**Machine Learning (UML501)**

**Project Report**

**Mood Recognition and Recommendation System**

Submitted by:

102103319 - Ananya Chawla

### 102103329 - Akshat Bhatnagar

**BE Third Year, COE**

Submitted to: Dr. Jyoti Maggu

A logo for a university

Description automatically generated

**Computer Science and Engineering**

**Department, TIET, Patiala**

**November 2023**

## Project Overview

In the era of personalization, understanding a user’s mood is crucial for providing tailored content that enhances the user experience. This project aims to bridge this gap by developing a machine learning model that can accurately recognize a person’s mood based on their facial expressions and recommend videos or songs that align with their current emotional state.

**Project Description**

The **Mood Recognition and Recommendation System** works by recognizing a user’s mood through their facial expressions. It uses a trained machine learning model to classify the user’s emotion from an image into one of several categories such as happiness, sadness, anger, surprise, etc. This mood classification is then used to recommend content - like songs or videos - that aligns with the user’s current emotional state.

This approach adds a layer of personalization to content recommendation systems. Instead of merely suggesting content based on past behaviour or popular trends, the system can adapt to the user’s current emotional state, providing a more tailored and responsive user experience. This could be particularly useful in entertainment platforms like music streaming or video streaming services, where mood-based recommendations could significantly enhance user engagement and satisfaction.

**Architecture**

A diagram of a face detection process

Description automatically generated

**Data Set Description**

**Dataset used for Recognition of mood:**

The data comes from the past Kaggle challenge "Challenges in Representation Learning: Facial Expression Recognition Challenge":

<https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge>

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centred and occupies about the same amount of space in each image. Each image corresponds to a facial expression in one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The dataset contains approximately 36K images.

The original data consisted in arrays with a greyscale value for each pixel. We converted this data into raw images and split them in multiple folders:

images/  
    train/  
        angry/  
        disgust/  
        fear/  
        happy/  
        neutral/  
        sad/  
        surprise/  
    train/  
        angry/  
        disgust/  
        fear/  
        happy/  
        neutral/  
        sad/  
        surprise/

80% of our images are contained inside the train folder, and the last 20% are inside the validation folder.

**Dataset for recommending the videos :**

It is a collection of video and song links segregated according to the mood requirement

## Methodology

**The methodology of mood recognition model involves the following steps:**

1. **Data Preprocessing**: The images are rescaled and normalized. The data is loaded in batches using the flow\_from\_directory method.
2. **Model Building**: A Convolutional Neural Network (CNN) model is built using Keras. The model consists of Conv2D, MaxPooling2D, Dropout, Flatten, and Dense layers.
3. **Model Compilation**: The model is compiled with the Adam optimizer and the categorical cross-entropy loss function.
4. **Model Training**: The model is trained using the fit\_generator method, which allows for real-time data augmentation.

This methodology allows the model to learn from the training data and accurately classify a person’s mood based on their facial expressions.

**For recommending the videos:**

1. **Data Loading**: The data is loaded from a CSV file using the pandas library’s read\_csv function. The CSV file contains responses from a form, which includes the time of response, mood, link to a video or song, and the type of content (video or song).
2. **Data Cleaning**: The ‘Time’ column is dropped from the dataframe as it is not necessary for the recommendation system.
3. **Data Encoding**: The ‘Mood’ and ‘Type’ columns are label encoded using the LabelEncoder class from the sklearn.preprocessing module. This converts the categorical values in these columns into numerical values which can be used for computation.
4. **Data Exploration**: The correlation between different columns is calculated and a scatter plot is created to visualize the relationship between ‘Mood’ and ‘Type’.
5. **Recommendation Function**: A function fun is defined which takes a mood as input and returns a random link from the dataframe that corresponds to that mood. This function is applied to all unique moods to get a recommendation for each mood.
6. **Data Export**: Finally, the processed dataframe is saved to a new CSV file.

The primary algorithm used in this code is label encoding, which is a preprocessing step that converts categorical data into numerical data.

# Preprocessing

In this project, data preprocessing is an important step that involves preparing the raw dataset for feeding into the machine learning model. The specific preprocessing steps used in this project are:

1. **Image Rescaling**: The images are rescaled by a factor of 1/255. This is done to normalize the pixel values to be in the range of 0-1. This helps the model to converge faster during the training process and reduces the chance of getting stuck in local optima.
2. **Data Augmentation**: Although not explicitly mentioned in the code, data augmentation can be performed using the ImageDataGenerator class. Data augmentation involves creating new training samples by applying random jitters and transformations to the images in the dataset. This can help improve the model’s performance by providing more varied and generalized training data.
3. **Flow From Directory**: The flow\_from\_directory method is used to load images from the dataset directory. This method generates batches of tensor image data from the specified directory. It’s a memory-efficient way of handling large datasets, as it loads images into memory as needed, rather than all at once. The method also automatically labels the images based on the directory structure.
4. **Grayscale Conversion**: The color\_mode argument is set to “grayscale” in the flow\_from\_directory method. This converts the images to grayscale, reducing the complexity of the model as it only needs to process one color channel.
5. **Categorical Labels**: The class\_mode argument is set to “categorical” in the flow\_from\_directory method. This converts the class labels to one-hot encoded format which is suitable for multi-class classification problems.

These preprocessing steps ensure that the data is in the right format and ready to be used by the machine learning model.

**Algorithm Used**

# A Convolutional Neural Network (CNN) is a type of deep learning model that is primarily used for image processing tasks. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from the input images.

# 

The primary algorithm used in our model is a Convolutional Neural Network (CNN), a class of deep learning models that are particularly effective for image classification tasks. Here’s a more detailed explanation of the algorithm:

1. **Conv2D Layers**:

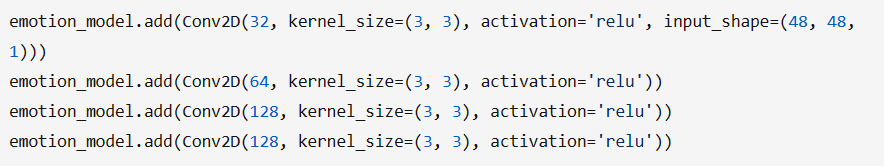
Conv2D layers are used as the first part of the our model. These layers perform convolution operations which are fundamental to Convolutional Neural Networks (CNNs).

The Conv2D layers take a 2D image as input and apply convolution operations on it using filters (also known as kernels). Each filter is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, each filter is slid across the width and height of the input volume and dot products are computed between the entries of the filter and the input at any position.

In our model, the first Conv2D layer has 32 filters, each of size 3x3, and uses the ‘relu’ activation function. The second Conv2D layer has 64 filters, each of size 3x3, and also uses the ‘relu’ activation function. The ‘relu’ activation function introduces non-linearity into the model, allowing it to learn more complex patterns.

These convolution operations allow the model to learn local features in the early layers and more global features in the deeper layers. In the context of our project, these could be features that represent different facial expressions associated with different moods. For example, a frown might be associated with a sad mood, while a smile might be associated with a happy mood. The Conv2D layers help the model learn to recognize these features and use them to classify the mood of a person based on their facial expression.

Here are the lines of code where Conv2D layers are used:



Each line adds a Conv2D layer to the model. The first parameter is the number of filters that the convolutional layer will learn. kernel\_size specifies the height and width of the 2D convolution window. The activation parameter is set to ‘relu’, which stands for Rectified Linear Unit, an activation function that introduces non-linearity into the model. The input\_shape parameter in the first Conv2D layer specifies the shape of the input images.

1. **MaxPooling2D Layers**:

MaxPooling2D layers are used in the model to reduce the spatial dimensions (width and height) of the output volume from the Conv2D layers. They help to decrease the computational complexity of the model, and also help to extract dominant features from the previous layer. This is particularly useful in a task like mood recognition from images, where the exact location of features isn’t as important as their presence.

In our model, these layers use a pool size of 2x2, which reduce the width and height of the output volume by half. This operation is performed separately on each depth slice of the input.

Here are the lines of code where MaxPooling2D layers are used:

A screenshot of a computer program

Description automatically generated

Each line adds a MaxPooling2D layer to the model after a Conv2D layer or a set of Conv2D layers. The pool\_size=(2, 2) parameter specifies that the maximum value over a 2x2 pooling window is used.

A diagram of a diagram of a network

Description automatically generated with medium confidence

1. **Dropout Layers**:

Dropout layers are used in the model as a regularization technique to prevent overfitting. During training, dropout layers randomly set a fraction of input units to 0 at each update, which helps to prevent overfitting. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

In our model, these layers drop out 25% of the neurons, which means each neuron has a 25% chance of being temporarily “dropped out” or deactivated during each training update. This helps to make the model more robust and less likely to rely heavily on any one feature, thereby improving generalization.

Here are the lines of code where Dropout layers are used:

A close up of text

Description automatically generated

Each line adds a Dropout layer to the model after a set of Conv2D and MaxPooling2D layers. The parameters 0.25 and 0.5 specify the fraction of the input units to drop.

1. **Flatten Layer**:

The Flatten layer is used in the model to convert the 2D matrix output of the previous layer into a 1D vector. This is necessary because after the convolution and pooling layers, the data is in the form of a 2D matrix. However, the fully connected Dense layers expect input in the form of a 1D array.

The Flatten layer essentially takes the 2D matrix of features created by the previous layers and ‘flattens’ it into a 1D array. This allows the output from the convolutional part of the CNN to be used in the Dense layers for the final classification.

Here is the line of code where the Flatten layer is used:



This line adds a Flatten layer to the model after the convolutional and pooling layers and before the Dense layers.

1. **Dense Layers**:

Dense layers, also known as fully connected layers, are used in the model as the final layers, which perform classification on the features extracted by the previous layers. Each neuron in a Dense layer receives input from all the neurons of the previous layer, hence they are ‘fully connected’.

In our model, the first Dense layer has 1024 neurons and uses the ‘relu’ activation function. The ‘relu’ activation function introduces non-linearity into the model, allowing it to learn more complex patterns.

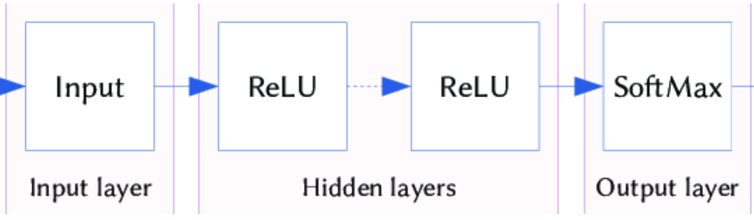
The final Dense layer uses the softmax activation function to output a probability distribution over the seven emotion classes. The softmax function outputs a vector that represents the probability distributions of a list of potential outcomes. It’s a way of normalizing the output of the network to represent probabilities.

Here are the lines of code where Dense layers are used:

A close-up of numbers

Description automatically generated

The first line adds a Dense layer with 1024 neurons and ‘relu’ activation function. The second line adds a Dense layer with 7 neurons (corresponding to the seven emotion classes) and a ‘softmax’ activation function. These Dense layers enable the model to classify the input image into one of the seven emotion classes based on the features extracted by the Conv2D and MaxPooling2D layers.



Layers of ReLu and softmax

A graph with a line

Description automatically generated A graph of a function

Description automatically generated

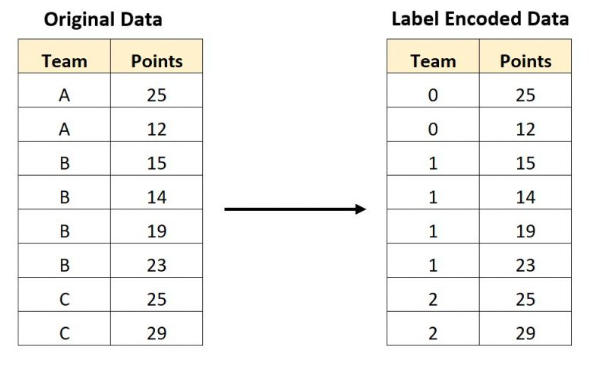
ReLu function Softmax Function

The model is compiled with the Adam optimizer, an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks. The categorical cross-entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one.

The model is trained using the fit\_generator method of the Sequential model. This method is used when the dataset is too large to fit into memory. It generates batches of image data with real-time data augmentation. The combination of these methodologies and algorithms allows the model to effectively learn from the training data and accurately classify the mood of a person based on their facial expressions.

**Algorithm used for recommendation model :**

The primary algorithm used in code for recommending the videos to user is label encoding, which is a preprocessing step that converts categorical data into numerical data.



Here are the lines of code where LabelEncoder is used:

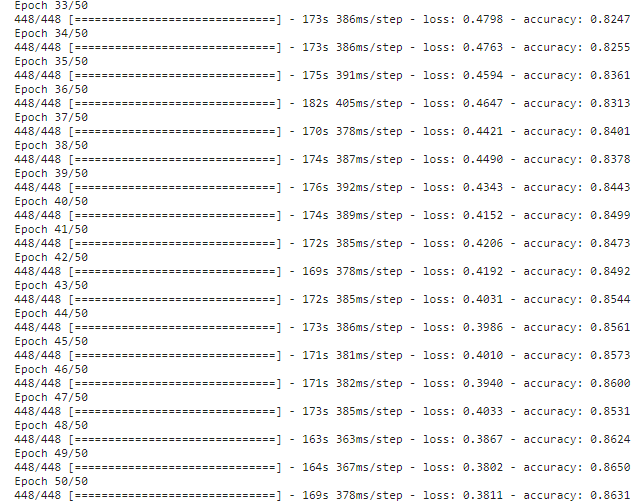
# A screenshot of a computer code Description automatically generated

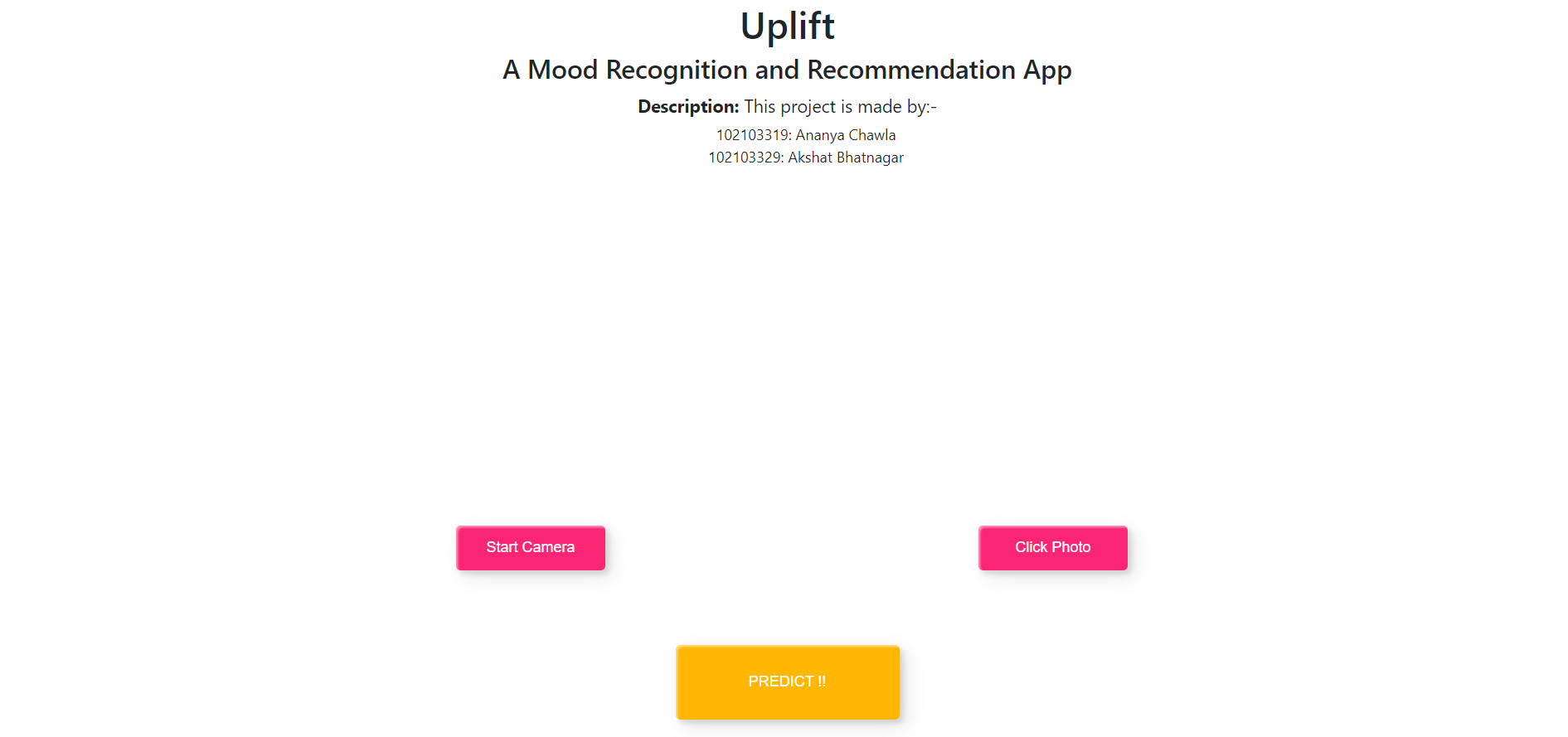
In these lines, two LabelEncoders are created: encoder\_mood and encoder\_type. The fit\_transform method is used to fit the encoder to the data and then transform the data. The transformed data, which is now numerical, replaces the original categorical data in the ‘Mood’ and ‘Type’ columns of the dataframe.

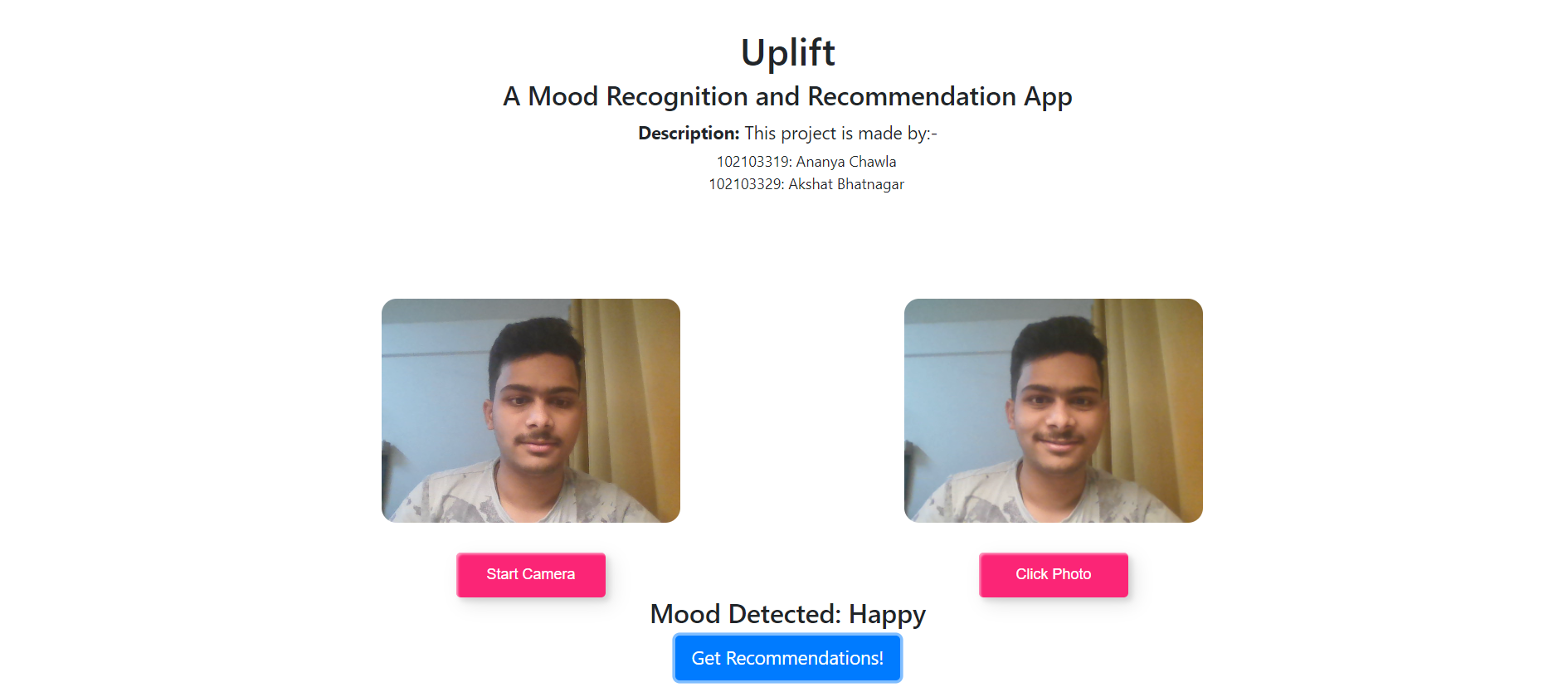
The encoder\_mood is fitted and used to transform the ‘Mood’ column, which contains the different moods. The encoder\_type is fitted and used to transform the ‘Type’ column, which contains the types of content (video or song).

**Result**

Our model could predict a person’s mood with an accuracy of 86%







A screenshot of a person

Description automatically generated