Sequence learning (e.g. action sequences in Smart Home environments, gaze patterns) via ML/DL techniques

Case Study: Real-time Emotion Prediction With Deep Learning

Kalu Chibueze Augustine, 26184 (Chibueze-Augustine.kalu@hsrw.org (mailto:Chibueze-Augustine.kalu@hsrw.org))

Kingsly Mbaaka Mukong, 26318 (kingsly-mbaaka.mukong@hsrw.org (mailto:kingsly-mbaaka.mukong@hsrw.org))

Course: CI_W.01 Ambient Intelligent Systems, SS2021

Lecturers: Prof. Dr. rer. Nat. Kai Essig

Lecturers: Prof. Dr.-Ing. Christian Ressel

Table of Contents

1.0 Introduction

Since the first application of automated facial recognition ***project man-machine*** in the 1960s by Woody Bledsoe et al, facial recognition system has been an active and evolving area of research[1] . Facial recognition systems, which initially began as a type of computer application, have over the years seen an increased use in smartphones as a facial unlock feature and in other forms of technology, such as human computer interaction. While the countless use cases for face recognition system outlines it importance, some challenges limit it from reaching its potential. These challenges can be summarized below[2]:

- **Illumination**: A small change in lighting can pose a difficulty for automatic facial recognition, thus having a significant impact on its outcomes.
- **Pose Variation**: When a person's head moves and their viewing angle changes, the pose of their face changes which affect the expected output
- **Occlusion**: Occlusion refers to a blocking of one or more areas of the face, preventing the entire face from being used as an input image.

- **Low-resolution**: A typical training image should have a minimum resolution of 16 x16 pixels. A low-resolution image is one with a resolution of less than 16 x16 pixels. This includes images collected from small scale standalone cameras, such as CCTV cameras in streets, security cameras and ATM cameras.
- **Ageing**: changes in Facial texture and appearance resulting from ageing can be problem when designing a facial recognition system.
- **Model Complexity**: complexity in facial recognition systems model can make them unsuitable for real time performance on embedded devices.

Emotion recognition systems harnesses the technology of facial recognition system to predict facial emotion, hence having the limitations of a facial recognition system.
In this project, we applied the technology of facial recognition system to build a context aware system that is capable of detecting common human emotions such as angry, disgust, fear, happy, sad, surprise and finally neutral (absence of emotion). The scope covered in this project include

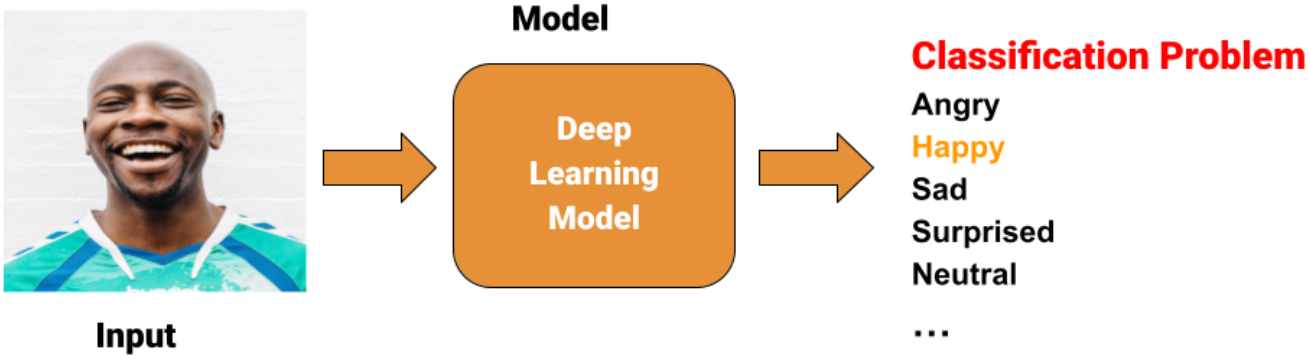
- Designing a Convolutional Neural Network,
- Train it from scratch by feeding it batches of images and
- Then export the trained model to be reused with real-time image data

2.0 Emotion Recorgnition Techniques

The abilty to read emotion from facial expressions is one of those human traits that we develop subconciously. For computers this may be a bit challenging because of some inconsistencies and drawbacks that exist with training a model using facial recognition system. Common emotion recorgnition techniques that exist today using computer are:

2.1 Vision based technique

This technique employs the use of a camera to recorgnize facial expression using data from a pretrained neural network. This technique can be somewhat inaccurate because facial deformation or persons with personallty traits like *Alexithymia* can make results from the vision based technique erroneous. We would be using the vision based technique for this project. The image below is custom made to explain how a vision based technique operates

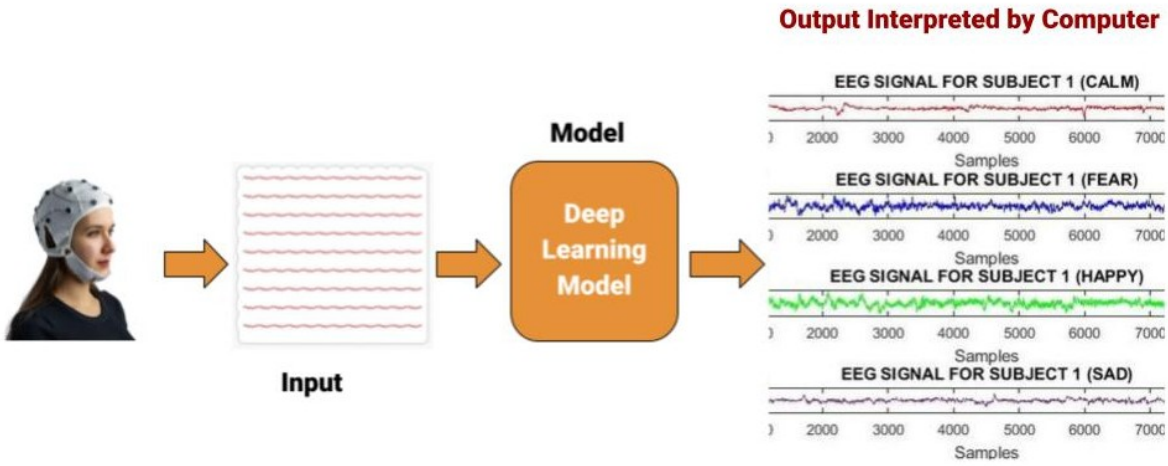


2.2 Bio-signals/Physiological based technique

This is the domain of medical data which makes use of biomedical signals such as

- electroencephalography (EEG),
- electrocorticography, which capture the electrical fields that are produced by the activity of the brain cells,

- magnetoencephalography that captures the magnetic fields that are produced by the electrical activity of the brain cells,
- electrocardiography that records the electrical activity that arises from the depolarization and repolarization activity of the heart,
- electromyography that records the electric potential that is generated by the muscle cells,
- electrooculography that records the electric potential generated by the cornea and the retinal activity. This technique presents a more accurate result. The image below is also custom made thought the Output signals was taken from Researchgate[3] (https://www.researchgate.net/publication/339809459_Emotion_recognition_using_electroencephalogram_si



3.0 Dataset (FER 2013)

Facial Expression Recognition 2013 is a well-studied dataset that was used in the past Kaggle competition titled **"Challenges in Representation Learning: Facial Expression Recognition Challenge" (https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge)**. The original FER data set was prepared by Pierre Luc Carrier and Aaron Courville by web crawling face images with emotion related keywords[4] . The images are filtered by human labelers, but the label accuracy is not very high. The dataset's 35,887 contained images are normalized to 48x48 pixels in grayscale. FER2013 is, however, not a balanced dataset, as it contains images of 7 facial expressions, with classification for

- Angry (3995)
- Disgust (436)
- Fear (4097)
- Happy (7215)
- Sad (4830),
- Surprise (3171), and
- Neutral (4965).

In [1]:

```
# count number of train images for each expression
import os

# input path for the images
base_path = "data/archive/"

for expression in os.listdir(base_path + "train/"):
    print(str(len(os.listdir(base_path + "train/" + expression))) + " " + expressio
```

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

Except for the ‘disgust' category, the face expressions in our training dataset are rather balanced. To fix the imbalance, we could argument data set using Generative Adversarial Networks(GANs). This approach was developed by Ian Goodfellow and his colleagues in 2014, the GANs approach learns to produce new data with the same statistics as the training set[5] . Data generation can also be performed with Keras inbuilt ImageDataGenerator class.

3.1 Review of the Dataset

The **FER 2013** data set consists of 48x48 pixel grayscale images of faces. The faces were automatically registered so that the each face is more or less centered and occupies about the same amount of space. Also 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 and the structure of the folder containing our images is as below;

```
data/
    archive/
            test/
                angry/
                disgust/
                fear/
                happy/
                neutral/
                sad/
                surprise/
             train/
                angry/
                disgust/
                fear/
                happy/
                neutral/
                sad/
                surprise/
```

Most of our images (75%) are contained inside the train folder, and the remaining images (25%) are inside the validation folder.

3.2 Examination of the Dataset

In [2]:

```
import matplotlib.pyplot as plt

from keras.preprocessing.image import load_img, img_to_array
# size of the image: 48*48 pixels
pic_size = 48



plt.figure(0, figsize=(12,20))
cpt = 0

for expression in os.listdir(base_path + "train/"):
    for i in range(1,6):
        cpt = cpt + 1
        plt.subplot(7,5,cpt)
        img = load_img(base_path + "train/" + expression + "/" +os.listdir(base_pat
        plt.imshow(img, cmap="gray")

plt.tight_layout()
plt.show()
```

With the human eye it is not too difficult to predict the emotions from the face, but it may be a bit challenging for a predictive algorithm because:

- the images have a low resolution
- the faces are not in the same position

- some images have text written on them
- some people hide part of their faces with their hands However all this diversity of images will contribute to make a more generalizable and realistic model.

4.0 Tools

4.1 Program and IDE

4.1.1Anaconda Software

Anaconda is a distribution of the Python and R programming languages for scientific computing that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012[7]
link on environment setup: Getting started with conda (https://conda.io/projects/conda/en/latest/user-guide/getting-started.html)

4.1.2 Jupyter Notebook

This a is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet[8].

4.2 Libraries and APIs

4.2.1 Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library (TensorFlow is a free and open-source software library for machine learning). We will use Keras to build, train and export out Neural Network.

4.2.2 Flask

Flask is a micro web framework written in Python that will allow us to serve our model directly into a web interface.

4.2.3 Open CV

OpenCV is a computer vision library with C++, Python and Java interfaces mainly aimed at real-time computer vision. We will use this library to automatically detect faces in images.

4.3 Install Libraries and Dependencies (after Anaconda environment setup)

```
    - Opencv
        - pip install opencv-python
    - pip install setuptools
    - python3 -m pip install --upgrade pip setuptools wheel
    - pip install OpenNMT-py
    - pip install opencv-python
    - pip install keras
    - pip3 install --user --upgrade tensorflow  # install in $HOME, or
    - pip install --upgrade tensorflow # virtual environment
    - pip install camera
```

4.4 Importing Libraries and Dependencies

In [3]:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import utils
import os
%matplotlib inline

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import plot_model

from IPython.display import SVG, Image
from livelossplot.tf_keras import PlotLossesCallback
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
Tensorflow version: 2.6.0
```

Our model works with Tensorflow version of atleat 2.0

5.0 Dataset Augmentation using Keras

To train a deep learning model, the model is fed with batches of data. Keras has a very useful class called ImageDataGenerator which is used to automatically feed data from a local directory. The ImageDataGenerator perform data augmentation while getting the images (randomly rotating the image, zooming, etc.).The function flow_from_directory() specifies how the generator should import the images (path, image size, colors, etc.).

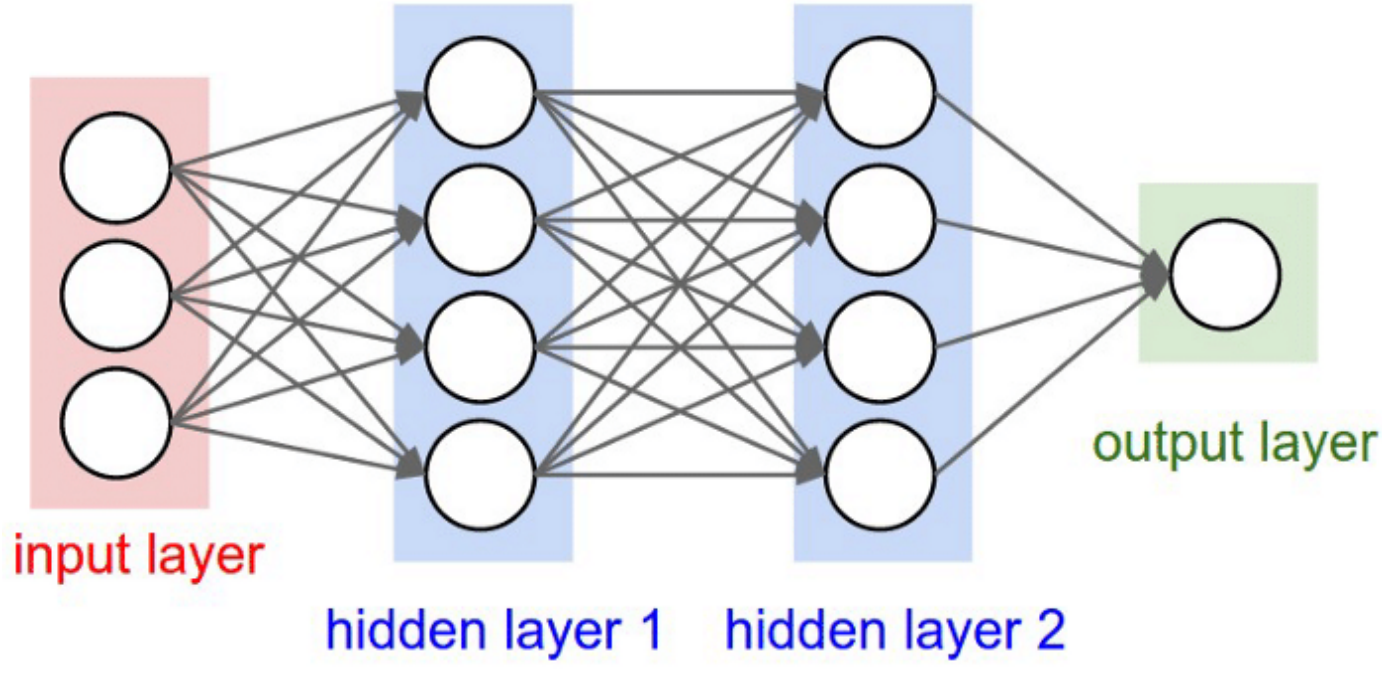
In [4]:

```
# number of images to feed for every batch
img_size = 48
batch_size = 64

datagen_train = ImageDataGenerator()
datagen_validation = ImageDataGenerator()

train_generator = datagen_train.flow_from_directory(base_path + "train/",
                                                    target_size=(pic_size,pic_size)
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=True)

validation_generator = datagen_validation.flow_from_directory(base_path + "test/",
                                                    target_size=(pic_size,pic_size)
                                                    color_mode="grayscale",
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False)
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

6.0 Understanding Neural network(NN)

We chose the convolution neural network for this project because it requires fewer parameters to learn as compared to a classical fully connected layer, also it is very efficient when extracting features from images for image analysis. A quick detour on what a neural network is - it is a learning framework that comprises of multiple layers of nodes, with each node getting weighted input data and passing it into an activation function which then outputs a result. A deep neural network on the other hand is a neural network with a lot of hidden layers. It is also important to note that in a typical neural network every node in one layer is linked to all nodes of the next layer[9].



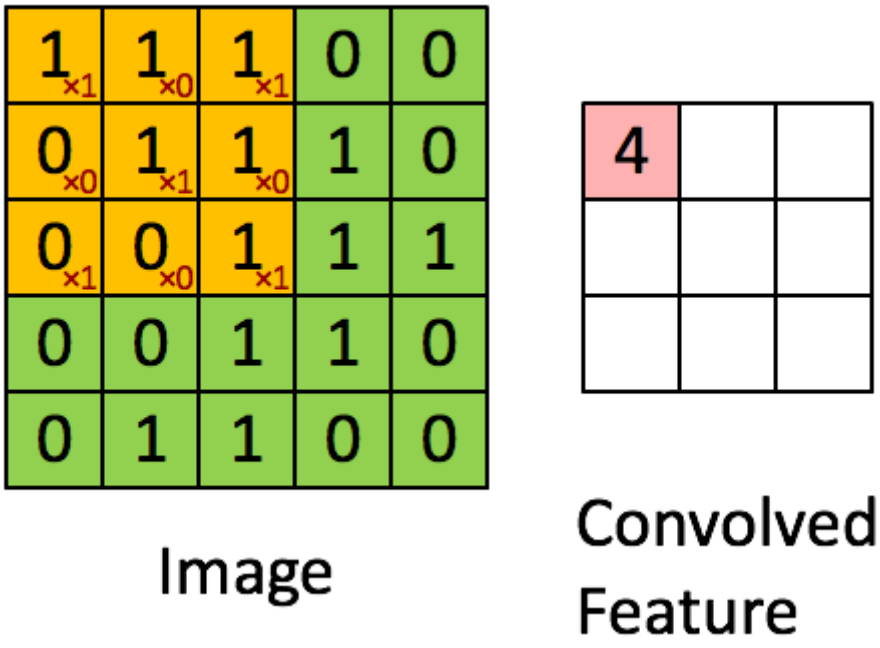
6.1 Defining the Convolution Neural Network Architecture(CNN)

Our CNN has a global architecture of 4 convolutional layers and 2 fully connected layers which focuses on using relevant features from our images to properly classify our images dataset. Filters (also known as kernels) are used by CNNs to recognize whether characteristics, such as edges, are present throughout an image. A filter is just a set of weighted values that have been taught to detect specific traits. The filter scans each area of the image to see if the feature it's looking for is there. The filter performs a convolution operation, which is an element-wise product and sum between two matrices, to generate a value expressing how confident it is that a specific feature is there. A stride value can also be used to apply a filter over the input image at different intervals. The stride value specifies how far the filter should move at every stage . The following equation can be used to calculate the output dimensions of a strided convolution:

$$n_{out} = floor(\frac{n_{in} - f}{s}) + 1$$

Where:

- n_in = dimension of the input image,
- f = window size, and
- s = stride.



The raw picture values are represented by the green matrix. A ‘filter' or ‘kernel' is the orange sliding matrix. At each step, this filter moves one pixel over the image (stride). We multiply the filter with the relevant elements of the base matrix in each step and add the results. To introduce non-linerity into our neural network, we used the Rectified Linear Unit(ReLU) activation function. Pooling was also used to reduce the dimensionality of each feature while retaining the most important information.

6.2 Defining our Convulation layers

In [5]:

```
# number of possible label values
nb_classes = 7

# Initialising the CNN
model = Sequential()

# 1 - Convolution
model.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 2nd Convolution layer
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flattening
model.add(Flatten())

# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(nb_classes, activation='softmax'))

opt = Adam(lr=0.0005)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
/home/razer/anaconda3/lib/python3.8/site-packages/keras/optimizer_v2/
optimizer_v2.py:355: UserWarning: The `lr` argument is deprecated, us
e `learning_rate` instead.
  warnings.warn(
```

For each layer, we additionally employ certain standard techniques:

- Batch normalization to improves the performance and stability of Neural Networks(NNs).
- Dropout to reduces overfitting by randomly not updating the weights of some nodes.

Softmax was chosen as our final activation function since it is frequently utilized for multi-label categorization.

6.2 Training the model

In [6]:

```
epochs = 10
steps_per_epoch = train_generator.n//train_generator.batch_size
validation_steps = validation_generator.n//validation_generator.batch_size

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
                              patience=2, min_lr=0.00001, mode='auto')
checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_accuracy',
                             save_weights_only=True, mode='max', verbose=1)
callbacks = [PlotLossesCallback(), checkpoint, reduce_lr]

history = model.fit(x=train_generator,steps_per_epoch=steps_per_epoch,epochs=epochs
)
model.save("model_weights.h5")
```

```
Epoch 1/10
448/448 [==============================] - 383s 853ms/step - loss: 1.
7770 - accuracy: 0.3205 - val_loss: 1.7404 - val_accuracy: 0.3777
Epoch 2/10
448/448 [==============================] - 382s 852ms/step - loss: 1.
4552 - accuracy: 0.4397 - val_loss: 1.4914 - val_accuracy: 0.4477
Epoch 3/10
448/448 [==============================] - 381s 852ms/step - loss: 1.
3081 - accuracy: 0.4996 - val_loss: 1.3357 - val_accuracy: 0.4802
Epoch 4/10
448/448 [==============================] - 385s 860ms/step - loss: 1.
2192 - accuracy: 0.5382 - val_loss: 1.1547 - val_accuracy: 0.5520
Epoch 5/10
448/448 [==============================] - 389s 869ms/step - loss: 1.
1634 - accuracy: 0.5569 - val_loss: 1.1826 - val_accuracy: 0.5413
Epoch 6/10
448/448 [==============================] - 387s 863ms/step - loss: 1.
1085 - accuracy: 0.5793 - val_loss: 1.1296 - val_accuracy: 0.5682
Epoch 7/10
448/448 [==============================] - 386s 862ms/step - loss: 1.
0642 - accuracy: 0.5922 - val_loss: 1.0889 - val_accuracy: 0.5878
Epoch 8/10
448/448 [==============================] - 405s 905ms/step - loss: 1.
0236 - accuracy: 0.6092 - val_loss: 1.0561 - val_accuracy: 0.6060
Epoch 9/10
448/448 [==============================] - 397s 886ms/step - loss: 0.
9841 - accuracy: 0.6285 - val_loss: 1.0724 - val_accuracy: 0.6011
Epoch 10/10
448/448 [==============================] - 327s 730ms/step - loss: 0.
9479 - accuracy: 0.6395 - val_loss: 1.1076 - val_accuracy: 0.5947
```

Our top model had a validation accuracy of around 65 percent, which is rather impressive considering our target

class contains seven possible values!

Keras examines each epoch to see if our model outperformed the previous epoch's models. The new best model weights are saved into a file if this is the case. This will allow us to load our model's weights without having to retrain it if we wish to use it in a different context.

We finally then stored the CNN's structure (layers, etc.) to a file:

Saving Model as JSON String

In [7]:

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

7.0 Real-time Prediction

7.1 Class for loading model and weights

In [8]:

```
from tensorflow.keras.models import model_from_json
import numpy as np

import tensorflow as tf


class FacialExpressionModel(object):

    EMOTIONS_LIST = ["Angry", "Disgust",
                     "Fear", "Happy",
                     "Neutral", "Sad",
                     "Surprise"]

    def __init__(self, model_json_file, model_weights_file):
        # load model from JSON file
        with open(model_json_file, "r") as json_file:
            loaded_model_json = json_file.read()
            self.loaded_model = model_from_json(loaded_model_json)

        # load weights into the new model
        self.loaded_model.load_weights(model_weights_file)
        self.loaded_model.make_predict_function()

    def predict_emotion(self, img):
        self.preds = self.loaded_model.predict(img)
        return FacialExpressionModel.EMOTIONS_LIST[np.argmax(self.preds)]
```

7.2 Getting frames and doing prediction

We create a camera class that performs the following tasks:

- obtain the picture stream from our webcam,

- use OpenCV to detect faces and create bounding boxes,
- convert the faces to greyscale, rescale, and send them to our pre-trained Neural Network,
- get a prediction back from our Neural Network and add caption to the webcam image frame,
- then finally return the image stream the final image stream

In [9]:

```
import cv2

import numpy as np

facec = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
model = FacialExpressionModel("model.json", "model_weights.h5")
font = cv2.FONT_HERSHEY_SIMPLEX

class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)

    def __del__(self):
        self.video.release()

    # returns camera frames along with bounding boxes and predictions
    def get_frame(self):
        _, fr = self.video.read()
        gray_fr = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
        faces = facec.detectMultiScale(gray_fr, 1.3, 5)

        for (x, y, w, h) in faces:
            fc = gray_fr[y:y+h, x:x+w]

            roi = cv2.resize(fc, (48, 48))
            pred = model.predict_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr,(x,y),(x+w,y+h),(255,0,0),2)

        return fr
```

7.3 Function for showing output video

In [10]:

```
def gen(camera):
    while True:
        frame = camera.get_frame()
        cv2.imshow('Facial Expression Recognization',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()
```

7.4 Running the code

In [11]:

```
gen(VideoCamera())
```

8.0 Use case and Application

Emotion recognition is a growing field of research currently used by many big companies to evaluate consumer mood toward their brand or product. The opportunity made available by this technology goes beyond market research and digital advertising. Our focus on this project was to design a context-aware system that could be used to evaluate users' real-time emotional states as system input and introduce adaptive changes based on this context. This could be incorporated into the smart home system, thus making the system capable of making complex decisions and adapting to the user's emotional cues.

Conclusion

Our application worked no doubt but there is still room for improvement especially if it is to be deployed into a real-world application. With an optimized dataset and additional CNN layers, our model could reach an accuracy higher than its current 65 percent benchmark. Also, the large computational requirements for training a model might be a problem when dealing with a larger dataset. With a normal PC of 16G RAM, Intel(R) UHD Graphics 630, and Nvidia GeForce GTX 1060 it took a couple of hours to attain the validation accuracy of 65 percent.

Another reason why our model is not very accurate is due to the fact that it was trained with black/white low resolution images. Some images even had texts on them which also affects the learning process of our model.

Again, our dataset has just 436 disgust images and 7215 happy images. This tend to make our model interpret less disgust images and more inclined towards happy. A more balanced dataset will give better results and accuracy

References

[1] Nilsson, Nils J. (2009). The Quest for Artificial Intelligence. Cambridge University Press. ISBN 9781139642828.

[2] Kajal Mishra, "Challenges Faced by Facial Recognition System" (https://www.pathpartnertech.com/challenges-faced-by-facial-recognition-system/) retrived on August 20th 2021

[3] Nadira A.N. et al. Emotion recognition using electroencephalogram signal. Indonesian Journal of Electrical Engineering and Computer Science August 2019

[4] Minaee, Shervin and Amirali Abdolrashidi. "Deep-Emotion: Facial Expression Recognition Using Attentional Convolutional Network" taken from https://pubmed.ncbi.nlm.nih.gov/33925371/ (https://pubmed.ncbi.nlm.nih.gov/33925371/) on August 20th 2021.

[5] Goodfellow, Ian J., ET AL. "Challenges in representation learning: A report on three machine learnig contests." Internatiional Conference on Meutal Information Processing. Springer, Berlin, Heidelberg, 2013 (Pierre-Luc Carrier and Aaron Courville)

[6] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Nets (PDF). Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[7] Jonathan Oheix. From raw images to real-time predictions with Deep Learning -Face expression recognition using Keras, Flask and OpenCV. January 2019. Taken on August 27th 2021, from https://towardsdatascience.com/from-raw-images-to-real-time-predictions-with-deep-learning-ddbbda1be0e4 (https://towardsdatascience.com/from-raw-images-to-real-time-predictions-with-deep-learning-ddbbda1be0e4)

[8] Jupyter official documentationWhat is Jupyter? (https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html) retrived on August 28th 2021

[9] Alejandro Escontrela. Convolutional Neural Networks from the ground up - A NumPy implementation of the famed Convolutional Neural Network: one of the most influential neural network architectures to date. June 2018. Taken on August 27th, from https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1 (https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1)

[10] OpenCV (https://docs.opencv.org/4.5.1/) official documentation

[11] Tensorflow (https://www.tensorflow.org/lite/tutorials/model_maker_image_classification) documentation. Image classification with TensorFlow Lite Model Maker. Retrived on August 30th 2021, from https://www.tensorflow.org/lite/tutorials/model_maker_image_classification (https://www.tensorflow.org/lite/tutorials/model_maker_image_classification)

[12] Conda (https://conda.io/projects/conda/en/latest/user-guide/getting-started.html) official documentation retrived on August 28th 2021

The End!
