# Facial Emotion Expressions

A facial recognition system is a technology capable of matching a [human face](https://en.wikipedia.org/wiki/Human_face) from a [digital image](https://en.wikipedia.org/wiki/Digital_image) or a [video frame](https://en.wikipedia.org/wiki/Film_frame) against a [database](https://en.wikipedia.org/wiki/Database) of faces. Such a system is typically employed to [authenticate](https://en.wikipedia.org/wiki/Authenticate) users through ID verification services, and works by pinpointing and measuring facial features from a given image[1]. Under this huge umbrella we are going to use one of its branches in this project i.e., facial emotion expressions to detect emotions of humans from a data set from Kaggle.

## **Importing the libraries required:**

**import** pandas **as** pd

**import** os

**import** glob **as** gb

**from** tensorflow **import** keras

**from** keras.models **import** Sequential

**from** keras.layers **import** Conv2D

**from** keras.layers **import** MaxPooling2D

**from** keras.layers **import** Flatten

**from** keras.layers **import** Dense [2]

pandas - This library is used for manipulation if data and analysis of data[3].

os - This module provides a way to interact with the operating system. It can be used to access files, directories, environment variables, and other system-related information[3], which we use to get our images for training and validation while building the model.

glob – This module provides a way to find all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. It is often used to search for files in a directory that match a certain pattern[3].

tensorflow – This is used here due to its high performance capabilities in handling large-scale computation and matrix multiplication operations[3].

keras - This is a high-level neural networks API that is written in Python and can run on top of TensorFlow, CNTK, or Theano. It allows for fast experimentation with deep neural networks[3].

**Storing the path of our dataset:**

TRAIN\_DIR **=** "/Users/akd/Facial Expressions/images/train/"

TEST\_DIR **=** "/Users/akd/Facial Expressions/images/validation/"

BATCH\_SIZE**=**64

**Checking the total number of items in each folder of the dataset:**

for folder in os.listdir(TRAIN\_DIR):

files = gb.glob(pathname= str(TRAIN\_DIR+ '/'+ folder + '/\*.jpg'))

print(f'For training data, found {len(files)} in folder {folder}') [2]



for folder in os.listdir(TEST\_DIR):

files = gb.glob(pathname= str(TEST\_DIR+ '/'+ folder + '/\*.jpg'))

print(f'For testing data, found {len(files)} in folder {folder}') [2]

A picture containing background pattern

Description automatically generated

import random

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

def view\_random\_image(target\_dir, target\_class):

# We will view images from here

target\_folder = target\_dir + target\_class

# Get a random image path

random\_image = random.sample(os.listdir(target\_folder), 1)

# read in the image and plot it using matplolib

img = mpimg.imread(target\_folder+'/'+random\_image[0])

plt.imshow(img)

plt.title(target\_class)

plt.axis('off')

print(f"Image shape {img.shape}")

return img [2]

The above function returns a random image for our dataset.

class\_names = ['angry','disgust','fear','happy','neutral','sad','surprise']

Creating a list with all the different expressions we have.

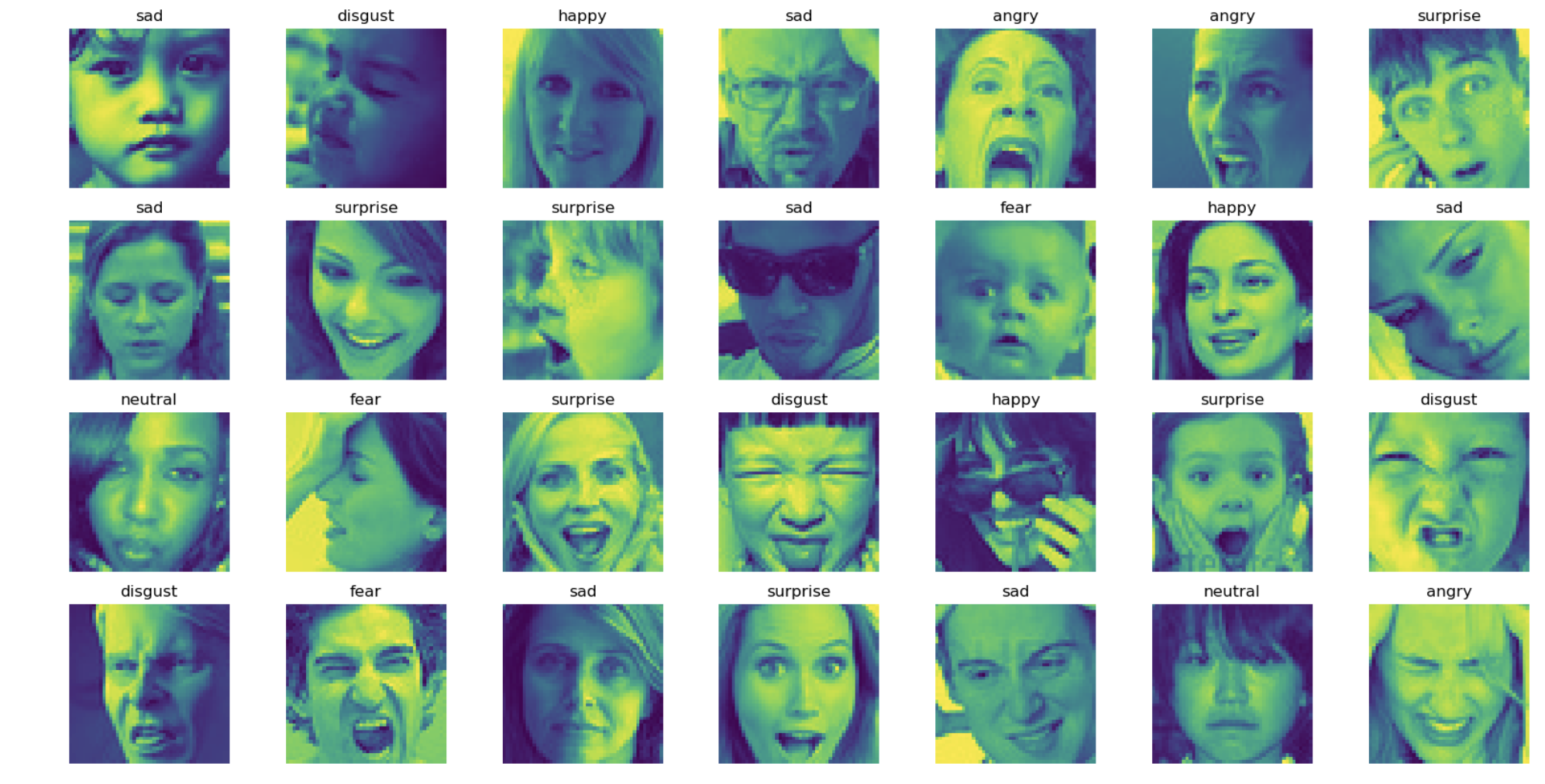
plt.figure(figsize=(20,10))

for i in range(28):

plt.subplot(4, 7, i+1)

class\_name = random.choice(class\_names)

img = view\_random\_image(target\_dir="/Users/akd/Facial Expressions/images/train/", target\_class=class\_name)



Here we are trying to generate a total of 28 images from our dataset with a fixed grid size.

from keras.preprocessing.image import ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True)

test\_datagen = ImageDataGenerator(rescale = 1./255)

training\_set = train\_datagen.flow\_from\_directory(TRAIN\_DIR,

target\_size = (128, 128),

batch\_size = BATCH\_SIZE,

class\_mode = 'categorical')

test\_set = test\_datagen.flow\_from\_directory(TEST\_DIR,

target\_size = (128, 128),

batch\_size = BATCH\_SIZE,

class\_mode = 'categorical') [2]



The above code is used to pre-process image data for a machine learning model using the ImageDataGenerator class from the keras.preprocessing.image module.

The train\_datagen object is created using ImageDataGenerator and contains various image augmentation techniques, such as rescaling, shearing, zooming, and horizontal flipping. These techniques are applied to the training set to increase the diversity of the data, thereby improving the generalization performance of the model.

The test\_datagen object is also created using ImageDataGenerator but only rescales the test set images by a factor of 1/255.

The training\_set object is created using the flow\_from\_directory() method from train\_datagen, which takes the path to the directory containing the training set images, TRAIN\_DIR, as input. It also sets the target size of the images to 128x128 pixels, the batch size to BATCH\_SIZE, and the class mode to 'categorical', which means that the labels of the images are represented as one-hot encoded vectors.

Similarly, the test\_set object is created using the flow\_from\_directory() method from test\_datagen, which takes the path to the directory containing the test set images, TEST\_DIR, as input.

These pre-processed data generators are then fed into a machine learning model to train and evaluate the model's performance on the facial expression classification task [3].

# Initialising the CNN

classifier = Sequential()

# Step 1 - Convolution

classifier.add(Conv2D(16, (3, 3), input\_shape = (128, 128, 3), activation = 'relu'))

# Step 2 - Pooling

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

# Adding a second convolutional layer

classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

classifier.add(MaxPooling2D(pool\_size = (2, 2)))

# Step 3 - Flattening

classifier.add(Flatten())

# Step 4 - Full connection

classifier.add(Dense(units = 128, activation = 'relu'))

classifier.add(Dense(units = 7, activation = 'softmax'))

# Compiling the CNN

classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy']) [2]

The above code initializes and compiles a Convolutional Neural Network (CNN) model with two convolutional layers, two pooling layers, one fully connected layer, and an output layer with softmax activation function for facial expression classification[3].

## model summary

classifier.summary() [2]

Table

Description automatically generated

Checking the summary of our model so far.

history = classifier.fit(training\_set,

epochs = 50,

validation\_data = test\_set)

classifier.save('model1.h5') # creates a HDF5 file 'my\_model.h5' [2]

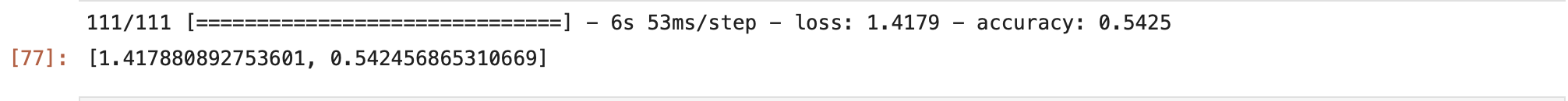
Table

Description automatically generated

The code trains the CNN model using the pre-processed training set and validation set, for 50 epochs, and saves the trained model as an HDF5 file named 'model1.h5'[3].

Our model accuracy is 73% for train data and 54% for the test data.

classifier.evaluate(test\_set) [2]



Evaluating our classifier with the test set.

pd.DataFrame(history.history)[['loss','val\_loss']].plot()

plt.title('Loss')

plt.xlabel('epochs')

plt.ylabel('Loss') [2]

Chart, line chart

Description automatically generated

Plotting a graph for loss to check our results.

pd.DataFrame(history.history)[['accuracy','val\_accuracy']].plot()

plt.title('Accuracy')

plt.xlabel('epochs')

plt.ylabel('Accuracy') [2]

Chart, line chart

Description automatically generated

Plotting a graph for accuracy to check our results.

model\_path = "model1.h5"

loaded\_model = keras.models.load\_model(model\_path)

import matplotlib.pyplot as plt

import numpy as np

import cv2

from PIL import Image

image = cv2.imread("/Users/akd/Facial Expressions/images/train/disgust/299.jpg")

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((128, 128))

expand\_input = np.expand\_dims(resize\_image,axis=0)

input\_data = np.array(expand\_input)

input\_data = input\_data/255

pred = loaded\_model.predict(input\_data)

result = pred.argmax()

result [2]



Checking our model prediction capability.

training\_set.class\_indices [2]

Background pattern

Description automatically generated with low confidence

Checking our indices of different emotions.

In the above code we gave the model a disgust image to test and we can see that the model has predicted as the given image as disgust by checking the indices.

CNN, or Convolutional Neural Network, is a type of deep neural network that is commonly used for image analysis and processing. It uses a combination of convolutional layers, pooling layers, and fully connected layers to extract features from an image, reduce its dimensionality, and classify it into one or more categories. By leveraging the hierarchical structure of images and learning the relevant features, CNNs can achieve high accuracy in a variety of computer vision tasks such as object detection, facial recognition, and image segmentation[3]. This the model we used to train our model in the above code.

# References

# [1] Facial recognition system. (2023, March 18). In *Wikipedia*. <https://en.wikipedia.org/wiki/Facial_recognition_system>

# [2] StudyGyaan. <https://www.youtube.com/watch?v=nLlbJjVgZyo>. Emotion Classification python file. <https://drive.google.com/file/d/1syrWpsgmcThF5twK0kzqai3GIOvmg2d7/view>

# [3] ChatGPT, personal communication, February 11, 2023.