A Capstone Project Report on

IMAGE TO EMOJI CONVERTER USING DEEP LEARNING AND CNN

Submitted by,

Fathima Rizniya S

Abstract

Nowadays, we are using several emojis or avatars to show our moods or feelings. They act as nonverbal cues of humans. They become the crucial part of emotion recognition, online chatting, brand emotion, product review, and a lot more. Data science research towards emoji-driven storytelling is increasing repeatedly. The detection of human emotions from images is very trendy, and possibly due to the technical advancements in computer vision and deep learning. In this deep learning project, to filter and locate the respective emojis or avatars, we will classify the human facial expressions.

Acknowledgement

A am using this capstone project as an opportunity to express my gratitude to everyone with me throughout the course. I am thankful for their inspiring guidance, leadership quality, and friendly advice for this project. This creates a very comfortable environment to complete the project at time with complete enthusiasm.

Further, I thank my mentor Anbu Joel. He has readily shared his immense knowledge as a great guide.

I certify that the work done by me for conceptualizing and completing this project is original and authentic.

Date – 31/07/2022 **Name** – Fathima Rizniya S

Certificate of Completion

I certify that the project titled “Image to Emoji Converter” was undertaken and completed (31/07/2022)

Mentor – Anbu Joel
Date – 31/07/2022
Place – Trichy

CHAPTER 1

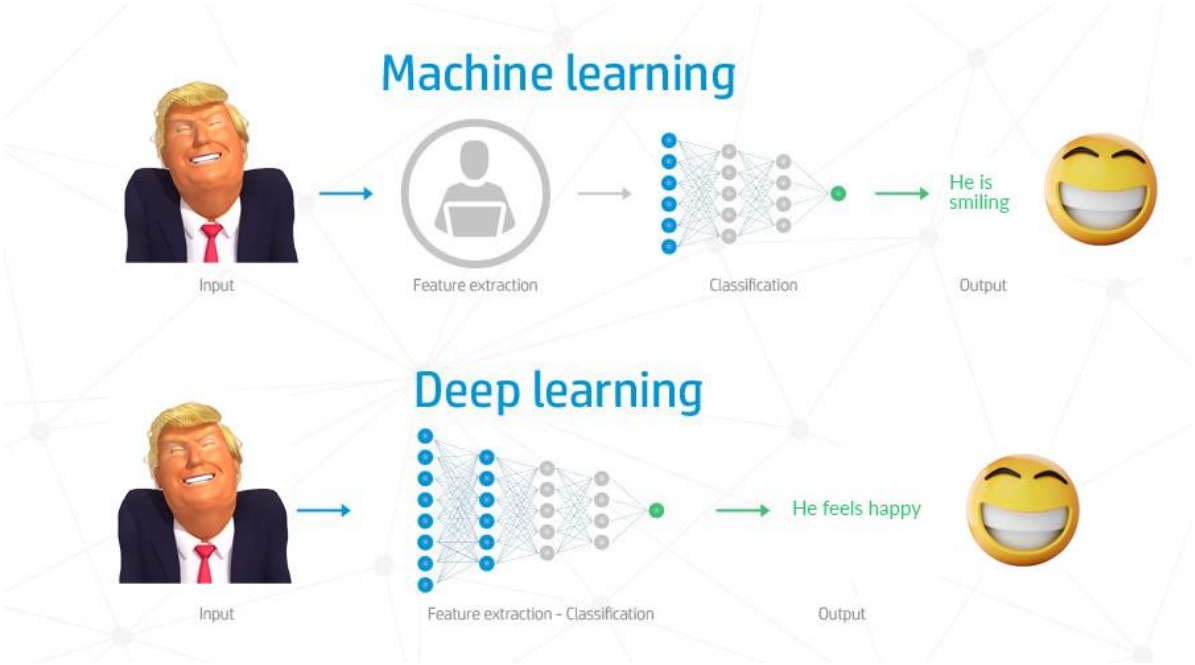
INTRODUCTION

1.1 Title & Objectives of the study

Title – Image to Emoji convertor using Deep learning

Deep learning

It is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning is an important element of data science, which includes statistics and predictive modelling.



Objective of the project–

To convert human facial expressions into emojis reflecting their emotions by Deep Learning technology, Python tools and CNN tenchnique.

1.2 Business under study

Image to Emoji converter model plays a vital role in social media. Because it reflects the emotion of the human just by recognizing their facial expressions. It make the interaction between one another to next level of internet space, like Meta. On the other hand this model reflects the client's mood on the time of online business discussions.

1.3 Tools and Techniques

Tools used are:

a) **Google Colab or Colab notebook –** It is an environment that is now provided by Google. This environment is based on Jupyter Notebook. It is one of the most efficient platforms for ML in the market. The only thing is everything in Colab will be cloud-based. You can work with many tools like TensorFlow, Pytorch, Keras on the Colab. Colab can improve your Python skills. We can also use a free GPU provided by Colab for extra processing. Google Drive is a storage method here.

b) **Tensorflow –** It is an open-source framework for numerical and large-scale ML. It uses both ML and neural network models. Tensorflow is a Machine Learning tool that is python friendly. It runs on both CPU and GPU. Also, TensorFlow trains and run models of neural networks. Image classification, NLP, etc. use these models.

c) **Numpy –** It is a Machine Learning tool used in scientific calculations. More advanced libraries like Tensorflow and Theano run on NumPy. For math-based calculations, this library comes in hand. It has an N-dimensional array, other sophisticated tools for data calculation. Numpy is a Python-based tool. It is multi-dimensional storage for generic data. Hence, we use it in various databases as well.

d) **Matplotlib –** It is a graph plotting library used in Python. With this, you can draw histograms, bar graphs, pie graphs, etc. We have a similar library called Glue used in Glueviz. That is also used for plotting graphs. The only difference is MatPlotLib is used for 2D drawing. Whereas glue can be used for 3D as well, also it can take images as input. It's built on Numpy arrays. The advantage is that it can take a large amount of data.

e) **Open CV –** It is a library which helps in computer vision. It runs on NumPy and many other tools. CV means computer vision. It is one of the most used tools in the world.

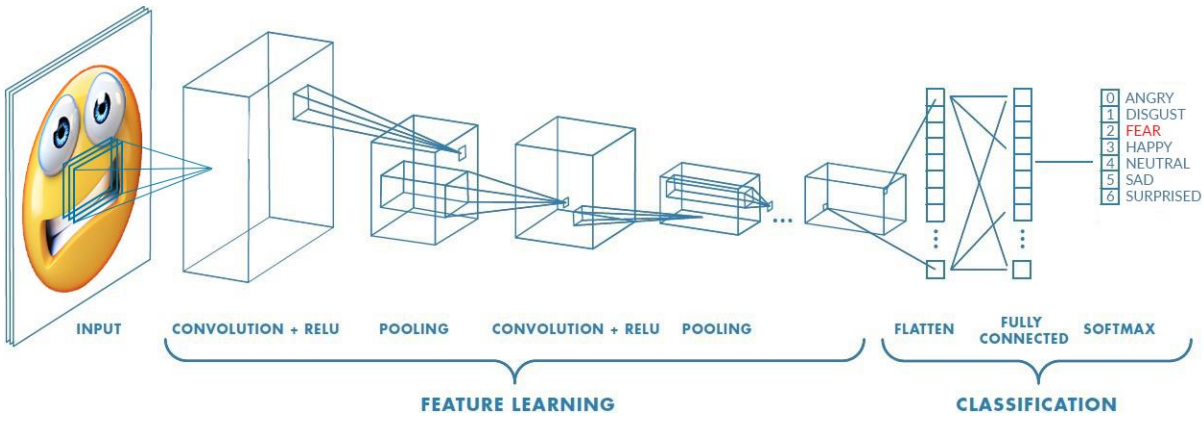
f) **Keras** – It is a high-level neural network API that is capable of running on top of TensorFlow or Theano. It is written in Python and was developed mainly on enabling faster experimentation. Keras deep learning library allows the user for easier and faster prototyping with the use of modularity, minimalism and easy extensibility. Keras is a deep learning tool that supports recurrent networks and convolutional networks individually as well as in the combination of the two. It also supports multi-input and multi-output training. It follows best practices for reducing cognitive load by offering consistent and simple APIs. Furthermore, it minimizes the number of user actions that are needed for common use cases and provides a clear feedback upon detection of any error.

g) **Tkinter** – It is the de facto way in Python to create Graphical User interfaces (GUIs) and is included in all standard Python Distributions. In fact, it's the only framework built into the Python standard library.

Technique in Image to Emoji converter is:

Convolutional Neural Networks(CNN) –

Convolutional Neural Networks also known as CNNs or ConvNets, are a type of feed-forward artificial neural network whose connectivity structure is inspired by the organization of the animal visual cortex. Small clusters of cells in the visual cortex are sensitive to certain areas of the visual field. Individual neuronal cells in the brain respond or fire only when certain orientations of edges are present. Some neurons activate when shown vertical edges, while others fire when shown horizontal or diagonal edges.



A convolutional neural network is a type of artificial neural network used in deep learning to evaluate visual information. These networks can handle a wide range of tasks involving images, sounds, texts, videos, and other media.

CHAPTER 2

DATA PREPARATION AND UNDERSTANDING

2.1 Data

The pictorial dataset used for this project is FER2013 (Facial Expression Recognition 2013). It contains 48*48-pixel grayscale face images. The images are located in the centre and occupy the same amount of space. Every type of expressions are in a set of every angle, age , gender and geometrical shape of faces. Below is the facial expression categories present in our dataset:

- 0 – angry
- 1 – disgust
- 2 – fear
- 3 – happy
- 4 – sad
- 5 – surprise
- 6 – neutral

Firstly, we build a deep learning model which classifies the facial expressions from the pictures. Then we will locate the already classified emotion with an avatar or an emoji.

2.2 Data Classes and Count Table

The dataset consist of two main classes namely,

- Train
- Test

Train data of 28709 images:

Expression	Angry	Disgust	Fear	Happy	Sad	Surprised	Neutral
Count	3995	436	4097	7215	4830	3171	4965

Test data of 7178 images:

Expression	Angry	Disgust	Fear	Happy	Sad	Surprised	Neutral
Count	958	111	1024	1774	1247	831	1233

CHAPTER 3

FITTING MODELS TO DATA

Convolution Neural Networks(CNN)

It have an input layer, an output layer, numerous hidden layers, and millions of parameters, allowing them to learn complicated objects and patterns. It uses convolution and pooling processes to sub-sample the given input before applying an activation function, where all of them are hidden layers that are partially connected, with the completely connected layer at the end resulting in the output layer. The output shape is similar to the size of the input image. It is the process of combining two functions to produce the output of the other function. The input image is convoluted with the application of filters in CNNs, resulting in a Feature map. CNN uses the same weights and biases for all neurons. Many filters can be created, each of which catches a different aspect from the input. Kernels are another name for filters.

Convolutional Layer

In convolutional neural networks, the major building elements are convolutional layers. This layer often contains input vectors, such as an image, filters, such as a feature detector, and output vectors, such as a feature map. The image is abstracted to a feature map, also known as an activation map, after passing through a convolutional layer.

Padding and Stride

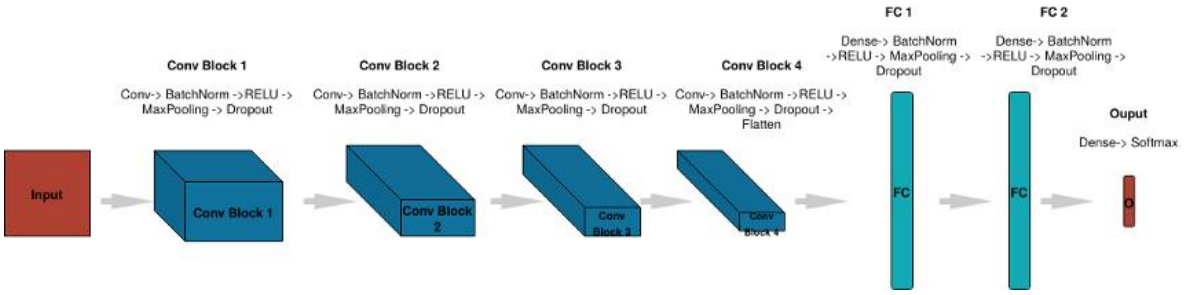
Padding is a term used in convolutional neural networks to describe how many pixels are added to an image when it is processed by the CNN kernel. If the padding in a CNN is set to zero, for example, every pixel value-added will have the value zero. If the zero padding is set to one, a one-pixel border with a pixel value of zero will be added to the image.

Stride determines how the filter convolves over the input matrix, i.e. how many pixels shift. When stride is set to 1, the filter moves across one pixel at a time, and when the stride is set to 2, the filter moves across two pixels at a time. The smaller the stride value, the smaller the output, and vice versa.

Pooling, ReLU, and others are will be explained in the further in this chapter

Facial Emotion Recognition using CNN

Build a convolution neural network(CNN) architecture and feed the FER2013 dataset to the model so that it can recognize emotion from images. We build the CNN model using the Keras layers in various steps. You can see each layer in the below diagram.



To build the network we use two dense layers, one flatten layer and four conv2D layers. We are going to use the Softmax equation to generate the model output. In the below steps will build a convolution neural network architecture and train the model on FER2013 dataset for Emotion recognition from images.

3.1 IMPORT

```
from google.colab import drive
drive.mount('/content/drive/')
import os
os.chdir("/content/drive/My Drive/")
!ls




#!pip install tensorflow
import tensorflow

#!pip install keras
import keras

import numpy as np
import cv2
import keras

from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from tensorflow.keras.optimizers import Adam
from keras.layers import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

Importing the data from google drive for colab and importing required tools and models from the python library such as tensorflow, Keras, numpy. And importing Sequential, load

model, Dense, Dropout, Flatten, conv2D, Adam, MaxPooling2D and ImageDataGenerator from keras.

3.2 INITIALIZE

```
train_dir = "archive (1)/train"
val_dir = "archive (1)/test"
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(48,48),
        batch_size=64,
        color_mode="grayscale",
        class_mode= "categorical")

validation_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=(48,48),
        batch_size=64,
        color_mode="grayscale",
        class_mode= "categorical")
```

```
Found 28719 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

```
or i in os.listdir("train/"):
      print(str(len(os.listdir("train/"+i))) +" "+ i +" images")
```

```
3995 angry images
436 disgust images
4097 fear images
7215 happy images
4965 neutral images
4830 sad images
3171 surprise images
```

```
for i in os.listdir("test/"):
      print(str(len(os.listdir("test/"+i))) +" "+ i +" images")
```

```
958 angry images
111 disgust images
1024 fear images
1774 happy images
1233 neutral images
1247 sad images
831 surprise images
```

Initializing the train and validation generators to build a model. It is done by rescaling of ImageDataGenerator function and fixing the target_size, batch_size, color_mode and class_mode for both train and validation directories respectively for CNN.

3.3 BUILD

```
emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))

emotion_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 46, 46, 32)        320

 conv2d_5 (Conv2D)            (None, 44, 44, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 22, 22, 64)        0
 2D)

 dropout_3 (Dropout)          (None, 22, 22, 64)        0

 conv2d_6 (Conv2D)            (None, 20, 20, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 10, 10, 128)       0
 2D)

 conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)         0
 2D)

 dropout_4 (Dropout)          (None, 4, 4, 128)         0

 flatten_1 (Flatten)          (None, 2048)              0
```

```
 dense_2 (Dense)              (None, 1024)              2098176

 dropout_5 (Dropout)          (None, 1024)              0

 dense_3 (Dense)              (None, 7)                 7175

=================================================================
Total params: 2,345,607
Trainable params: 2,345,607
Non-trainable params: 0
_________________________________________________________________
```

Building the convolution network architecture by adding Conv2D, MaxPooling2D, Dropout, Flatten, Dense with activation of "relu" and "softmax" at correct kernel and pool size.

- **Conv2D** is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.
- **MaxPooling2D**, downsamples the image along its spatial dimensions (height and width) by taking the maximum value over an image window (of size defined by pool_size) for each channel of the input. The window is shifted by strides along each dimension. Max pooling is done to in part to help over-fitting by providing an abstracted form of the representation.
- **Dropout** is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. 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.
- **ReLU** function is another non-linear activation function. It stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.
- **Softmax** function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution.

3.4 COMPILE

```
emotion_model.compile(loss='categorical_crossentropy',optimizer=Adam(learning_
rate=0.0001, decay=1e-6),metrics=['accuracy'])

emotion_model_info = emotion_model.fit(
        train_generator,
        steps_per_epoch=28709 // 64,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=7178 // 64)
```

```
Epoch 1/50
448/448 [==============================] - 11568s 26s/step - loss:
1.7982 - accuracy: 0.2627 - val_loss: 1.7023 - val_accuracy: 0.3324
Epoch 2/50
```

```
448/448 [==============================] - 441s 984ms/step - loss:
1.6223 - accuracy: 0.3672 - val_loss: 1.5393 - val_accuracy: 0.4083
Epoch 3/50
448/448 [==============================] - 439s 980ms/step - loss:
1.5224 - accuracy: 0.4160 - val_loss: 1.4542 - val_accuracy: 0.4488
Epoch 4/50
448/448 [==============================] - 447s 997ms/step - loss:
1.4517 - accuracy: 0.4464 - val_loss: 1.3943 - val_accuracy: 0.4714
Epoch 5/50
448/448 [==============================] - 457s 1s/step - loss: 1.3933
- accuracy: 0.4709 - val_loss: 1.3512 - val_accuracy: 0.4870
Epoch 6/50
448/448 [==============================] - 449s 1s/step - loss: 1.3457
- accuracy: 0.4902 - val_loss: 1.3092 - val_accuracy: 0.5008
Epoch 7/50
448/448 [==============================] - 441s 984ms/step - loss:
1.2992 - accuracy: 0.5057 - val_loss: 1.2722 - val_accuracy: 0.5183
Epoch 8/50
448/448 [==============================] - 441s 984ms/step - loss:
1.2637 - accuracy: 0.5217 - val_loss: 1.2808 - val_accuracy: 0.5047
Epoch 9/50
448/448 [==============================] - 442s 986ms/step - loss:
1.2316 - accuracy: 0.5399 - val_loss: 1.2348 - val_accuracy: 0.5339
Epoch 10/50
448/448 [==============================] - 430s 960ms/step - loss:
1.2017 - accuracy: 0.5518 - val_loss: 1.2158 - val_accuracy: 0.5340
Epoch 11/50
448/448 [==============================] - 433s 965ms/step - loss:
1.1744 - accuracy: 0.5628 - val_loss: 1.1820 - val_accuracy: 0.5516
Epoch 12/50
448/448 [==============================] - 431s 962ms/step - loss:
1.1485 - accuracy: 0.5726 - val_loss: 1.1646 - val_accuracy: 0.5534
Epoch 13/50
448/448 [==============================] - 432s 963ms/step - loss:
1.1245 - accuracy: 0.5783 - val_loss: 1.1483 - val_accuracy: 0.5643
Epoch 14/50
448/448 [==============================] - 431s 962ms/step - loss:
1.1004 - accuracy: 0.5898 - val_loss: 1.1406 - val_accuracy: 0.5699
Epoch 15/50
448/448 [==============================] - 435s 971ms/step - loss:
1.0754 - accuracy: 0.5982 - val_loss: 1.1322 - val_accuracy: 0.5695
Epoch 16/50
448/448 [==============================] - 431s 961ms/step - loss:
1.0482 - accuracy: 0.6118 - val_loss: 1.1183 - val_accuracy: 0.5798
Epoch 17/50
448/448 [==============================] - 432s 965ms/step - loss:
1.0298 - accuracy: 0.6159 - val_loss: 1.1163 - val_accuracy: 0.5778
Epoch 18/50
448/448 [==============================] - 433s 966ms/step - loss:
1.0058 - accuracy: 0.6266 - val_loss: 1.1011 - val_accuracy: 0.5865
Epoch 19/50
448/448 [==============================] - 431s 963ms/step - loss:
0.9859 - accuracy: 0.6354 - val_loss: 1.0941 - val_accuracy: 0.5837
Epoch 20/50
448/448 [==============================] - 430s 959ms/step - loss:
0.9699 - accuracy: 0.6394 - val_loss: 1.0866 - val_accuracy: 0.5935
Epoch 21/50
```

```
448/448 [==============================] - 431s 961ms/step - loss:
0.9433 - accuracy: 0.6514 - val_loss: 1.0894 - val_accuracy: 0.5931
Epoch 22/50
448/448 [==============================] - 431s 962ms/step - loss:
0.9186 - accuracy: 0.6631 - val_loss: 1.0761 - val_accuracy: 0.6010
Epoch 23/50
448/448 [==============================] - 441s 985ms/step - loss:
0.8915 - accuracy: 0.6704 - val_loss: 1.0681 - val_accuracy: 0.6014
Epoch 24/50
448/448 [==============================] - 436s 972ms/step - loss:
0.8772 - accuracy: 0.6790 - val_loss: 1.0696 - val_accuracy: 0.6021
Epoch 25/50
448/448 [==============================] - ETA: 0s - loss: 0.8457 -
accuracy: 0.6898
```

Compiling the above created model at a rate of epochs(=50), by dividing it into steps. The accuracy depends on the number of epochs. The model generated will be more efficient in the high number of epochs. In the whole process this compilation takes more time. But its just for model generation. So, it won't affect the GUI code after compiled model extraction.

3.5 SAVE

```
emotion_model.save_weights('model.h5')
```

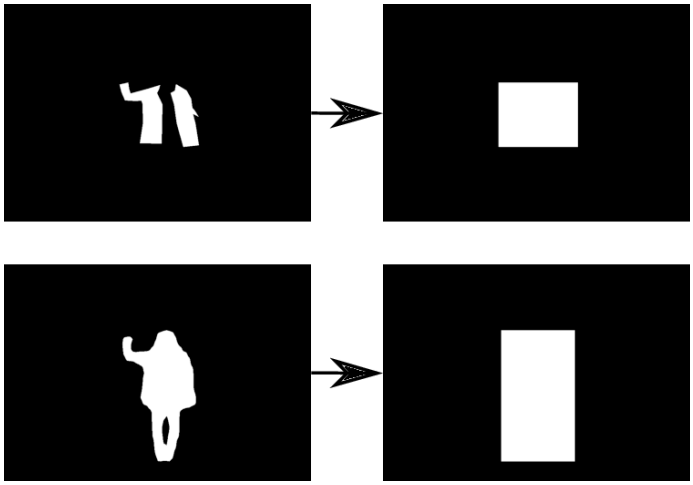
Save the compiled model in the name of "model.h5". And, make a separate file on drive for this "model.h5" in case of Google colab notebook.

3.6 BOUNDING BOX

```
cv2.ocl.setUseOpenCL(False)

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4:
"Neutral", 5: "Sad", 6: "Surprised"}

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    bounding_box =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    num_faces = bounding_box.detectMultiScale(gray_frame,scaleFactor=1.3,
minNeighbors=5)

    for (x, y, w, h) in num_faces:
        cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)
        roi_gray_frame = gray_frame[y:y + h, x:x + w]
```

```
        cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame,
(48, 48)), -1), 0)
        emotion_prediction = emotion_model.predict(cropped_img)
        maxindex = int(np.argmax(emotion_prediction))
        cv2.putText(frame, emotion_dict[maxindex], (x+20, y-60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    cv2.imshow('Video', cv2.resize(frame,(1200,860),interpolation =
cv2.INTER_CUBIC))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

To detect bounding boxes of face in the webcam and to predict the emotions we use OpenCV Haarcascade xml.



Bounding-box regression is a popular technique to refine or predict localization boxes in recent object detection approaches. is an imaginary rectangle that serves as a point of reference for object detection and creates a collision box for that object. This code defines the bounding box dimensions in all axis such as length and width.

CHAPTER 4

GUI AND MAPPING

Firstly, create a folder with emojis name and then save the images of different facial expression(cartonify images) with respect to the seven emotions which are present in dataset.

4.1 Importing libraries

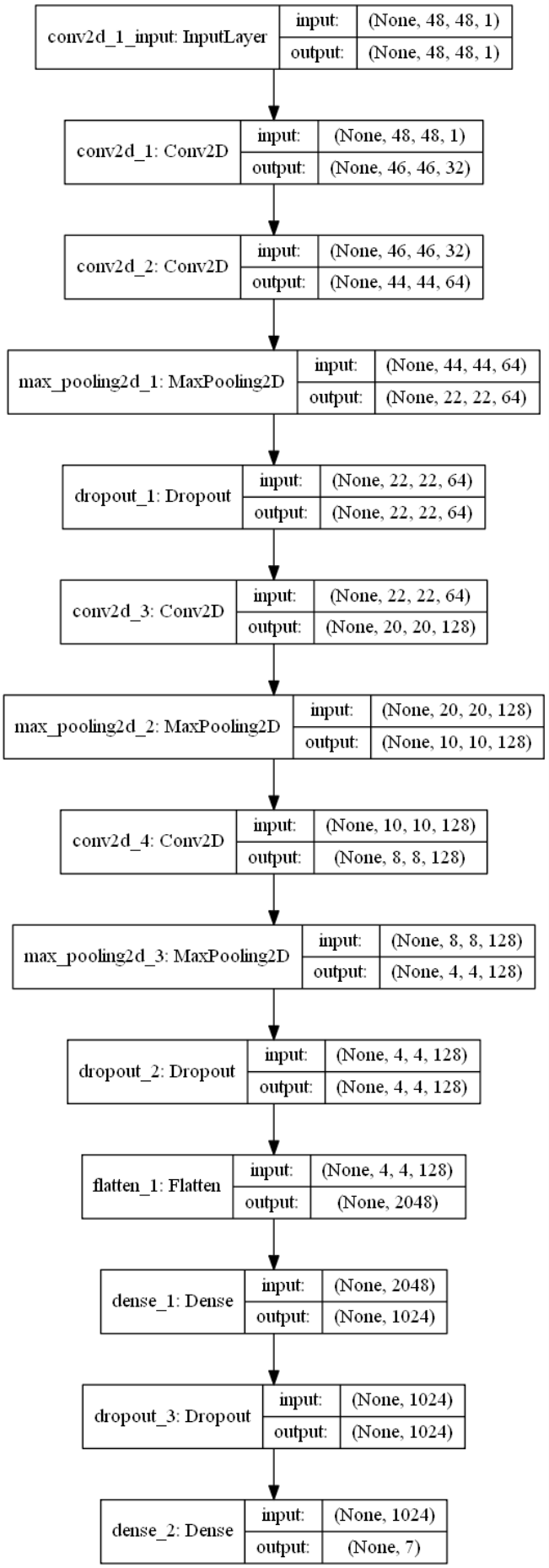
```
import tkinter as tk
from tkinter import *
import cv2
from PIL import Image, ImageTk
import os
import numpy as np
import cv2
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from keras.optimizers import Adam
from keras.layers import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

This method is similar as done before but tkinter, cv2 and PIL is added for the capturing and image processing. Here we need to reimport, because this code will run the project and it may be in separate notebook.

- **Tkinter** is the de facto way in Python to create Graphical User interfaces (GUIs) and is included in all standard Python Distributions. In fact, it's the only framework built into the Python standard library.
- **PIL** is the Python Imaging Library which provides the python interpreter with image editing capabilities. The Image module provides a class with the same name which is used to represent a PIL image.
- **OS** module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os. path* modules include many functions to interact with the file system.
- **ImageTk** module contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images.

4.2 Model Creation

It involves the addition of different Keras layers to create a deep learning model shown in the below image.


conv2d_1_input: InputLayer
input: (None, 48, 48, 1)
output: (None, 48, 48, 1)
conv2d_1: Conv2D
input: (None, 48, 48, 1)
output: (None, 46, 46, 32)
conv2d_2: Conv2D
input: (None, 46, 46, 32)
output: (None, 44, 44, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 44, 44, 64)
output: (None, 22, 22, 64)
dropout_1: Dropout
input: (None, 22, 22, 64)
output: (None, 22, 22, 64)
conv2d_3: Conv2D
input: (None, 22, 22, 64)
output: (None, 20, 20, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 20, 20, 128)
output: (None, 10, 10, 128)
conv2d_4: Conv2D
input: (None, 10, 10, 128)
output: (None, 8, 8, 128)
max_pooling2d_3: MaxPooling2D
input: (None, 8, 8, 128)
output: (None, 4, 4, 128)
dropout_2: Dropout
input: (None, 4, 4, 128)
output: (None, 4, 4, 128)
flatten_1: Flatten
input: (None, 4, 4, 128)
output: (None, 2048)
dense_1: Dense
input: (None, 2048)
output: (None, 1024)
dropout_3: Dropout
input: (None, 1024)
output: (None, 1024)
dense_2: Dense
input: (None, 1024)
output: (None, 7)

```
emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))
emotion_model.load_weights('model.h5')

emotion_model.summary()
cv2.ocl.setUseOpenCL(False)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 46, 46, 32)        320

conv2d_17 (Conv2D)           (None, 44, 44, 64)        18496

max_pooling2d_12 (MaxPoolin  (None, 22, 22, 64)        0
g2D)

dropout_12 (Dropout)         (None, 22, 22, 64)        0

conv2d_18 (Conv2D)           (None, 20, 20, 128)       73856

max_pooling2d_13 (MaxPoolin  (None, 10, 10, 128)       0
g2D)

conv2d_19 (Conv2D)           (None, 8, 8, 128)         147584

max_pooling2d_14 (MaxPoolin  (None, 4, 4, 128)         0
g2D)

dropout_13 (Dropout)         (None, 4, 4, 128)         0

flatten_4 (Flatten)          (None, 2048)              0

dense_8 (Dense)              (None, 1024)              2098176

dropout_14 (Dropout)         (None, 1024)              0

dense_9 (Dense)              (None, 7)                 7175
```

```
=================================================================
Total params: 2,345,607
Trainable params: 2,345,607
Non-trainable params: 0
_________________________________________________________________
```

The model type that we will be using is Sequential. **Sequential** is the easiest way to build a model in Keras. It allows you to build a model layer by layer. Each layer has weights that correspond to the layer the follows it.

We use the **'add()'** function to add layers to our model. We will add two layers and an output layer.

'Dense' is the layer type. Dense is a standard layer type that works for most cases. In a dense layer, all nodes in the previous layer connect to the nodes in the current layer.

We have 7 nodes in each of our input layers. This number can also be in the hundreds or thousands. Increasing the number of nodes in each layer increases model capacity. I will go into further detail about the effects of increasing model capacity shortly.

'Activation' is the activation function for the layer. An activation function allows models to take into account nonlinear relationships.

The activation function we will be using is **ReLU** or Rectified Linear Activation. Although it is two linear pieces, it has been proven to work well in neural networks.

4.3 Mapping of facial emotion with Emoji

```
emotion_dict = {0: "   Angry   ", 1: "Disgusted", 2: "  Fearful  ", 3:
"   Happy   ", 4: "  Neutral  ", 5: "    Sad    ", 6: "Surprised"}

emoji_dist={0:"C:/Users/Jothy Natarajan/Downloads/Angry
emoji.png",1:"C:/Users/Jothy Natarajan/Downloads/Disgusted
emoji.png",2:"C:/Users/Jothy Natarajan/Downloads/Fear
emoji.png",3:"C:/Users/Jothy Natarajan/Downloads/Happy
emoji.png",4:"C:/Users/Jothy Natarajan/Downloads/Neutral
emoji.png",5:"C:/Users/Jothy Natarajan/Downloads/Sad
emoji.png",6:"C:/Users/Jothy Natarajan/Downloads/Surprised emoji.png"}


global last_frame1
last_frame1 = np.zeros((480, 640, 3), dtype=np.uint8)
global cap1
show_text=[0]
def show_vid():
    cap1 = cv2.VideoCapture(0)
    if not cap1.isOpened():
        print("cant open the camera1")
    flag1, frame1 = cap1.read()
```

```
    frame1 = cv2.resize(frame1,(600,500))

    bounding_box =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    gray_frame = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
    num_faces = bounding_box.detectMultiScale(gray_frame,scaleFactor=1.3,
minNeighbors=5)

    for (x, y, w, h) in num_faces:
        cv2.rectangle(frame1, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)
        roi_gray_frame = gray_frame[y:y + h, x:x + w]
        cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray_frame,
(48, 48)), -1), 0)
        prediction = emotion_model.predict(cropped_img)

        maxindex = int(np.argmax(prediction))
        cv2.putText(frame1, emotion_dict[maxindex], (x+20, y-60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
        show_text[0]=maxindex
    if flag1 is None:
        print ("Major error!")
    elif flag1:
        global last_frame1
        last_frame1 = frame1.copy()        pic = cv2.cvtColor(last_frame1,
cv2.COLOR_BGR2RGB)

        img = Image.fromarray(pic)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
        lmain.after(10, show_vid)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        exit()


def show_vid2():
    frame2=cv2.imread(emoji_dist[show_text[0]])
    pic2=cv2.cvtColor(frame2,cv2.COLOR_BGR2RGB)
    img2=Image.fromarray(frame2)
    imgtk2=ImageTk.PhotoImage(image=img2)
    lmain2.imgtk2=imgtk2
    lmain3.configure(text=emotion_dict[show_text[0]],font=('arial',45,'bold'))

    lmain2.configure(image=imgtk2)
    lmain2.after(10, show_vid2)

if __name__ == '__main__':
    root=tk.Tk()
    img = ImageTk.PhotoImage(Image.open("Logo.png"))
```

```
    heading = Label(root,image=img,bg='black')

    heading.pack()
    heading2=Label(root,text="Photo to Emoji",pady=20,
font=('arial',45,'bold'),bg='black',fg='#CDCDCD')


    heading2.pack()
    lmain = tk.Label(master=root,padx=50,bd=10)     lmain2 =
tk.Label(master=root,bd=10)


    lmain3=tk.Label(master=root,bd=10,fg="#CDCDCD",bg='black')
    lmain.pack(side=LEFT)
    lmain.place(x=50,y=250)
    lmain3.pack()
    lmain3.place(x=960,y=250)
    lmain2.pack(side=RIGHT)
    lmain2.place(x=900,y=350)



    root.title("Photo To Emoji")
    root.geometry("1400x900+100+10")
    root['bg']='black'
    exitbutton = Button(root,
text='Quit',fg="red",command=root.destroy,font=('arial',25,'bold')).pack(side
= BOTTOM)
    show_vid()
    show_vid2()
    root.mainloop()
```

GUI Programming

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

Standard Attributed for GUI

- Dimensions
- Fonts
- Colours
- Cursors
- Anchors

- Bitmaps

Methods for Geometry Management

- The pack(): This method manages the geometry of widgets in blocks
- The grid(): This method organizes widgets in a tabular structure
- The place(): This method organizes the widgets to place them in a specific position

Mapping

It concerned with the development of math strategies associated with the problem of correlating digitized gray-shade data obtained from the common area of two overlapping photos.

CHAPTER 5

CONCLUSION

This project is based on the Keras library of deep learning technology. In order to recognize facial emotions, we have built a convolution neural network. After that, we fed our model with the FER2013 dataset. And finally, we map each facial emotion with its corresponding emojis or avatars. To detect the bounding box of images in the webcam we use the OpenCV's Haarcascade XML. In the end, we serve these boxes to the trained model for the purpose of classification.

CHAPTER 6

REFERENCES

FER2013 dataset