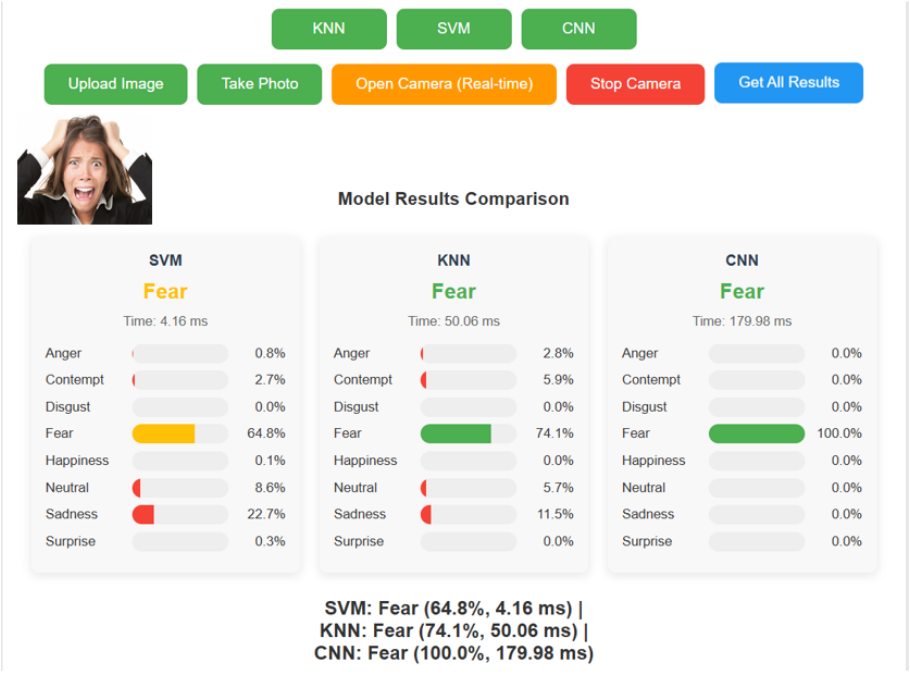



Figure 47: Fear (Ck+ dataset)

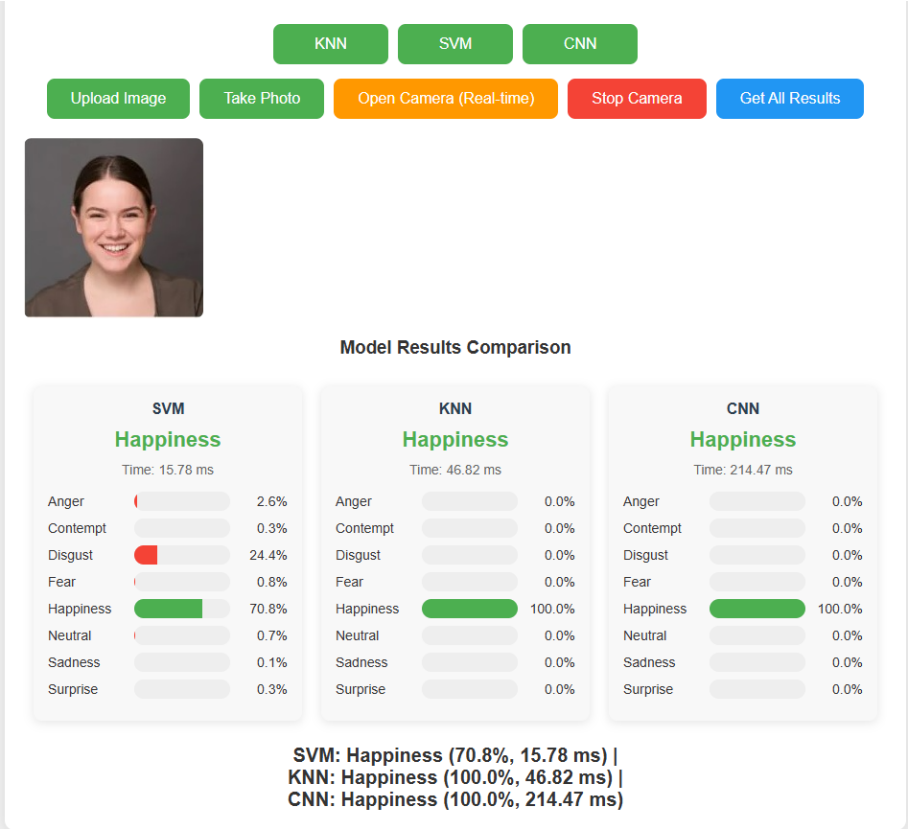



Figure 48: Happiness (Ck+ dataset)

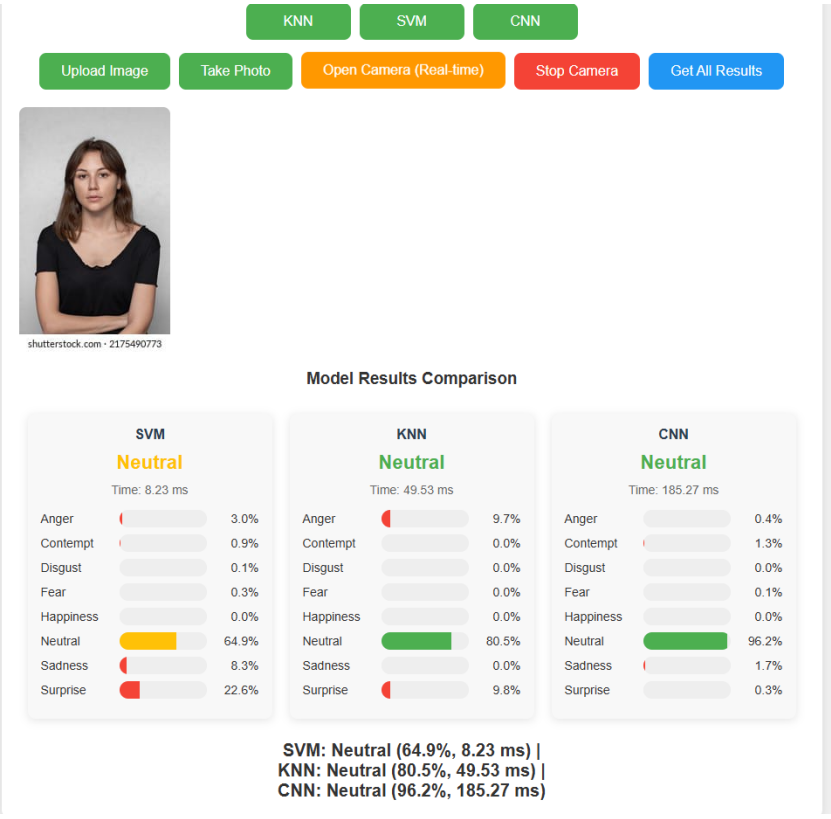



Figure 49: Neutral (Ck+ dataset)

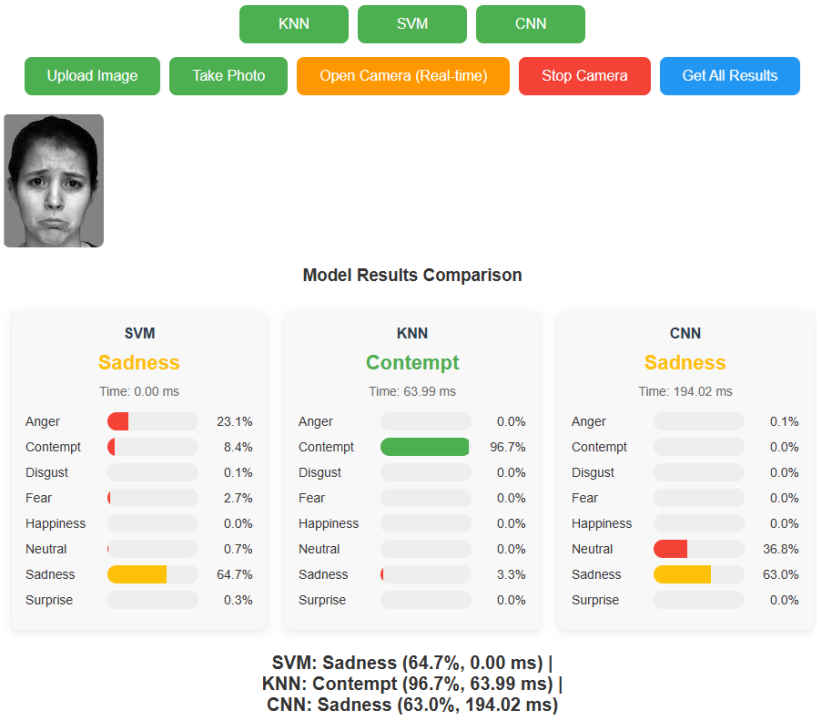



Figure 50: Sadness (Ck+ dataset)

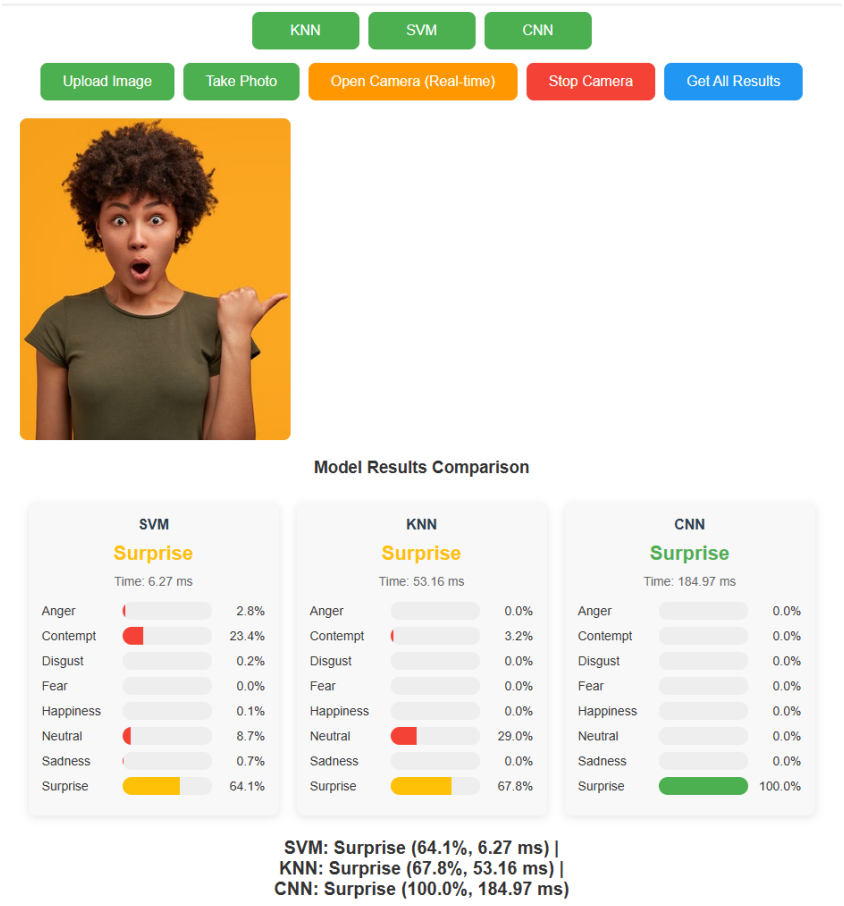



Figure 51: Surprise (Ck+ dataset)

We can see that the models are generally doing a great job classifying each image to its correct label, although each has its own challenges. K-NN misclassifies sadness and contempt, while CNN also misclassifies contempt, likely due to similar facial expressions in the CK+ dataset. On the other hand, SVM performed excellently, classifying all emotions with the correct labels.

In terms of time, SVM took significantly less time than the other models for each prediction. Next is K-NN, which took a little more time—about 50ms ahead of CNN. Finally, CNN took significantly more time to classify the image, with 120ms more than K-NN.

Although these time differences may seem small, if we used larger datasets, the time differences would become much more significant, with the CNN model taking considerably more time than the others.

**Conclusion** :

SVM excels in accuracy and fast prediction time, while CNN, although slower, might offer better performance in real-time due to its deep learning capabilities. KNN handles noise efficiently, with an average time between the other two models.

### 5.4.5 KNN (Fer2013 dataset)

#### 5.4.5.1 Introduction

In this chapter, we will present the model's performance, accuracy, classification metrics, confusion matrix, robustness to noise, and complexity analysis, alongside how we trained the model.

#### 5.4.5.2 Data Preprocessing

```
print("[INFO] Applying PCA...")
pca = PCA(n_components=100)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

Figure 52: Applying pca (k-NN / fer2013)

For FER2013, we applied PCA (Principal Component Analysis) to reduce the dimensionality of the images while retaining the most important information, reducing it to 100 components in

our case. We also ensured that both x_train and x_test were in the same feature space.

Additionally, PCA can help reduce the training time and prevent overfitting.

```python
def extract_face(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        return None
    dets = face_detector(img, 1)
    if len(dets) == 0:
        return None
    x, y, x2, y2 = dets[0].left(), dets[0].top(), dets[0].right(), dets[0].bottom()
    x, y = max(0, x), max(0, y)
    x2, y2 = min(img.shape[1], x2), min(img.shape[0], y2)
    face = img[y:y2, x:x2]
    try:
        face = cv2.resize(face, IMG_SIZE)
    except:
        return None
    return hog(face, pixels_per_cell=(8,8), cells_per_block=(2,2), orientations=9)
```

Figure 53: Face extraction function (k-NN / fer2013)

This snippet shows how we extracted the face from each image using dlib. Once the first face is detected, we obtain the coordinates and assign them to their respective variables. We then ensure that the face is within the boundaries, crop it, resize it, and finally apply HOG to return the image as a 2D array that captures edges and texture information.

#### 5.4.5.3 Model Training

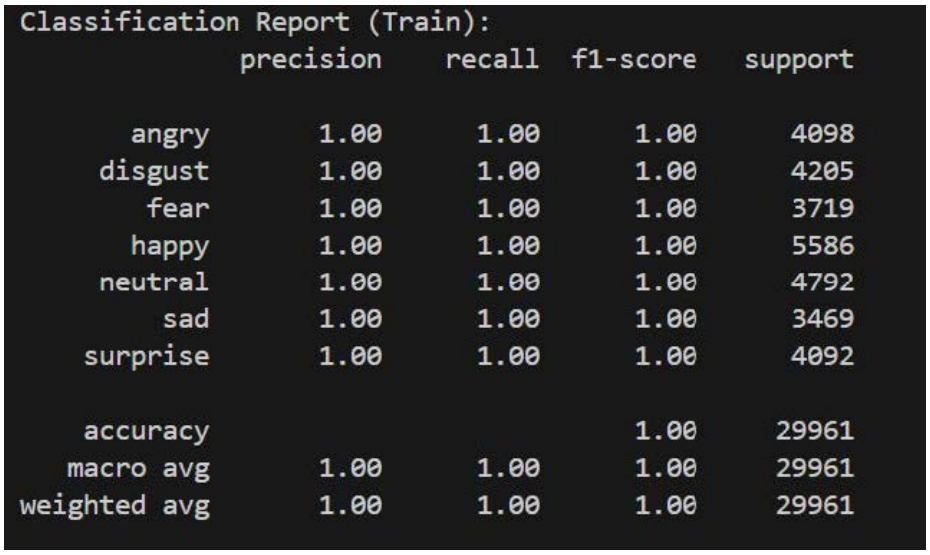
```python
knn = KNeighborsClassifier(n_neighbors=5, weights='distance',metric='cosine')
```

Figure 54: Training model (k-NN / fer2013)

K-NN requires a parameter $k$ (`n_neighbors`) that signifies the number of neighbors we should consider to classify a new data point. For our model, we selected 5, meaning it will consider the 5 closest points to it. We also used distance weighting, which means that closer points have more influence than others. Finally, in the metric, we used cosine, which means the model will consider the angle between the vectors rather than the Euclidean distance. This makes it effective for large datasets like FER2013.

#### 5.4.5.4 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:

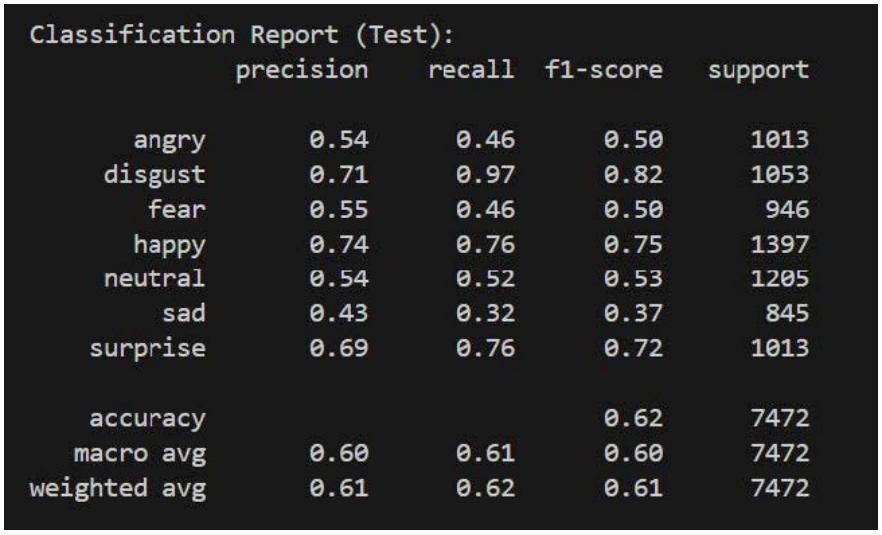


Classification Report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| angry | 1.00 | 1.00 | 1.00 | 4098 |
| disgust | 1.00 | 1.00 | 1.00 | 4205 |
| fear | 1.00 | 1.00 | 1.00 | 3719 |
| happy | 1.00 | 1.00 | 1.00 | 5586 |
| neutral | 1.00 | 1.00 | 1.00 | 4792 |
| sad | 1.00 | 1.00 | 1.00 | 3469 |
| surprise | 1.00 | 1.00 | 1.00 | 4092 |
| accuracy | | | 1.00 | 29961 |
| macro avg | 1.00 | 1.00 | 1.00 | 29961 |
| weighted avg | 1.00 | 1.00 | 1.00 | 29961 |

Figure 55: Classification report train (k-NN / fer2013)

All the emotions (precision, recall, score f1, accuracy) while testing:



Classification Report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| angry | 0.54 | 0.46 | 0.50 | 1013 |
| disgust | 0.71 | 0.97 | 0.82 | 1053 |
| fear | 0.55 | 0.46 | 0.50 | 946 |
| happy | 0.74 | 0.76 | 0.75 | 1397 |
| neutral | 0.54 | 0.52 | 0.53 | 1205 |
| sad | 0.43 | 0.32 | 0.37 | 845 |
| surprise | 0.69 | 0.76 | 0.72 | 1013 |
| accuracy | | | 0.62 | 7472 |
| macro avg | 0.60 | 0.61 | 0.60 | 7472 |
| weighted avg | 0.61 | 0.62 | 0.61 | 7472 |

Figure 56: Classification report test (k-NN / fer2013)

In both of these figures, we can see how the model has been trained and tested, with all the outcomes for each emotion. We can also observe its accuracy along with the F1 score (a metric that balances precision and recall, providing a single score for evaluating the performance of a classification model).

#### 5.4.5.5 Confusion Matrix

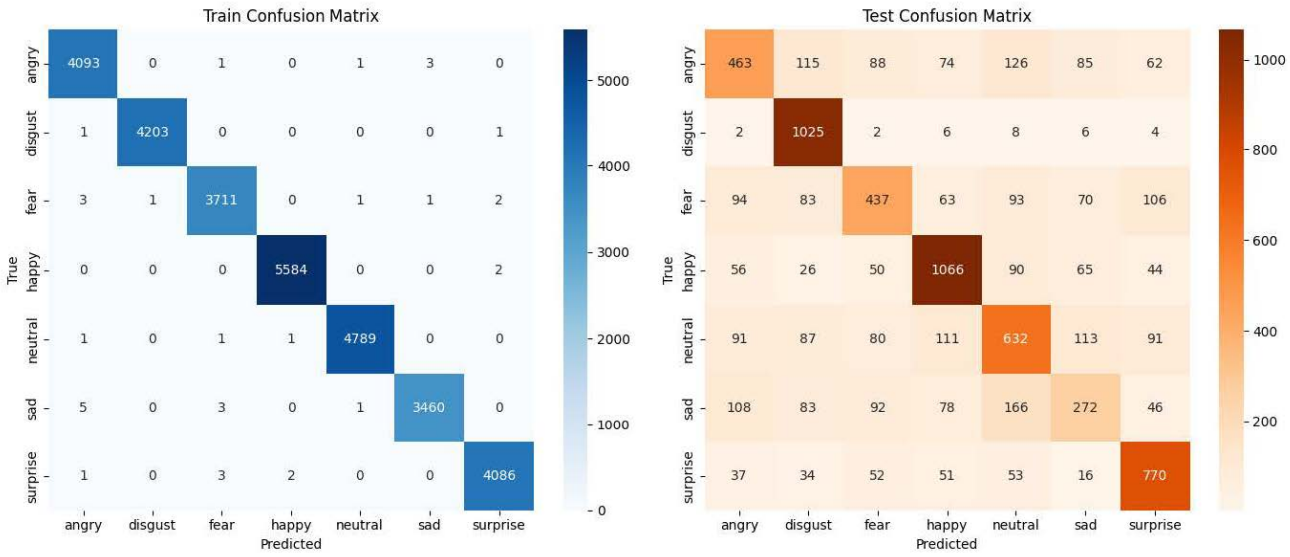



Figure 57: Confusion Matrix (k-NN / fer2013)

This figure provides a confusion matrix and will interpretaion.

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.

  → Sadness ↔ Neutral

  → Anger ↔ Disgust

  → Neutral ↔ Anger

**Observation for knn:**

These issues occurred due to similar facial features in the data. If certain emotions have low precision or recall:

→ For the neutral class, we had about 100 fewer images compared to the other datasets, which explains the low recall.

→ The emotion visually resembles another, like the common confusion patterns mentioned above.

→ We can also see that KNN might struggle with large datasets.

#### 5.4.5.6 Accuracy, Complexity, Computational cost (prediction phase) and Robustness on noisy data

```
Training time: 513.83 s
Test Accuracy: 62.43%
Inference time per sample: 0.010527 s
Training data memory: 103.09 MB
Accuracy on noisy data: 57.45%
[INFO] Model complexity at inference: O(29961 × 900)
```

Figure 58: Accuracy, noisy data, complexity (KNN / fer2013)

The accuracy achieved is 62.43%, which means the model performs reasonably well in classifying the data with the correct label when using the FER2013 dataset. This outcome suggests that K-NN might be limited when handling complex datasets.

Regarding the training time, it took nearly 5 minutes, which is relatively low. Additionally, the inference time per sample was just 0.01 seconds, which is decent. The model also used 103 MB of memory during training, which is good in terms of RAM usage, as K-NN stores the entire dataset in memory.

Our model relies on K-NN, and the complexity of K-NN is mostly referred to as O(1), because the model only stores the data during the training phase. This means that the time required to train the model is constant, regardless of the number of training samples, since K-NN models do not involve any learning or optimization steps.

We had a total of 29,961 training samples and 900 features per sample (taken from the images in the training set), resulting in approximately 26.9 million operations. For each new image we provide to the model, it must process these 26.9 million operations to make a single prediction.

In our case, we achieved 57.45% accuracy on noisy data. This indicates that the model struggled with nearly half of the noisy images, which might negatively affect its performance in real-world situations.

#### 5.4.5.7 Learning curve

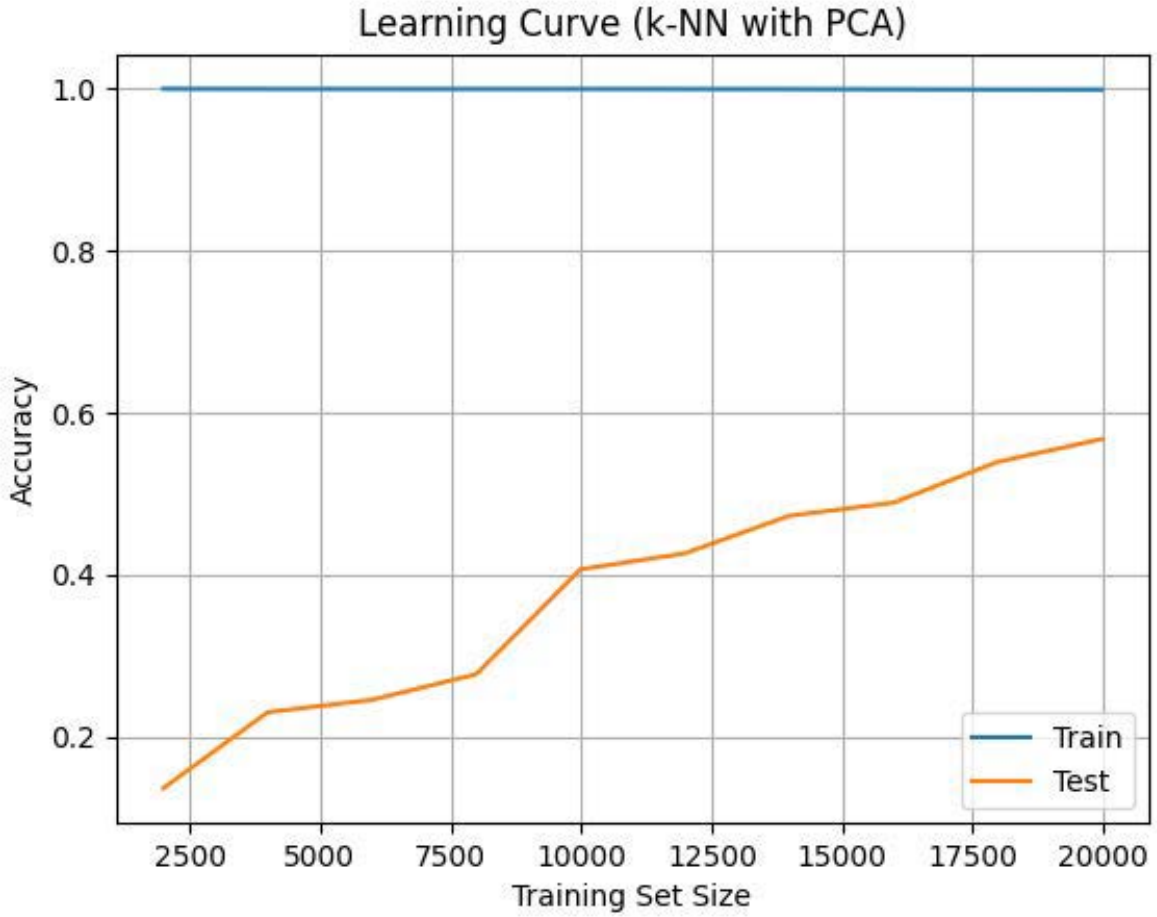



Figure 59: Accuracy, noisy data, complexity (KNN / fer2013)

### 5.4.6 SVM (Fer2013 dataset)

#### 5.4.6.1 Introduction

In this chapter, we will present the model's performance, accuracy, classification metrics, confusion matrix, robustness to noise, and complexity analysis, alongside how we trained the model.

#### 5.4.6.2 Model Training

```
svm = SVC(C=1.0, kernel='rbf', gamma='scale'
```

Figure 60: Training the model (SVM / fer2013)

An RBF (Radial Basis Function) kernel was selected for the FER2013 dataset due to its ability to handle complex, non-linear relationships between features. The RBF kernel transforms the data into a higher-dimensional space, making it easier to find a separating hyperplane. The gamma='scale' parameter helps normalize the kernel's behavior by considering the dataset's feature size, which can improve model performance by making the model more adaptable and less sensitive to noise.

#### 5.4.6.3 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:

Classification Report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| angry | 0.91 | 0.89 | 0.90 | 5586 |
| disgust | 0.93 | 0.98 | 0.96 | 5586 |
| fear | 0.95 | 0.86 | 0.90 | 5586 |
| happy | 0.93 | 0.96 | 0.94 | 5586 |
| neutral | 0.88 | 0.91 | 0.90 | 5586 |
| sad | 0.93 | 0.92 | 0.93 | 5586 |
| surprise | 0.94 | 0.93 | 0.93 | 5586 |
| accuracy | | | 0.92 | 39102 |
| macro avg | 0.92 | 0.92 | 0.92 | 39102 |
| weighted avg | 0.92 | 0.92 | 0.92 | 39102 |

Figure 61: Classification report train (SVM / fer2013)

All the emotions (precision, recall, score f1, accuracy) while testing:

Classification Report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anger | 0.96 | 0.98 | 0.97 | 119 |
| Disgust | 0.99 | 0.99 | 0.99 | 119 |
| Fear | 1.00 | 0.97 | 0.98 | 119 |
| Happiness | 1.00 | 1.00 | 1.00 | 119 |
| Sadness | 1.00 | 0.99 | 1.00 | 119 |
| Surprise | 1.00 | 1.00 | 1.00 | 119 |
| Neutral | 0.94 | 0.95 | 0.94 | 95 |
| Contempt | 0.99 | 1.00 | 1.00 | 119 |
| accuracy | | | 0.99 | 928 |
| macro avg | 0.98 | 0.99 | 0.98 | 928 |
| weighted avg | 0.99 | 0.99 | 0.99 | 928 |

Figure 62: Classification report test (SVM / fer2013)

#### 5.4.6.4 Confusion Matrix

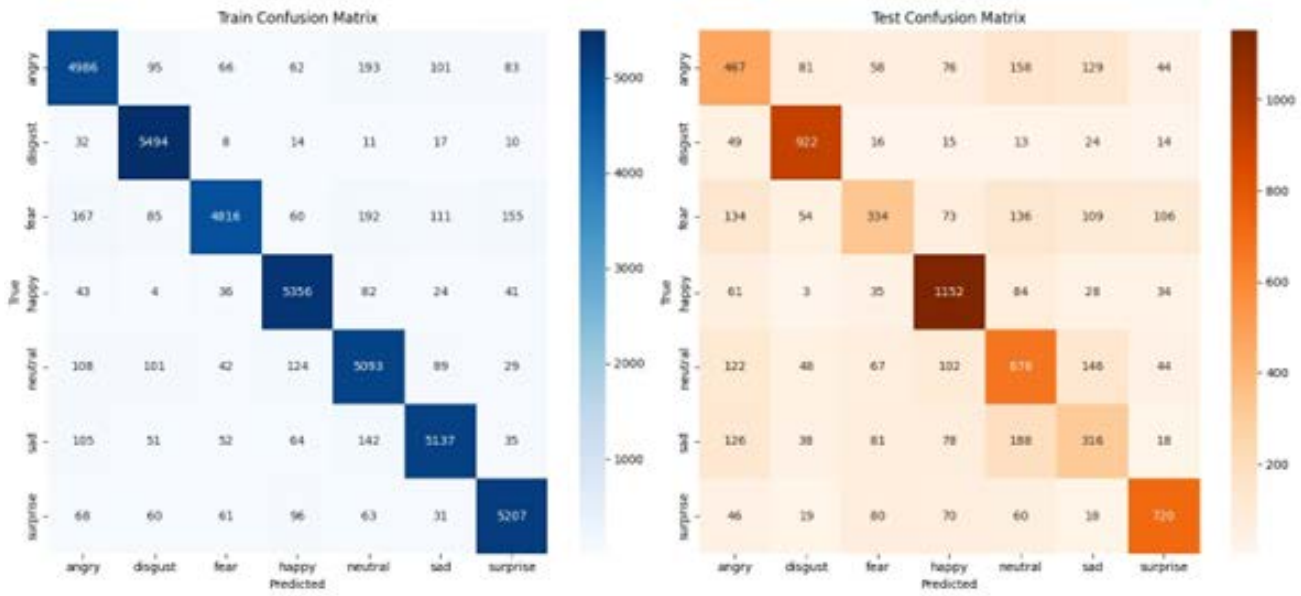



Figure 63: Confusion Matrix (SVM / fer2013)

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.
  - → Sadness ↔ Neutral
  - → Anger ↔ Disgust
  - → Neutral ↔ Anger

**Observation for SVM:**

The problem with the 'neutral' class is not due to the SVM or KNN, but rather in the FER2013 dataset itself. The neutral class has approximately 100 fewer images than the other classes, which explains the low recall.

#### 5.4.6.5 Accuracy, Complexity, Computational cost (prediction phase) and Robustness on noisy data

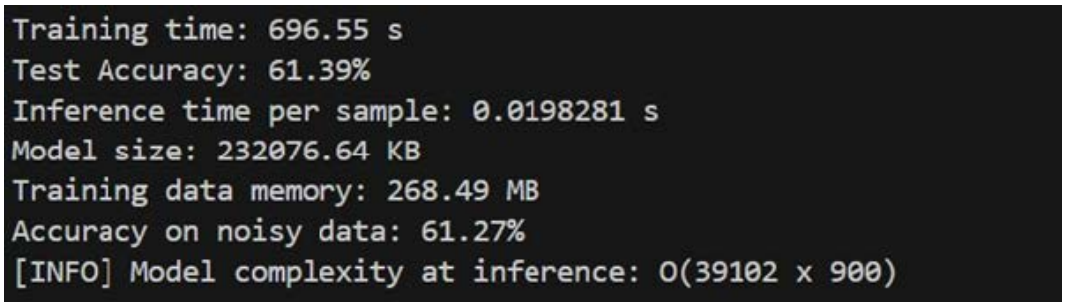



Figure 64: Accuracy, noisy data, complexity (SVM / fer2013)

The model achieves a test accuracy of 61.39%, indicating that it is moderately effective in classifying the data correctly. While the SVM model generally performs well in classification tasks, there is still potential for improvement, especially with more complex datasets.

The training time of 696.55 seconds is relatively moderate and suggests that the SVM model may require significant resources and time for large datasets. However, the inference time per sample (0.019 seconds) is quite efficient, making it suitable for real-time predictions.

The model size of 232KB and the memory usage of 268.49MB during training are reasonable, indicating that the model is not too demanding for the given resources. The model complexity

at inference (O(39102 x 900)) suggests that it is handling a large number of operations, given the considerable number of features and samples.

Despite its performance, the SVM model faced challenges with noisy data, achieving an accuracy of 61.27% on such data. This implies that it may not be as robust in real-world scenarios, where noise and uncertainty are more prevalent.

In conclusion, while the SVM model shows good potential, especially in terms of memory and computational efficiency, it could benefit from better handling of noisy data and further optimization for more complex datasets.

#### 5.4.6.6 Learning curve

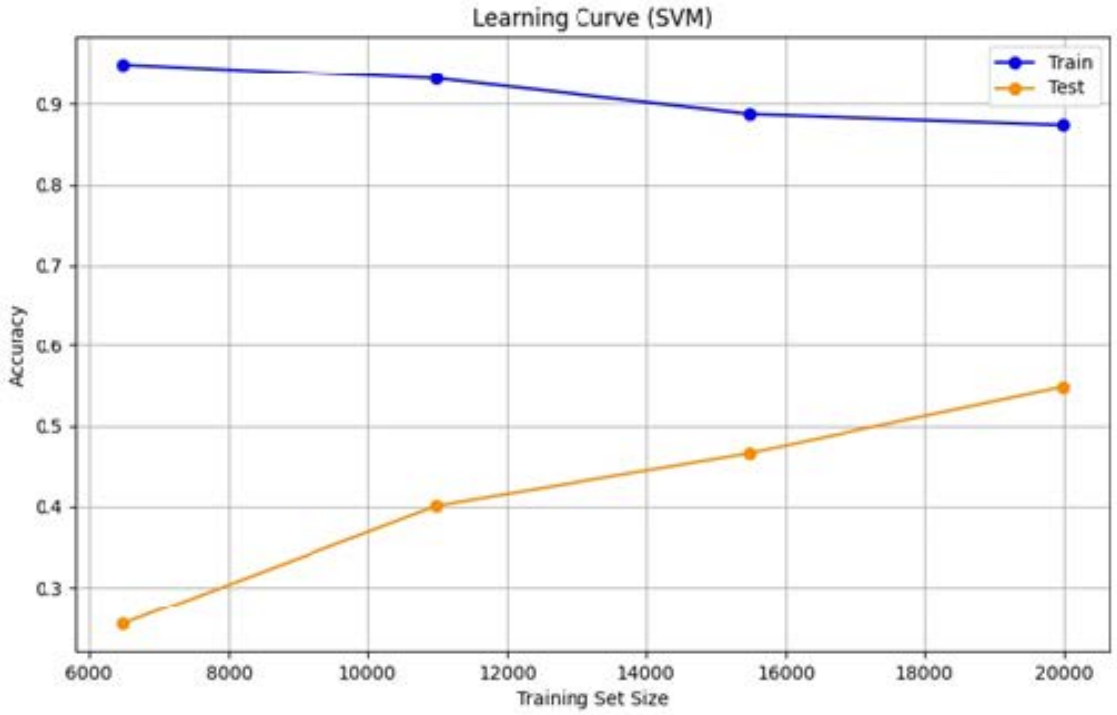



Figure 65: Accuracy, noisy data, complexity (SVM / fer2013)

### 5.4.7 CNN (Fer2013 dataset)

#### 5.4.7.1 Introduction

As we know, CNNs pass through multiple layers, and we can repeat this process multiple times. We chose 120 epochs, with the condition that the model should stop if it stops learning

without any improvement (using callbacks). We also used a batch size of 64, doubling the normal batch size used for CNN.

#### 5.4.7.2 Model construction

```
model = Sequential([
    Input(shape=(48, 48, 1)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.35),

    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.4),
    Flatten(),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(len(class_labels), activation='softmax')
```

Figure 66: Model construction snipset (CNN / Fer2013)

The snippet above describes how we trained the model and the number of layers the image goes through before predicting its correct label.

The model goes through four main layers, as explained in the CNN chapter: the convolutional layer (Conv2D), followed by the pooling layer (MaxPooling2D), and then the fully connected layer (Dense). The ReLU activation function is applied as a parameter in the convolutional layer.

In the first Conv2D layer, we applied 64 filters with a 3x3 kernel and ReLU activation, using same padding to preserve the input dimension of 48x48 pixels. This helps the model retain as much information as possible from the sample in the early stages.

Next, we applied BatchNormalization, a technique used to stabilize and accelerate the training process.

We applied the Conv2D layer twice to extract more features, followed by BatchNormalization after each convolution.

Then, the MaxPooling2D layer with a 2x2 block was used to reduce the image dimensions

while retaining the most important information. We also applied a dropout rate of 0.25, and increased the dropout rate in subsequent layers to prevent overfitting.

In the second Conv2D layer, we applied 128 filters to capture more features from the image, using the same 3x3 kernel. The dimensions were reduced after this layer, followed by Batch-Normalization and MaxPooling2D to further refine the feature extraction.

In the third Conv2D layer, we applied 256 filters with the same parameters to extract even more information and reduce dimensionality.

In the final Conv2D layer, we applied 512 filters to capture complex patterns and further deepen the feature extraction process.

The Flatten layer converts the 3D output from the last pooling layer into a 1D vector, which is then passed to the Dense (fully connected) layer.

In the Dense layer, we used 512 units to extract the most important features and combined this with the ReLU activation function to contribute to the final decision. We then used 256 units in the next Dense layer to ensure the model runs efficiently on average devices.

To prevent overfitting, we applied a dropout rate of 0.5, meaning half of the units are randomly dropped during training.

Finally, in the output Dense layer, we used the parameter num_class to ensure we have 8 units (neurons) for the final prediction. The softmax activation function ensures that the sum of the 8 neurons equals 1, providing a valid probability distribution for the predictions.

#### 5.4.7.3 Accuracy curve while training and testing

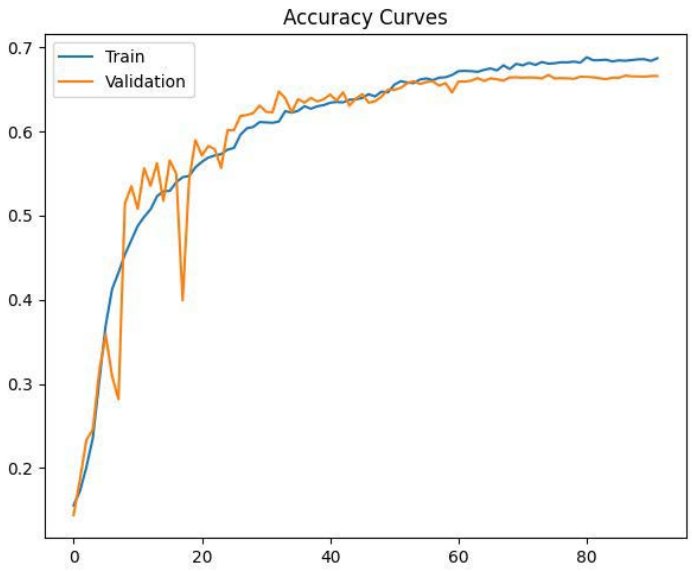



Figure 67: CNN accuracy curve (CNN/ fer2013)

This figure provides valuable insights. We observe that the model's training accuracy steadily improved, stabilizing around 0.67 after approximately 80 epochs. The testing accuracy followed a similar trend, stabilizing around 0.70.

We can conclude that the model is learning effectively and generalizing well, as evidenced by its decent accuracy. Furthermore, the small gap between the training and testing accuracies suggests that the model is not overfitting."

#### 5.4.7.4 Loss curve while training and testing

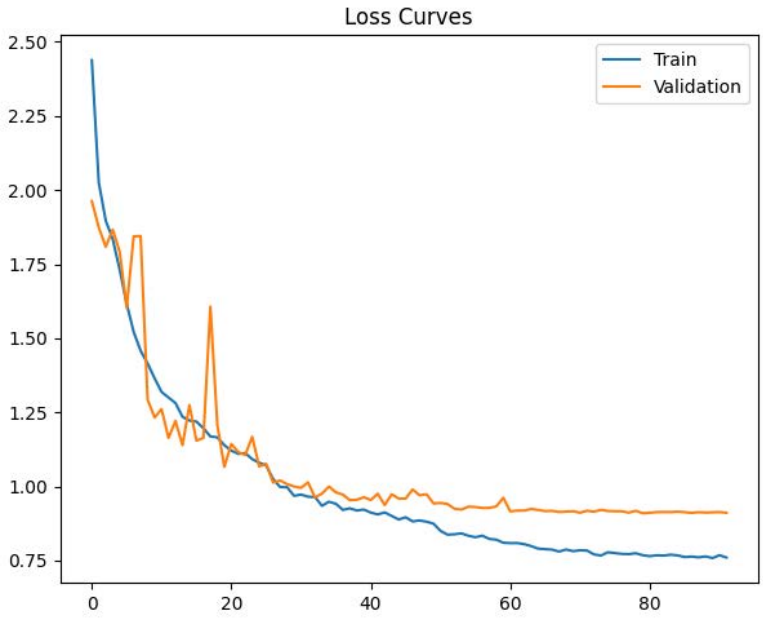



Figure 68: CNN loss curve (CNN/ fer2013)

The training loss decreases sharply during the initial epochs. After the first few epochs, the rate of decrease slows, but it continues to reduce steadily, stabilizing just below 0.75. The testing loss (validation loss) also stabilizes, just below 1.

Similarly, the test loss exhibits a gradual decline.

The small gap between the training and testing loss suggests that the model is generalizing well.

#### 5.4.7.5 Classification Report for Test and Train Data

All the emotions (precision, recall, score f1, accuracy) while training:

[RESULT] Classification Report (Train):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| angry | 0.66 | 0.69 | 0.68 | 3995 |
| disgust | 0.69 | 1.00 | 0.82 | 436 |
| fear | 0.70 | 0.48 | 0.57 | 4097 |
| happy | 0.93 | 0.87 | 0.90 | 7215 |
| neutral | 0.60 | 0.82 | 0.70 | 4965 |
| sad | 0.67 | 0.57 | 0.62 | 4830 |
| surprise | 0.81 | 0.87 | 0.84 | 3171 |
| accuracy | | | 0.73 | 28709 |
| macro avg | 0.72 | 0.76 | 0.73 | 28709 |
| weighted avg | 0.74 | 0.73 | 0.73 | 28709 |

Figure 69: Classification report train (CNN / fer2013)

All the emotions (precision, recall, score f1, accuracy) while testing:

[RESULT] Classification Report (Test):

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| angry | 0.58 | 0.63 | 0.61 | 958 |
| disgust | 0.48 | 0.78 | 0.60 | 111 |
| fear | 0.57 | 0.38 | 0.46 | 1024 |
| happy | 0.91 | 0.84 | 0.87 | 1774 |
| neutral | 0.54 | 0.75 | 0.63 | 1233 |
| sad | 0.59 | 0.49 | 0.54 | 1247 |
| surprise | 0.76 | 0.82 | 0.79 | 831 |
| accuracy | | | 0.67 | 7178 |
| macro avg | 0.63 | 0.67 | 0.64 | 7178 |
| weighted avg | 0.68 | 0.67 | 0.66 | 7178 |

Figure 70: Classification report train (CNN / fer2013)

#### 5.4.7.6 Confusion Matrix

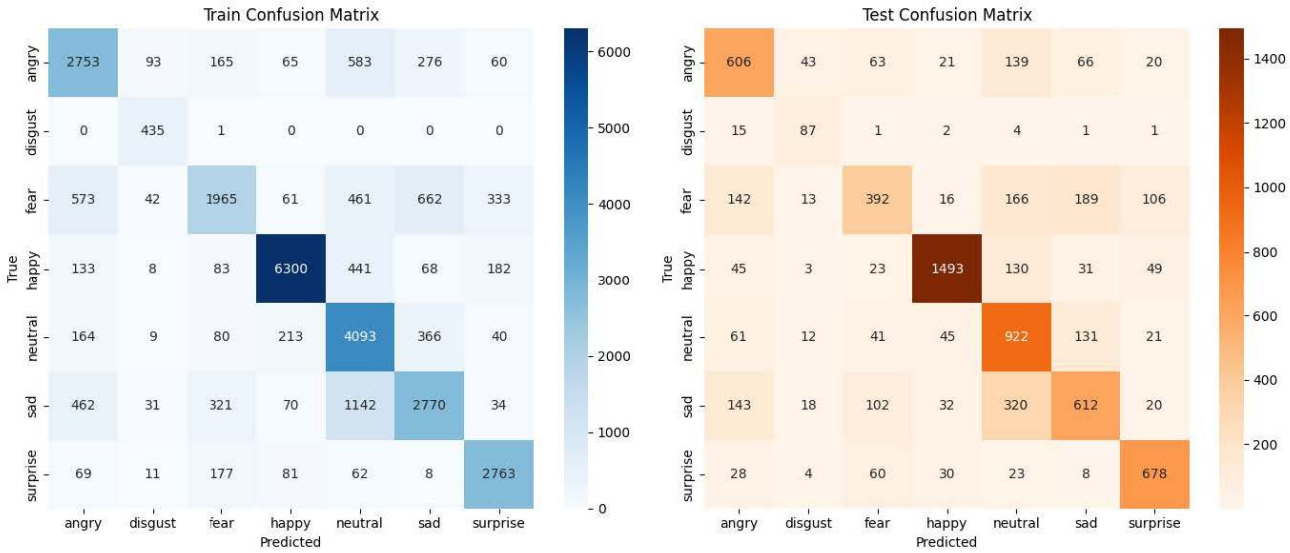



Figure 71: Confusion Matrix (CNN / fer2013)

Interpretation :

- Each row represents the true emotion.
- Each column represents the predicted emotion.
- Strong values on the diagonal = good classification.
- Off-diagonal values = misclassifications.

  → Sadness ↔ Neutral

  → Anger ↔ Disgust

  → Neutral ↔ Anger

**Observation for cnn:**

We can observe from the matrix that the model achieved strong diagonals, indicating how many emotions were predicted correctly in the test. We can also see that the model mostly struggled with the 'disgust' emotion, likely due to its similarity with the 'anger' emotion. However, overall, we can conclude that the model performs decently.

#### 5.4.7.7 Performance Metric

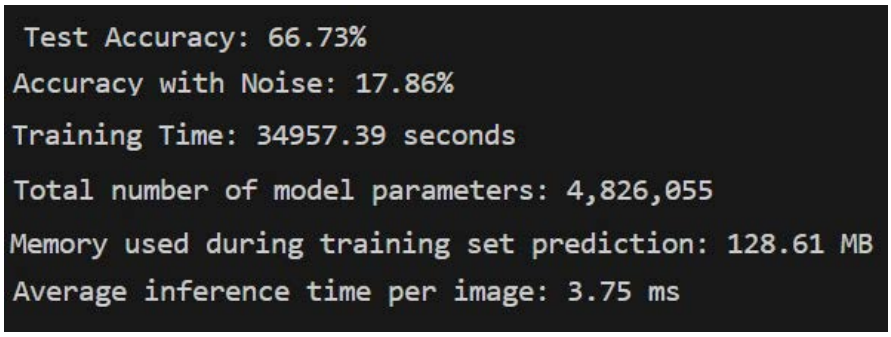



Figure 72: Performance metrics (CNN / fer2013)

The training time for CNN is always important because it goes through many layers. Keep in mind that we are working with the FER2013 dataset, which is relatively large compared to other datasets. Therefore, 9.7 minutes (349,757.39 seconds) can be considered a normal training time, given that we are working with a larger dataset and more layers.

We can see that CNN is not very accurate on noisy data, achieving only 17.86% accuracy, which can be concerning for real-time situations.

The model accuracy is 66.73%, indicating that the model is learning well and making decent predictions.

In terms of inference speed, it takes a small amount of time to process an image—just 0.375 seconds. However, this is still relatively slow, contributing to the model taking up to 9 hours for training.

Regarding memory usage, it is only 128.61 MB. By today's standards, this is not much memory usage and is relatively low compared to the SVM model.

### 5.4.8 Fer2013 dataset summary

| Criteria | Advantages of CNN | Disadvantages of CNN | Advantages of KNN | Disadvantages of KNN | Advantages of SVM | Disadvantages of SVM |
|---|---|---|---|---|---|---|
| Overall accuracy | Great (66.73%) | Sensible aux perturbations | Good (62.43%) | Less accurate than CNN and more accurate than SVM | Good (61.39%) | Less accurate than the other models |
| Noise robustness | Faible (17.86%) | Sensitive to variations | Not bad (57.45%) | No control over generalization | Good (61.27%) | Doing a great job than the others |
| Training time | 34957.39 s | Verry slow | 513,83 s | Slow | 696,55s | Slow |
| Inference time | Very Slow (0.375s) | Requires GPU for optimal speed | Slow (0.010527s) | Much slower than SVM and CNN | Slow (0.0198281s) | Slower than CNN |
| Occupied memory | Strong (128.61 MB) | Consumes a lot of resources for training | Strong (103.09 MB) | Stores all data → high memory | Consumes a lot (268.49 mb) | Unstable with unbalanced data |
| Complexity | Fixed after training | High training cost (9 hours) | Simple to understand | O(29961 x 900) | Fixed and optimized | Can be difficult to interpret |
| Scalability | Very good | Requires a lot of computing | Weak | Becomes very slow with large dataset | Good | May be limited by the number of classes |

Figure 73: Fer2013 summary

**CNN, SVM, and KNN demonstrate high performance, but their strengths differ:**

- **CNN** has the highest accuracy (66.73%), making it ideal for pure classification systems.
- **KNN** is the fastest in inference (0.000564s), making it optimal for real-time applications such as emotion tracking in video.
- **SVM** is the most robust to perturbations (61.27%), making it preferable for noisy environments or degraded images.

Comparison of each model with each emotion:

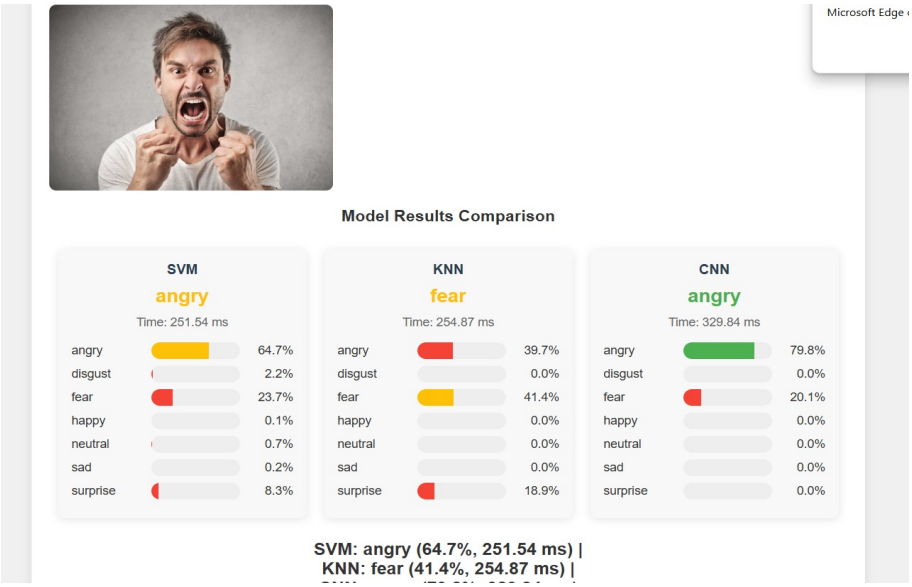



Figure 74: Anger (fer2013 dataset)

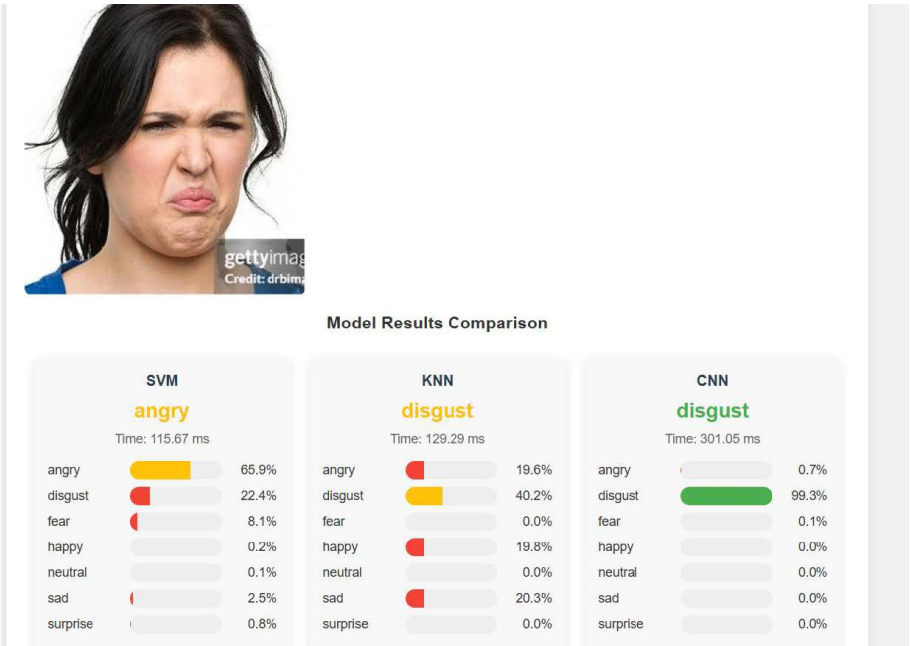



Figure 75: Disgust (fer2013 dataset)

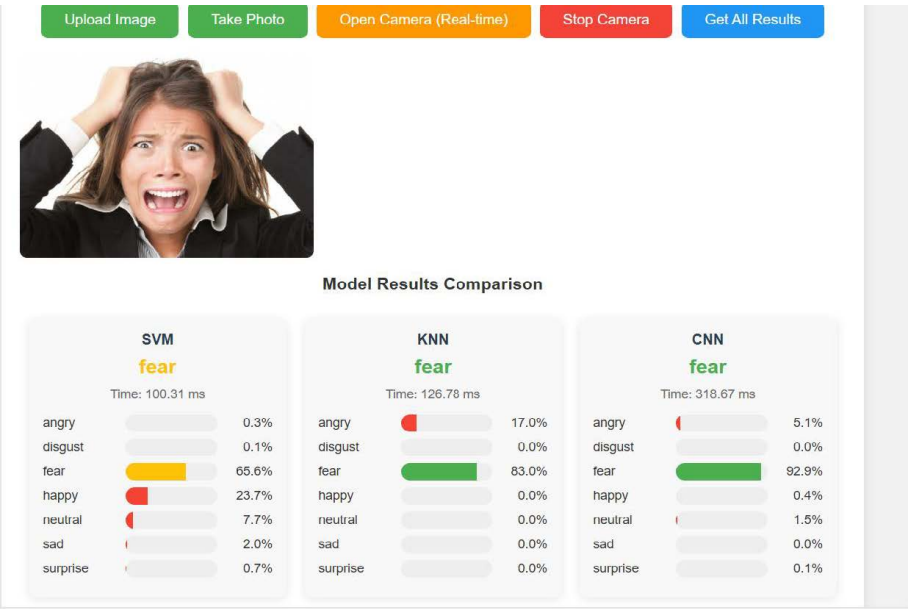



Figure 76: Fear (fer2013 dataset)

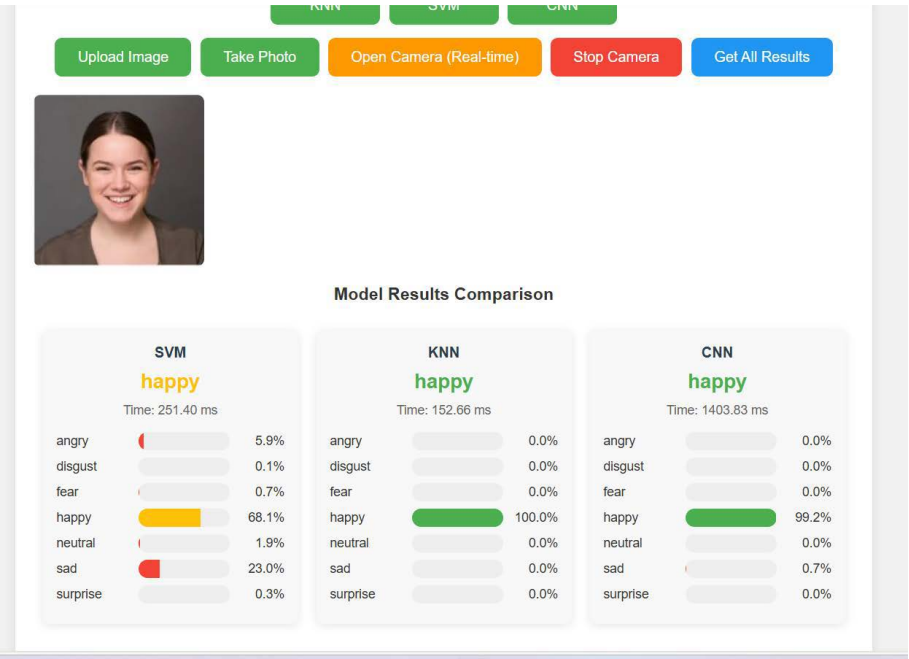



Figure 77: Happiness (fer2013 dataset)

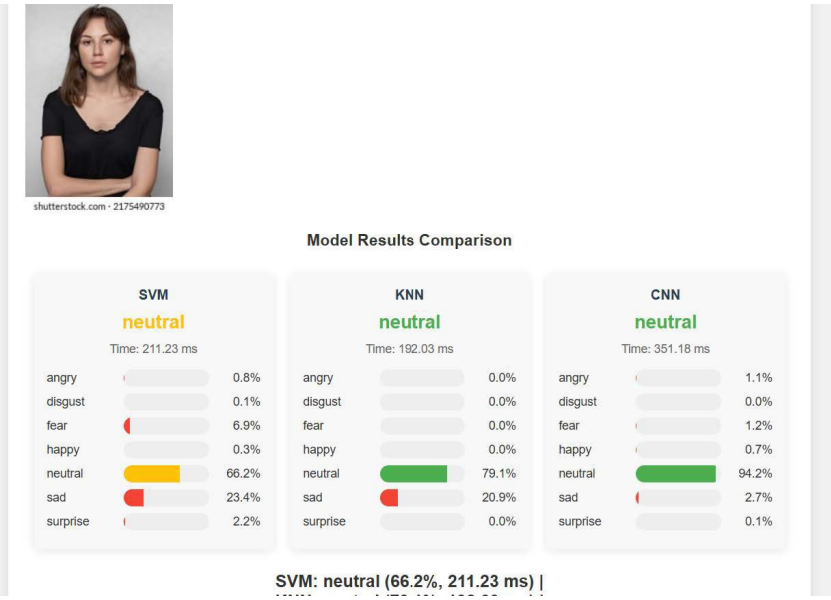



Figure 78: Neutral fer2013 dataset)

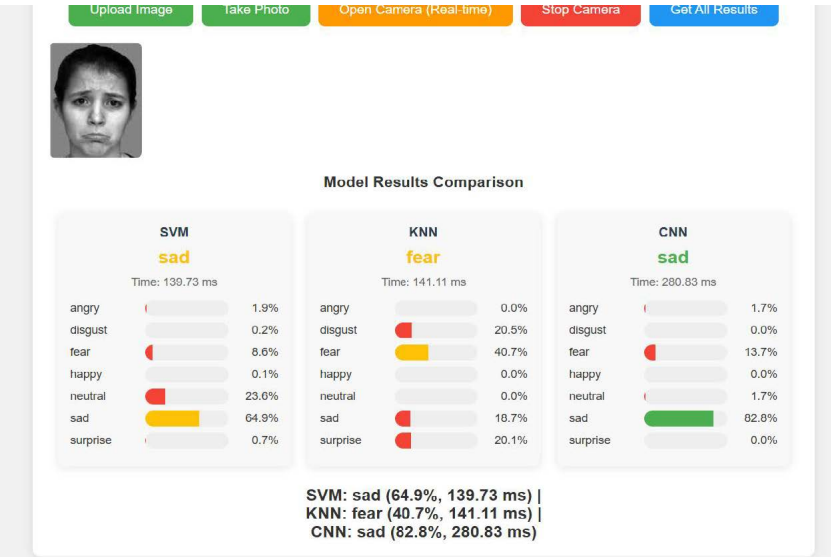



Figure 79: Sadness (fer2013 dataset)

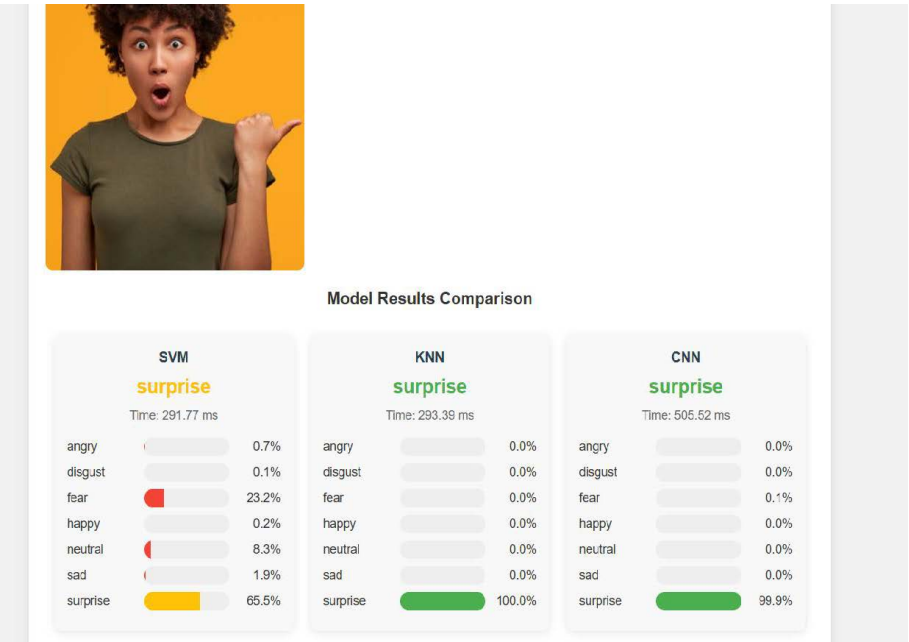



Figure 80: Surprise (fer2013 dataset)

**Conclusion**

SVM and KNN perform well in prediction time, while CNN, although slower, might offer better labeling, making it the right choice for real-time applications due to its deep learning capabilities.

SVM and KNN are traditional machine learning algorithms, meaning they classify based on the data. With larger datasets, we can observe that they struggle more than the CNN model.

## 5.5 Conclusion on datasets

We can observe that each dataset has its own advantages, and each may be better than the other in certain cases. For example, the FER2013 dataset might be more representative of real-life emotions, with more human-like expressions. In contrast, the CK+ dataset presents emotions in a more raw and absolute form, with faces shown directly and upfront. The CK+ dataset may be better suited for machine learning algorithms, while FER2013 could be more appropriate for deep learning algorithms.

# 6 Building Flask web application

## 6.1 Introduction

Flask is a lightweight and flexible framework for web development and web applications that utilizes the Python programming language. With a minimalistic design and feature-rich offerings, Flask is a go-to choice for development teams for creating dynamic websites, APIs, and microservices.

With that being said, we used Flask to create a user-friendly interface that allows users to interact with it seamlessly. Users can see which model performs the best, whether they upload an image, take a photo, or try it in real time. Each model displays its accuracy and the time it took to make a prediction.

## 6.2 Building the Flask App

We loaded the model files that were previously stored in their specified directories, along with the emotion labels.

```
# Loading the models
svm_model = joblib.load('svm_model.pkl')
knn_model = joblib.load('knn_emotion_model.pkl')
scaler = joblib.load('svm_scaler.pkl')
cnn_model = load_model('emotion_cnn_model.h5')

# Emotion labes
emotion_labels = {
    0: "Anger", 1: "Contempt", 2: "Disgust", 3: "Fear",
    4: "Happiness", 5: "Neutral", 6: "Sadness", 7: "Surprise"
}
```

Figure 81: Loading models

In the screenshot below, we will take a look at the website and how it looks in each use case

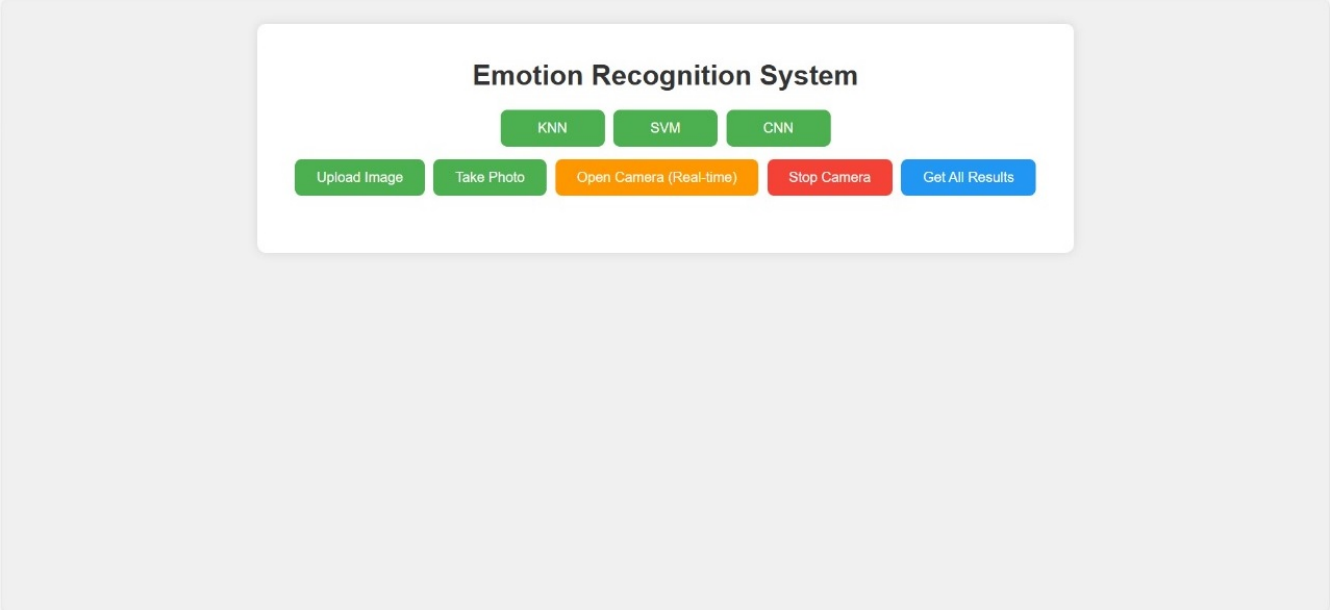



Figure 82: Web Flask app interface

In the screenshot above there is the interface of the website with three different model button each one goes for it’s own subset.

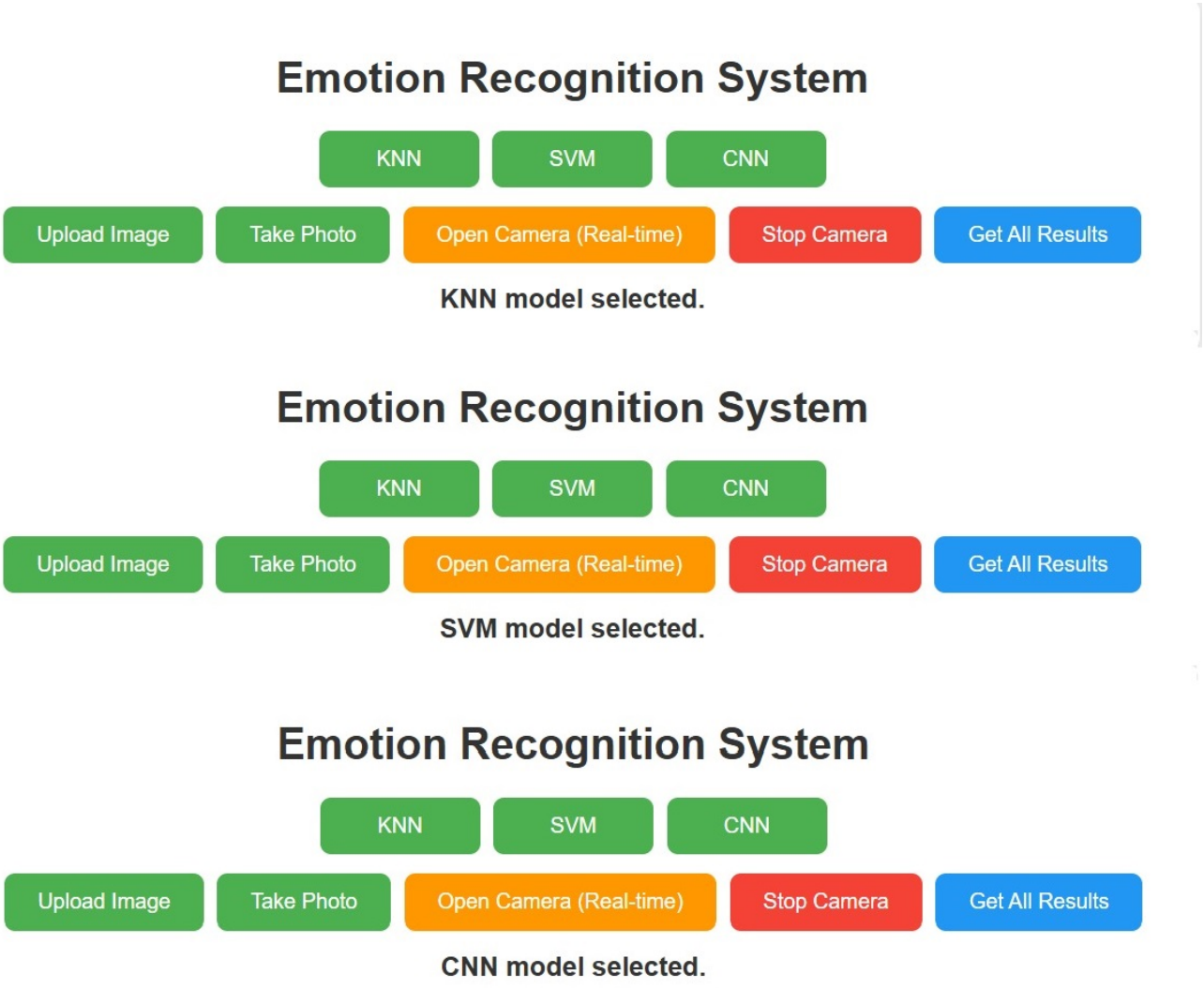



Figure 83: Models buttons

After each click on the buttons, it selects the specified model to display the results belonging to that model, along with informative text beneath.

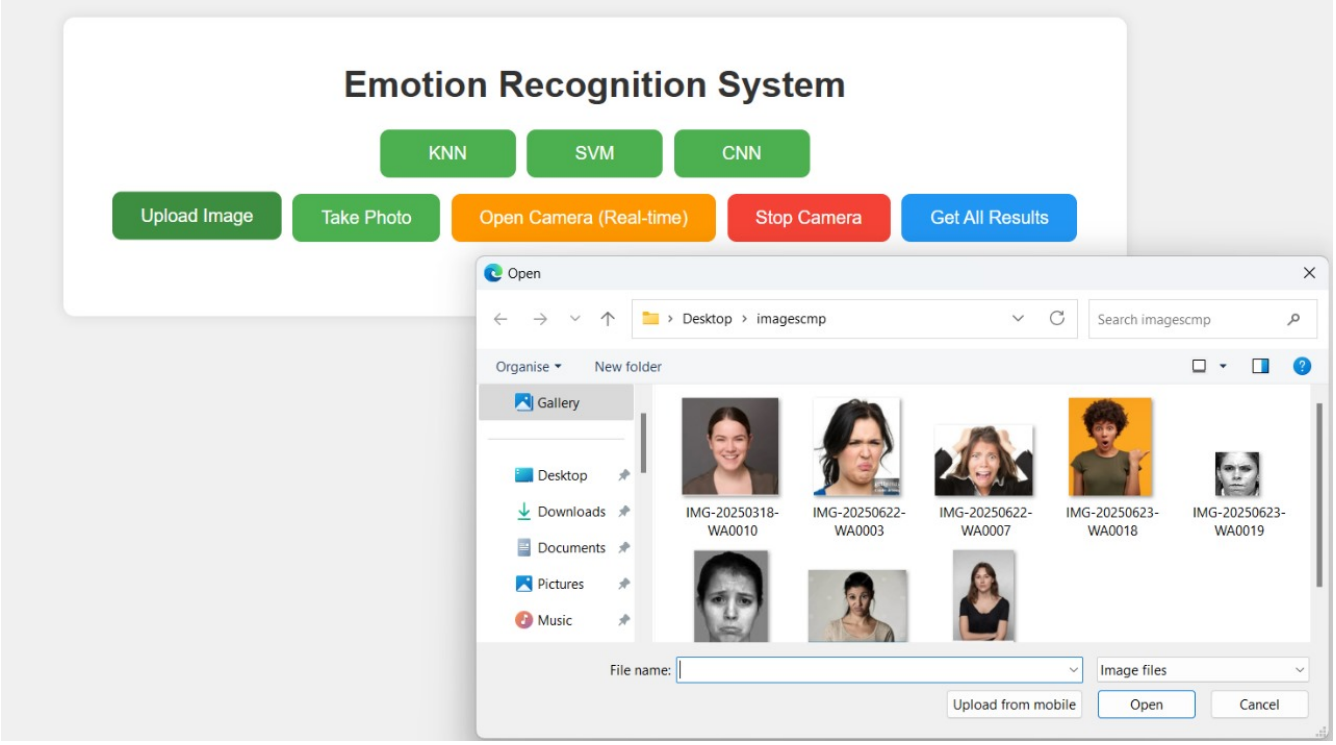



Figure 84: Upload image button

After clicking on the ’Upload Image’ button, a small upload interface window pops up, allowing us to upload all types of image files.

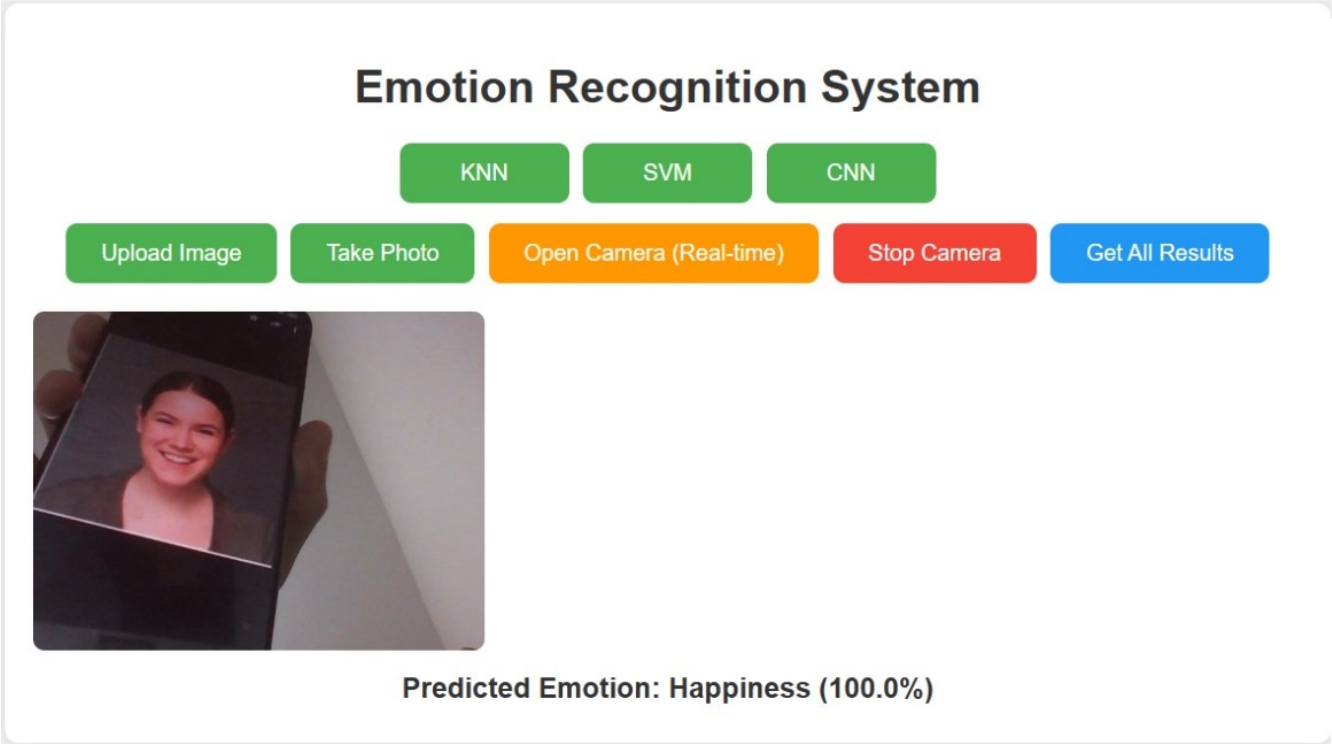



Figure 85: Take Photo button

The ’Take Photo’ button allows us to take a photo in real-time without any issues, and labels it with the percentage predicted by the model.

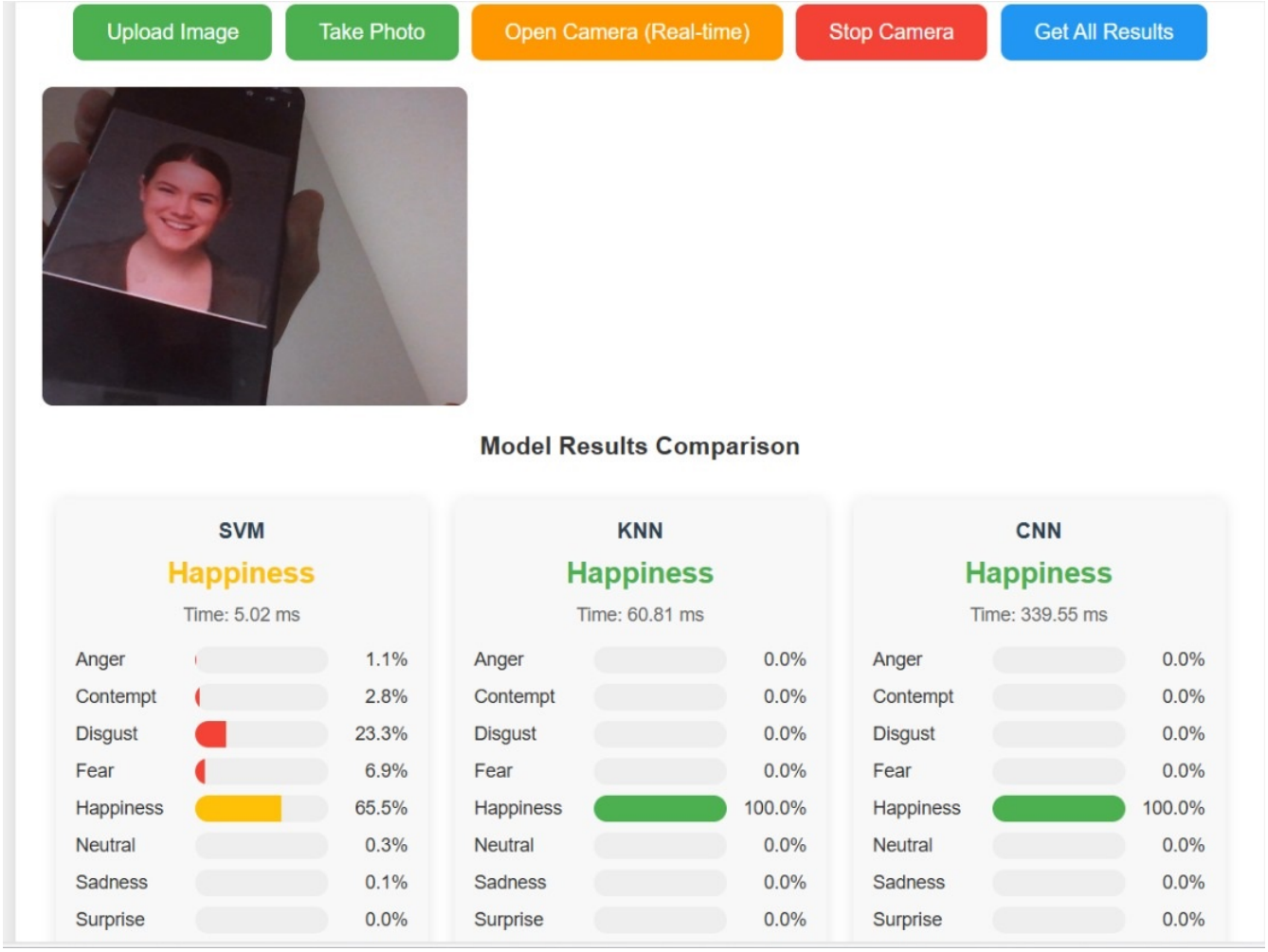



Figure 86: Get all result button

The 'Get All Results' button allows us to see the predictions of all the models for each emotion, along with the percentage associated with each emotion in the corresponding model.

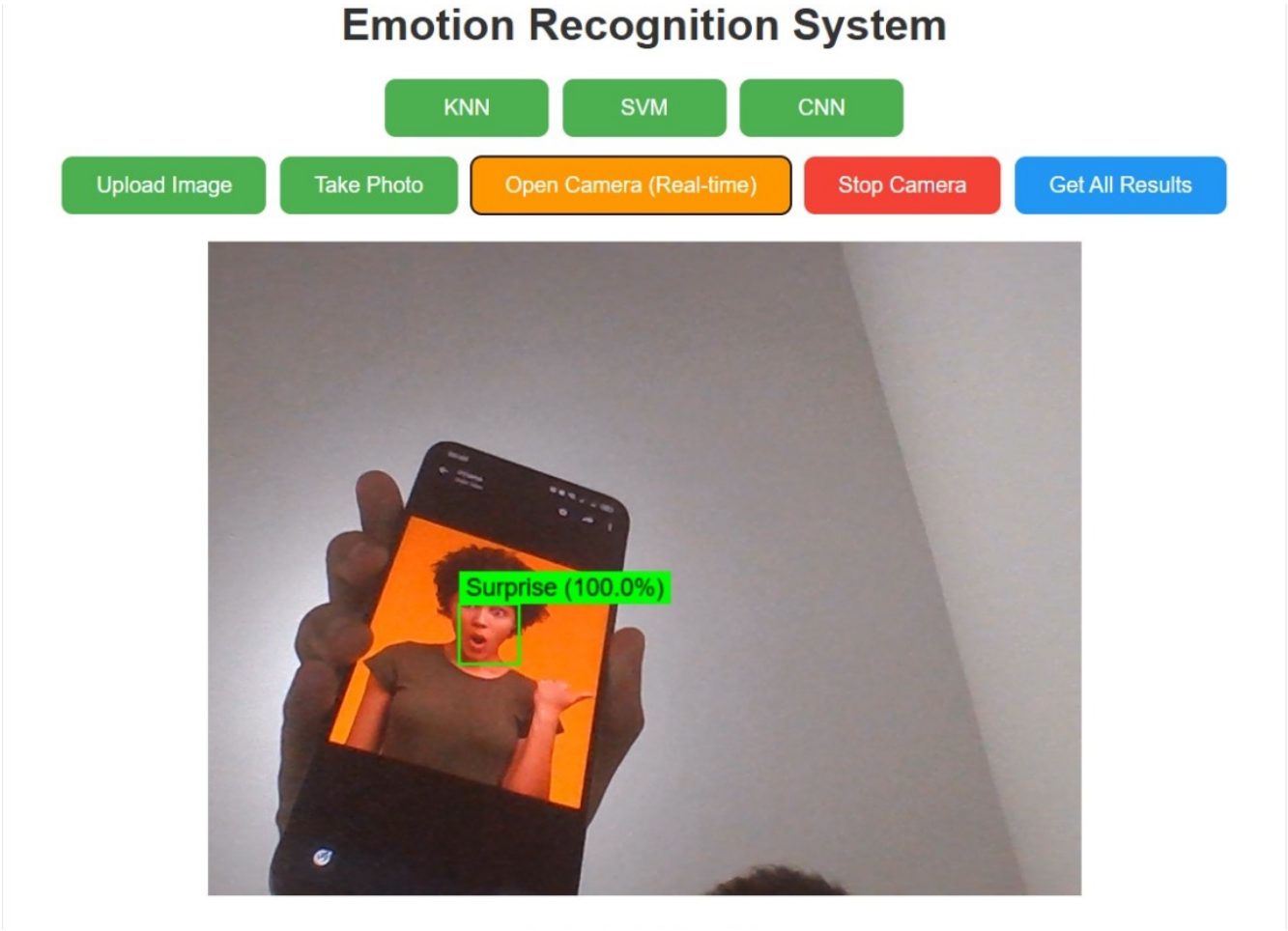



Figure 87: Open Camera (real-time) button

The 'Open Camera' button allows us to use the model in real-time, which can also be applied in real-world situations, making predictions alongside their corresponding percentages.

## 6.3 Conclusion

In this chapter, we've seen how we uploaded the models using the Joblib library and developed the Flask web app. The app was created with HTML, CSS, JavaScript, and Python, with functions similar to those we used while training the models to extract faces, enabling us to obtain the results. We ensured that the interface is user-friendly, making it simple to use without any complications, in order to achieve our project goal.

# General conclusion and perspectives

Our end-of-year project focuses on facial emotion detection, which has potential applications in many fields using machine learning and deep learning. We followed several key steps while developing this project. First, we collected and prepared the training data, applying known techniques such as balancing the data and converting CSV files to images for our model. We then integrated the results into a Flask web application developed in Python, HTML, and CSS—languages commonly used for Flask web applications.

This project allowed us to explore the fields of artificial intelligence, machine learning, and deep learning. We utilized technologies such as Python, along with platforms like Google Colab, Visual Studio Code, and Overleaf, which enriched our knowledge and developed our skills.

Our Flask web application for facial expression detection offers an intuitive interface to compare the performance of different models for expression detection, using TensorFlow with the help of other libraries. This framework is lightweight and efficient for training the model, which is then stored using Joblib. This allows the app to function without an internet connection.

In addition to the achievements mentioned above, our project still has room for improvement. While the CK+ dataset yielded excellent results, with accuracy nearing 98

- **Feature Extraction:** We can try different algorithms for feature extraction, such as Local Binary Patterns (LBP) or Gabor filters, and evaluate their performance with machine learning algorithms.
- **Hyperparameter Tuning:** We can use Grid Search or Randomized Search to optimize parameters like C, gamma, and kernel settings.
- **Use SVM with Deep Features:** We can extract features from intermediate CNN layers and feed them into an SVM classifier.
- **Use Pretrained Models:** Fine-tuning models like VGG, ResNet, or MobileNet through transfer learning on the FER2013 dataset could enhance performance.
- **Optimize Model for Real-Time:** To improve the efficiency of live predictions, we can reduce model complexity through techniques like pruning, quantization, or using lightweight architectures.

# References


Css. https://en.wikipedia.org/wiki/CSS, note = Accessed: 2025-06-26.

Ck+ dataset. https://www.kaggle.com/datasets/davilsena/ckdataset. Accessed: 2025-06-26.

Fer2013 dataset. https://www.kaggle.com/datasets/msambare/fer2013. Accessed: 2025-06-26.

Html. 15.https://en.wikipedia.org/wiki/HTMLhtml. Accessed: 2025-06-26.

Javascritp. https://en.wikipedia.org/wiki/JavaScript, note = Accessed: 2025-06-26.

K-nearest neighbors (knn). https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/. Accessed: 2025-06-26.

Numpy. https://numpy.org/, note = Accessed: 2025-06-26.

Opencv. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html, note = Accessed: 2025-06-26.

Python. https://en.wikipedia.org/wiki/Python_(programming_language). Accessed: 2025-06-26.

Support vector machine (svm). https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/. Accessed: 2025-06-26.

Tensorflow. https://www.tensorflow.org/, note = Accessed: 2025-06-26.

Visual studio code. https://en.wikipedia.org/wiki/Visual_Studio_Code, note = Accessed: 2025-06-26.

cascade. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html, note = Accessed: 2025-06-26.

dlib. https://ieeexplore.ieee.org/abstract/document/9523885, note = Accessed: 2025-06-26, a.

dlib$_2$., note = Accessed: 2025-06-26, b.

joblib. https://eudl.eu/pdf/10.4108/eai.18-12-2023.2348107, note = Accessed: 2025-06-26.

keras. https://www.researchgate.netpublication/368073383_Understanding_and_Working_with_Keras, note = Accessed: 2025-06-26.

matplotlib. https://www.jabawok.net/gentoo/distfiles/users_guide_0.91.2svn.pdf, note = Accessed: 2025-06-26.

scikit-learn. http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?source=post_page, note = Accessed: 2025-06-26.

sckit-image. https://peerj.com/articles/453/, note = Accessed: 2025-06-26.

seaborn. https://seaborn.pydata.org/tutorial/introduction.html, note = Accessed: 2025-06-26.