****
National Institute of Applied Sciences and Technology

****
Karunya institute of technology and sciences

Summer internship

Topic: Real time emotion recognition cloud-based solution

Departement : Computer science

Accomplished by:

Hanen Regaieg

Supervised by:

Dr. Julia Punitha Malar Dhas

Academic year: 2021-2022

Abstract:

It will be amazing if machines become emotionally intelligent enough to make decisions based on people's emotions that can be detected by computer vision. We can distinguish between two types of face analysis. Using a sequential approach, we may determine whether anything is a face based on facial traits, which are static. Is it a person? What age is it? Is it a man or a woman? Additionally, the facial state is dynamic. We can choose whether it is shocked or incensed Depending on the application, face emotion recognition systems may employ offline data or real-time input. In this paper, we're going to use a convolutional neural network to create and test a real-time facial recognition model. There is a lot of interest in this field, many people want to take part in Artificial Intelligence research, but they don't have the resources necessary to do so. Thus, we will lead them to Google Colab as a cloud-based solution without having to download or install anything on their hardware, as well as free GPU and TPU usage.

Introduction:

Machines mimicking human behavior is one of the scientists' favorite challenges. Using artificial neural networks, with their structures inspired by the human brain, it is possible to train models without requiring human intervention to recognize objects' features. This is the main reason people can start working on deep learning without being experts in data science. The joy of artificial intelligence can be experienced with a little knowledge and a few steps to follow as mentioned in Figure1.

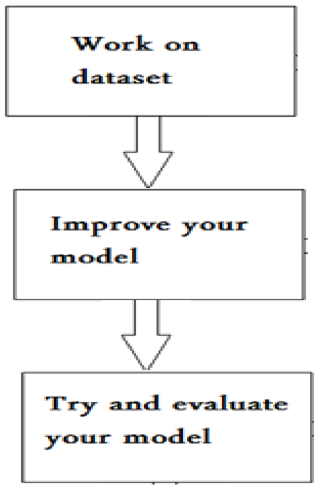


Figure 1:Steps of the work

Before starting, there is some basic acquittance to know.

Tensorflow: It's a python library for fast numerical computing. It's a foundation library created and released by google and used to build new deep learning models.

OpenCV: A python library that allows you to perform image processing and computer vision task. It's used for face detection and writing and drawing on images.

Keras: It's an application programming interface used for neural network implementation.

Pandas: It's an open-source library that provides fast and flexible data structures

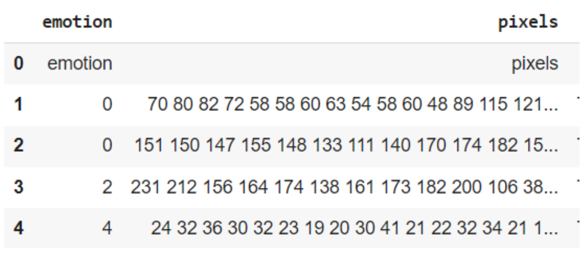
Numpy: It's a python library that adds support for large and multidimensional arrays and matrices

Figures list

I- Dataset Work:

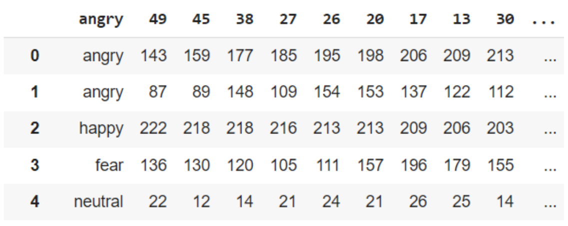
Datasets online are provided in many forms: They can be jpg images (Figure 4) or CSV files, and they can be split into two parts, for training and testing or joined in only one part.

Labels may be explicit(figure3) or implicit (Figure2). The most famous platform to find datasets is Kaggle. Kaggle datasets can even be directly imported into Colab.



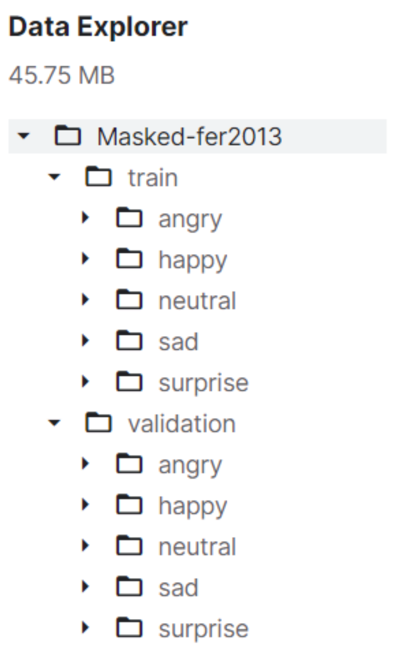
	emotion	pixels
0	emotion	pixels
1	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
2	0	151 150 147 155 148 133 111 140 170 174 182 15...
3	2	231 212 156 164 174 138 161 173 182 200 106 38...
4	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...

Figure 2: exapmle1 of the dataset



	angry	49	45	38	27	26	20	17	13	30	...
0	angry	143	159	177	185	195	198	206	209	213	...
1	angry	87	89	148	109	154	153	137	122	112	...
2	happy	222	218	218	216	213	213	209	206	203	...
3	fear	136	130	120	105	111	157	196	179	155	...
4	neutral	22	12	14	21	24	21	26	25	14	...

Figure 3:example2 of the dataset



Data Explorer

45.75 MB

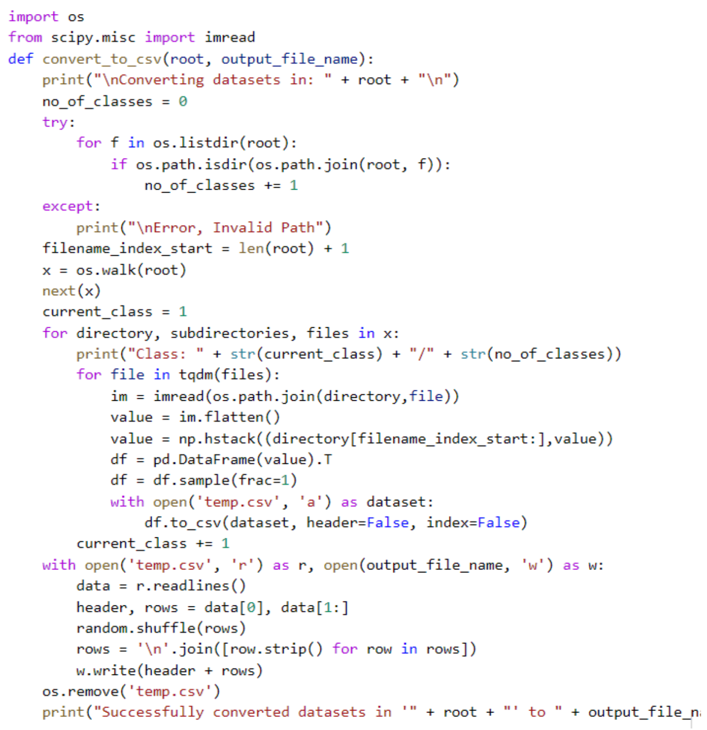
- Masked-fer2013
  - train
    - angry
    - happy
    - neutral
    - sad
    - surprise
  - validation
    - angry
    - happy
    - neutral
    - sad
    - surprise

Figure 4.example3 of the dataset

It's not important to pick the right dataset for the first time. What matters is to convert whatever data you have into a suitable input for your model. To achieve this conversion, we may face some problems.

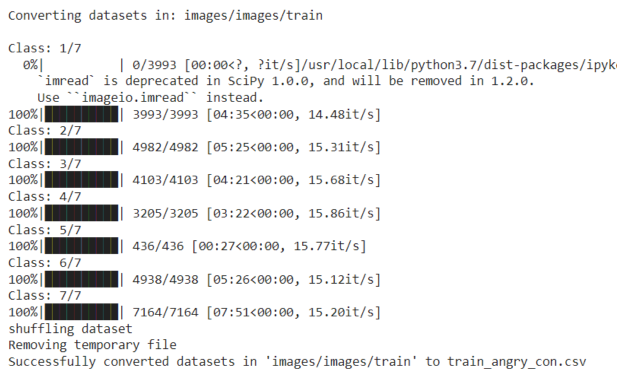
In the case of images dataset, from other websites, not colab You may like to convert all your images into a NumPy array. But uploading a whole image dataset to google Colab takes so much time, particularly because you can’t Sauvegarde upload files after runtime disconnection. It’s a waste of time to upload such a huge dataset every time you connect to google colab.

The best solution is to convert the image database to a CSV file. (Figure5 and Figure6).



```
import os
from scipy.misc import imread
def convert_to_csv(root, output_file_name):
    print("\nConverting datasets in: " + root + "\n")
    no_of_classes = 0
    try:
        for f in os.listdir(root):
            if os.path.isdir(os.path.join(root, f)):
                no_of_classes += 1
    except:
        print("\nError, Invalid Path")
    filename_index_start = len(root) + 1
    x = os.walk(root)
    next(x)
    current_class = 1
    for directory, subdirectories, files in x:
        print("Class: " + str(current_class) + "/" + str(no_of_classes))
        for file in tqdm(files):
            im = imread(os.path.join(directory,file))
            value = im.flatten()
            value = np.hstack((directory[filename_index_start:],value))
            df = pd.DataFrame(value).T
            df = df.sample(frac=1)
            with open('temp.csv', 'a') as dataset:
                df.to_csv(dataset, header=False, index=False)
        current_class += 1
    with open('temp.csv', 'r') as r, open(output_file_name, 'w') as w:
        data = r.readlines()
        header, rows = data[0], data[1:]
        random.shuffle(rows)
        rows = '\n'.join([row.strip() for row in rows])
        w.write(header + rows)
    os.remove('temp.csv')
    print("Successfully converted datasets in '" + root + "' to " + output_file_n
```

Figure 5: code used to convert a folder to a csv file

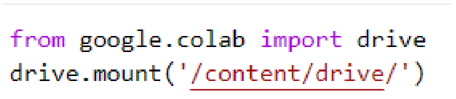


```
Converting datasets in: images/images/train

Class: 1/7
  0%|          | 0/3993 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipyk
    `imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
    Use ``imageio.imread`` instead.
100%|██████████| 3993/3993 [04:35<00:00, 14.48it/s]
Class: 2/7
100%|██████████| 4982/4982 [05:25<00:00, 15.31it/s]
Class: 3/7
100%|██████████| 4103/4103 [04:21<00:00, 15.68it/s]
Class: 4/7
100%|██████████| 3205/3205 [03:22<00:00, 15.86it/s]
Class: 5/7
100%|██████████| 436/436 [00:27<00:00, 15.77it/s]
Class: 6/7
100%|██████████| 4938/4938 [05:26<00:00, 15.12it/s]
Class: 7/7
100%|██████████| 7164/7164 [07:51<00:00, 15.20it/s]
shuffling dataset
Removing temporary file
Successfully converted datasets in 'images/images/train' to train_angry_con.csv
```

Figure 6.Converting a folder of jpg images into a csv file

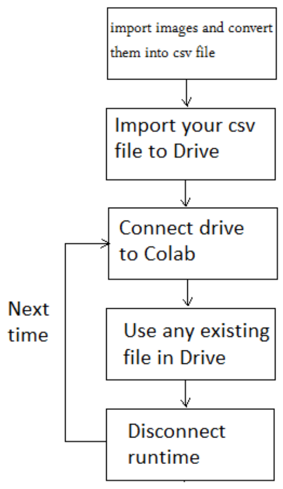
Since Colab and Drive are Google's products, it's easy to access Drive from Colab. You don't need to upload all your files every time. All you need is to connect your Colab session to your Drive. In Figure 8 you can find drive-colab collaboration facilitation.

Figure7 shows how to connect drive to colab.



```
from google.colab import drive
drive.mount('/content/drive/')
```

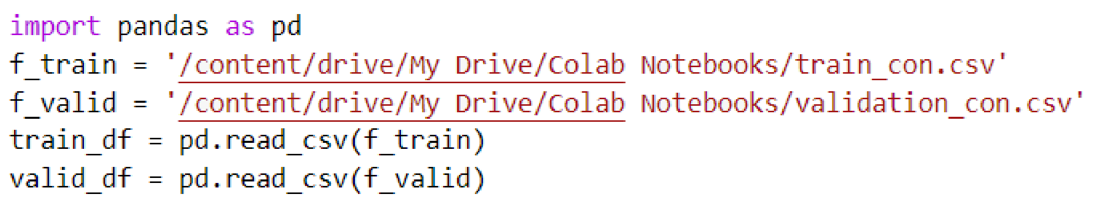
Figure 7:Connect drive to colab



import images and convert them into csv file
Import your csv file to Drive
Connect drive to Colab
Use any existing file in Drive
Disconnect runtime
Next time

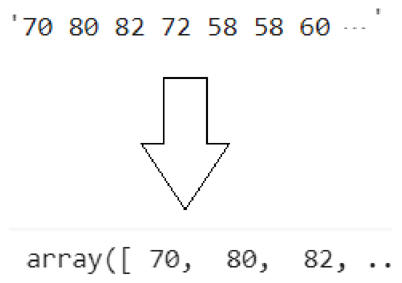
Figure 8:Facilitations of Drive-Colab collaboration

Pandas' library is used to get data frame from the CSV file (Figure9). When we upload the CSV files, every column of the data frame is a series, every case is a list. But, we can't do operations on lists or strings, we can't access directly an element of a list and lists use more memory. In many times, pixels are located in only one colon. Every line is a string containing pixels of an image. For each string of pixels, we can make it into a list of pixels using the split method then by applying an np.array() function and int() function we can get an array of int (Figure 10).



```
import pandas as pd
f_train = '/content/drive/My Drive/Colab Notebooks/train_con.csv'
f_valid = '/content/drive/My Drive/Colab Notebooks/validation_con.csv'
train_df = pd.read_csv(f_train)
valid_df = pd.read_csv(f_valid)
```

Figure 9: using pandas to get dataframe from csv file



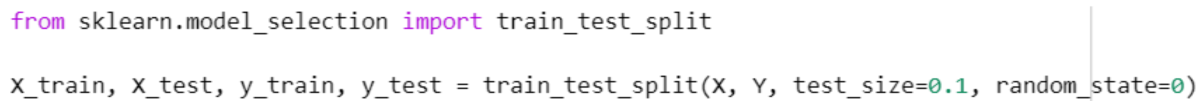
```
'70 80 82 72 58 58 60 ...'
```

```
array([ 70,  80,  82, ..
```

Figure 10:Convert list of Strings into array of int

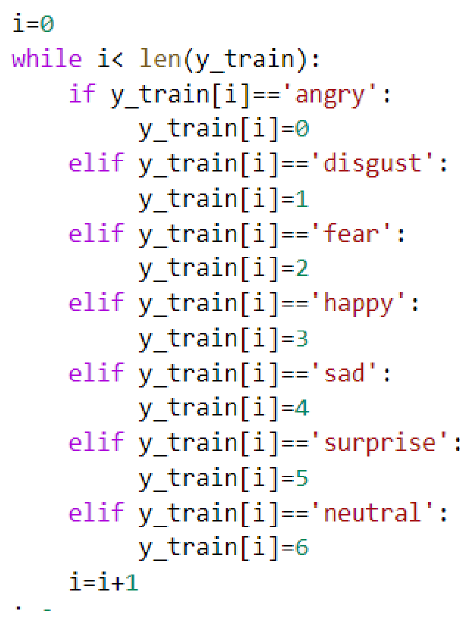
But we need to split our data into two groups (Figure11), the first one is for training, and the second is for tests.
For this purpose, you can import “train_test_split” from “sklearn.model_selection”. As an output for this function, we get: x_train: pixels of images used for training the neural network, y_train: Correct labels for the x_train, used to evaluate the model's predictions during training, x_test: Pixels of images set aside for validating the performance of the model after it has been trained, y_test: Correct labels for the x_test, used to evaluate the model's predictions after it has been trained

Furthermore, labels may be strings that are not practical in fitting models. In this case, you may only convert them into int to simplify your work (Figure12 and Figure13)



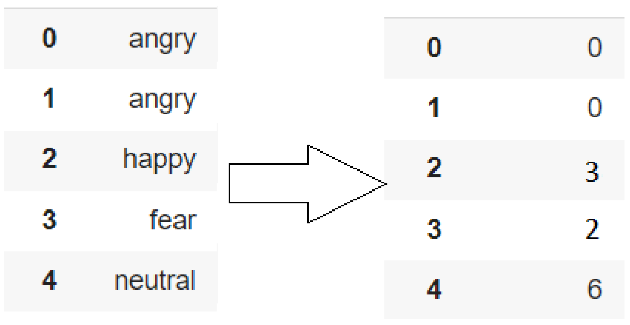
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=0)
```

Figure 11: split the data



```
i=0
while i< len(y_train):
    if y_train[i]=='angry':
         y_train[i]=0
    elif y_train[i]=='disgust':
         y_train[i]=1
    elif y_train[i]=='fear':
         y_train[i]=2
    elif y_train[i]=='happy':
         y_train[i]=3
    elif y_train[i]=='sad':
         y_train[i]=4
    elif y_train[i]=='surprise':
         y_train[i]=5
    elif y_train[i]=='neutral':
         y_train[i]=6
    i=i+1
```

Figure 12: Code to convert string labels into int



0	angry
1	angry
2	happy
3	fear
4	neutral

0	0
1	0
2	3
3	2
4	6

Figure 13:converting string labels into int

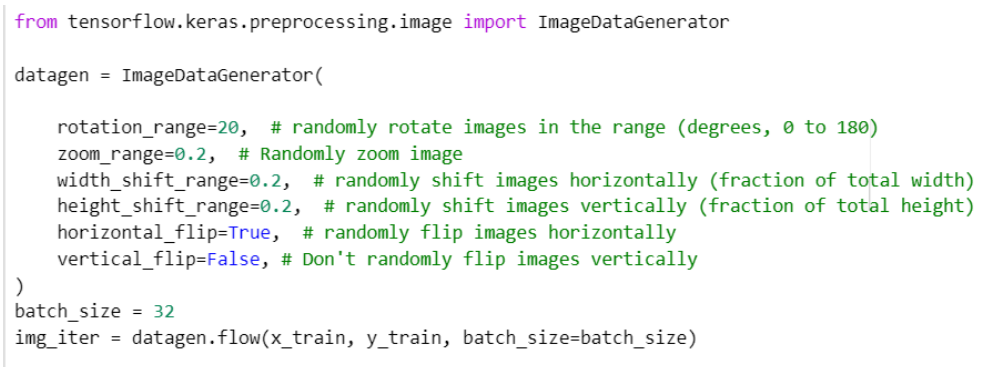
After getting 4 arrays, now we have to assure that our arrays are suitable. You need to pass by preprocessing.

- Normalization: to make the x_train and x_test values easier to work with for the model. By dividing by 255, we get all values between 0 and1.

-reshape: to put our data in an appropriate shape,

For example: (N,48,48,1) for N images of (48x48) pixels.

- Data augmentation: Deep Learning must be robust to tackle different images with increasing the amount of data by generating new data points from existing data
It's an imported function called ImageDataGenerator (Figure14) from tensorflow.keras.preprocessing.image that uses the rotation of images, zooming, and shift…some results are shown in Figure15.



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

    rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range=0.2,  # Randomly zoom image
    width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
    horizontal_flip=True,  # randomly flip images horizontally
    vertical_flip=False, # Don't randomly flip images vertically
)
batch_size = 32
img_iter = datagen.flow(x_train, y_train, batch_size=batch_size)
```

Figure 14: ImageDataGenerator function

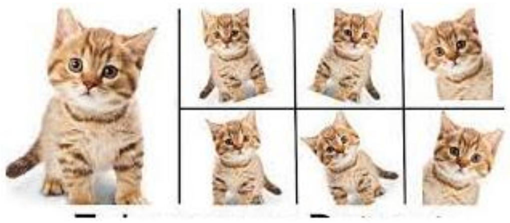


Figure 15:Example of data augmentation result

ImageDataGenerator is not the only way for data augmentation.

Since data augmentation is a solution for many problems such as insufficiency in the database, one-sided person, painted faces or cartoons, occlusion: covering the face with heads, eyeglasses… there are many other tools to increase data like GANS technology that mix two faces to get another one.

II- Improve your model

The model will be made up of several layers and will be comprised of 3 main parts:

-An input layer, it will receive data in a format that it expects.

-Several hidden layers, each comprised of many neurons. As the network learns from feedback and gets feedback on its performance, each neuron will be able to affect the network's guess with its weights.
-An output layer, this layer displays the network's guess for a particular image.

Some Layers:

-Conv2D: earlier convolution: detect simple features like lines, later convolution: complex features (Figure16)

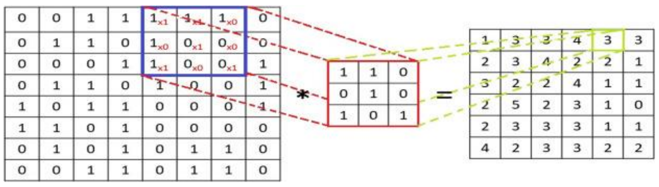


Figure 16: Convolutional layer

-Batch Normalization: to reduce internal covariate shift (imbalance in weight) (Figure17)

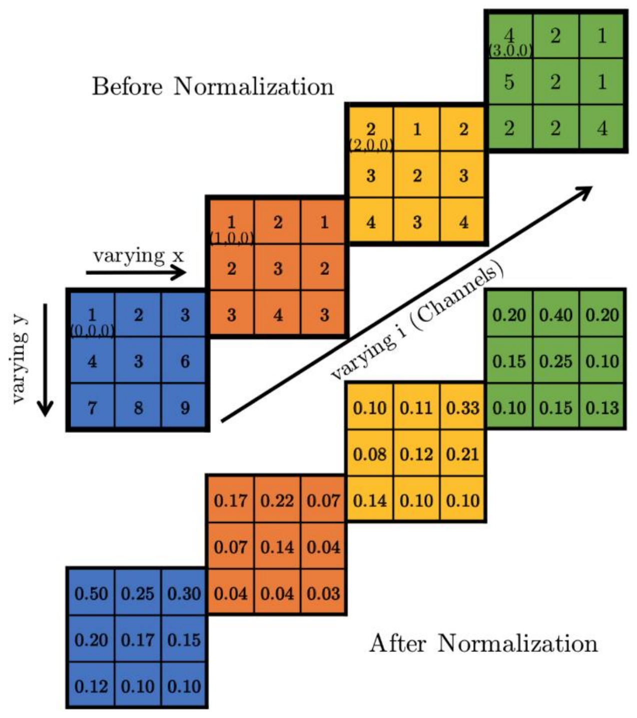


Figure 17: Normalization layer

-Max Pool2D: Shrink the image to a lower resolution-> help the model to be robust to translation of objects: it makes the model faster(Figure18).

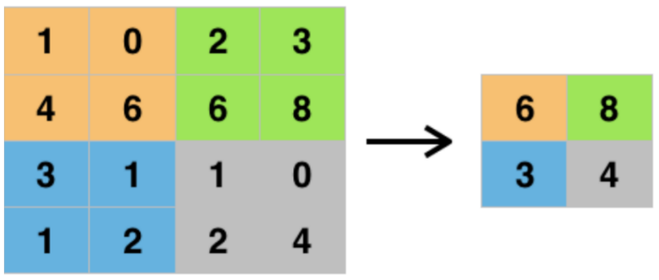


Figure 18: max pooling

-Dropout: prevent overfitting: select a subset of neurons and turn them off -> the network does not rely on any one area to come up with an answer

-Activation: defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. (Figures 19 and 20).

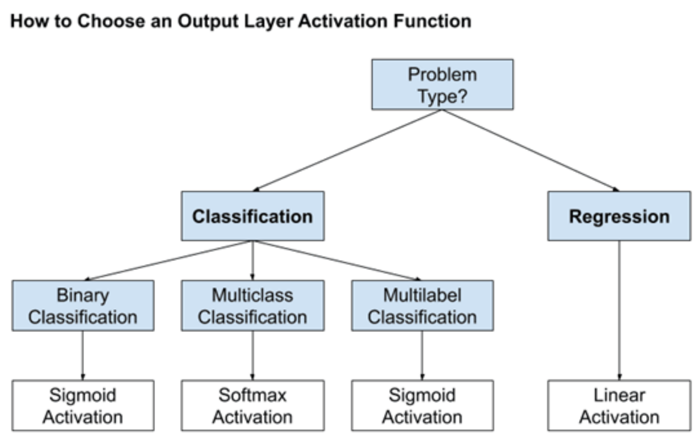


Figure 19:Activation function for the output layer

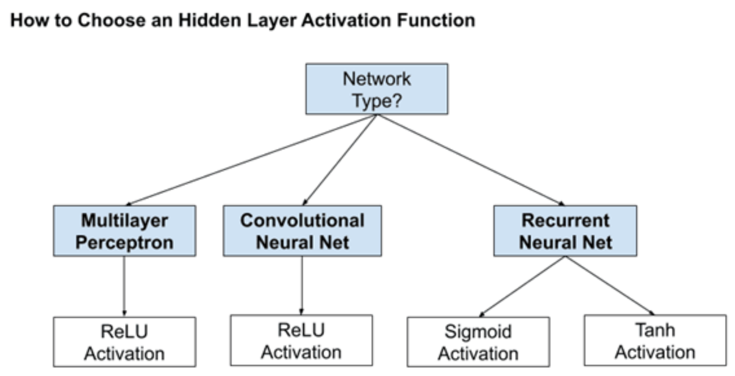
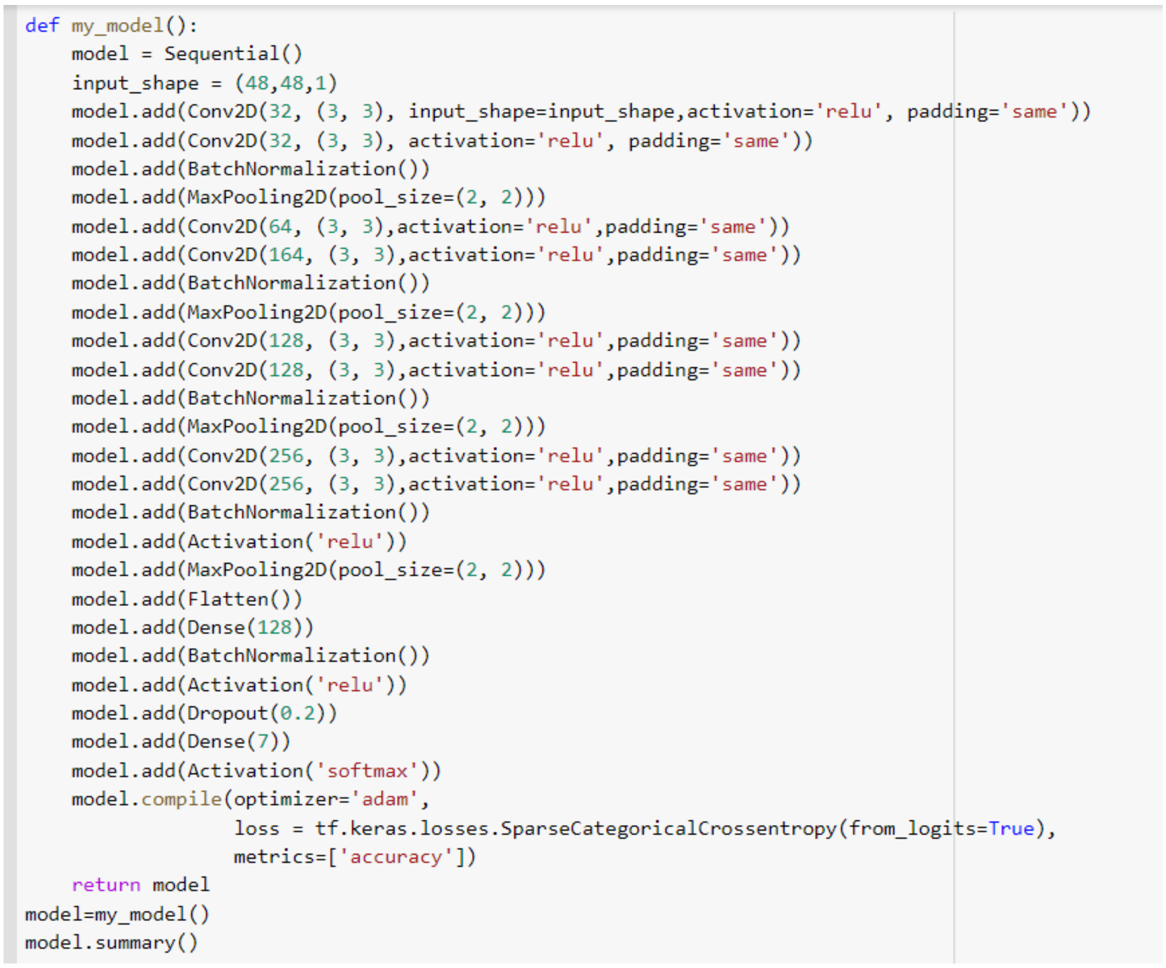


Figure 20:: activation function for the hidden layer

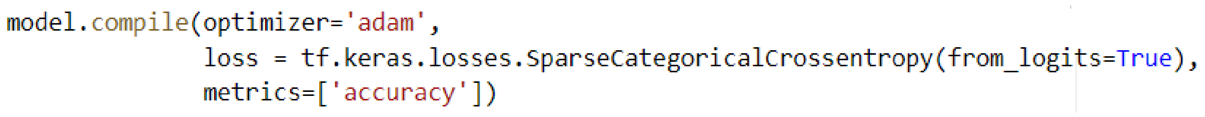
After choosing your layers (Figure21), you have to compile your model using a suitable optimizer (Figure22). A deep learning model needs to be trained by modifying each epoch's weights and minimizing its loss function. An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rates. As a result, it reduces overall loss and improves accuracy. Adam optimizer is one of the most famous optimizers. It updates the learning rate for each network weight individually. Also, the algorithm is straightforward to implement, has faster running time, low memory requirements, and requires less tuning than any other optimization algorithm.

Gradient Descent, AdaDelta, and RMSProp are also some considerable optimizers.



```
def my_model():
    model = Sequential()
    input_shape = (48,48,1)
    model.add(Conv2D(32, (3, 3), input_shape=input_shape,activation='relu', padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3),activation='relu',padding='same'))
    model.add(Conv2D(164, (3, 3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3),activation='relu',padding='same'))
    model.add(Conv2D(128, (3, 3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
    model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Dense(7))
    model.add(Activation('softmax'))
    model.compile(optimizer='adam',
                  loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
    return model
model=my_model()
model.summary()
```

Figure 21: Model definition function



```
model.compile(optimizer='adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 22: Compiling using adam optimizer

To fit your model, you have to set your learning rate, your batch size, and your epochs.(Figure24 and Figure25)

Epoch: it is defined as the number of times the algorithm runs on the entire training set.

Batch: it indicates how many samples will be taken for updating the model parameters.

Sample: it’s one row in a dataset.

Learning rate: This parameter determines how often model weights should be updated.

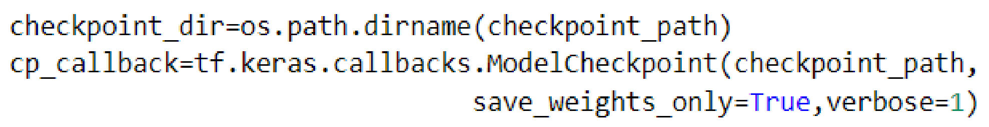
It’s very important to save the model weights in the checkpoint path then you are not obliged to fit your model every time you need it again.(Figure23)

If your training is taking so long, you should make sure your GPU is enabled:

Go to Runtime -> Change Runtime type -> Under Hardware accelerator select GPU

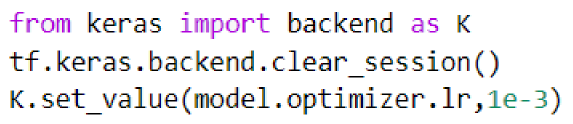
Based on the loss, the accuracy you may apply some modifications.

It's possible to have a better visualization by plotting the history of the outputs after every epoch (Figure26 and Figure27).



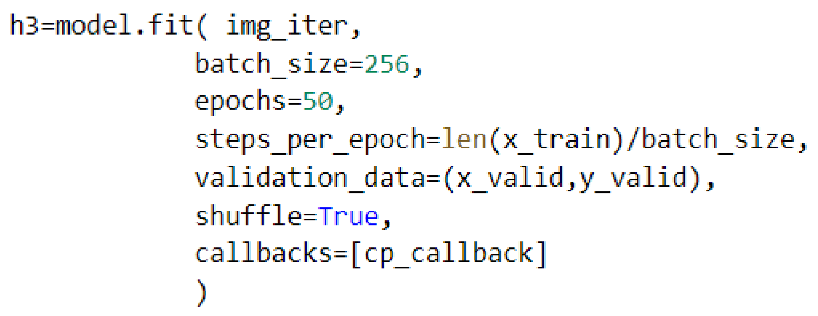
```
checkpoint_dir=os.path.dirname(checkpoint_path)
cp_callback=tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                               save_weights_only=True,verbose=1)
```

Figure 23: Save model weights



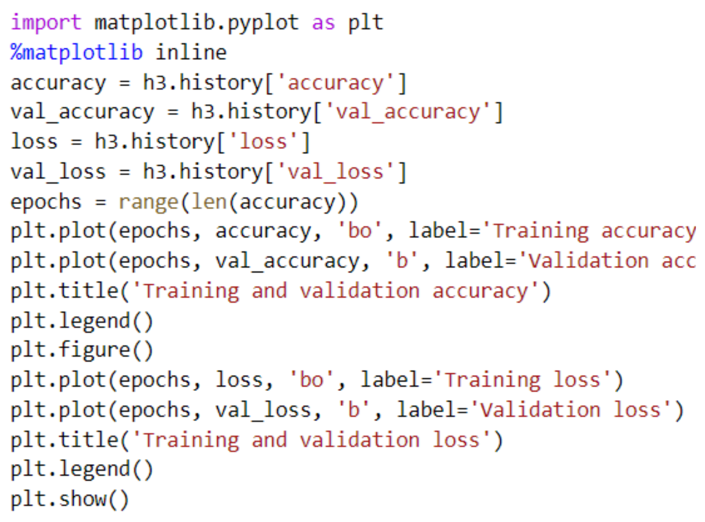
```
from keras import backend as K
tf.keras.backend.clear_session()
K.set_value(model.optimizer.lr,1e-3)
```

Figure 24: Setting learning_rate



```
h3=model.fit( img_iter,
                   batch_size=256,
                   epochs=50,
                   steps_per_epoch=len(x_train)/batch_size,
                   validation_data=(x_valid,y_valid),
                   shuffle=True,
                   callbacks=[cp_callback]
                   )
```

Figure 25: Fitting the model



```
import matplotlib.pyplot as plt
%matplotlib inline
accuracy = h3.history['accuracy']
val_accuracy = h3.history['val_accuracy']
loss = h3.history['loss']
val_loss = h3.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy
plt.plot(epochs, val_accuracy, 'b', label='Validation acc
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Figure 26:code to plot the history

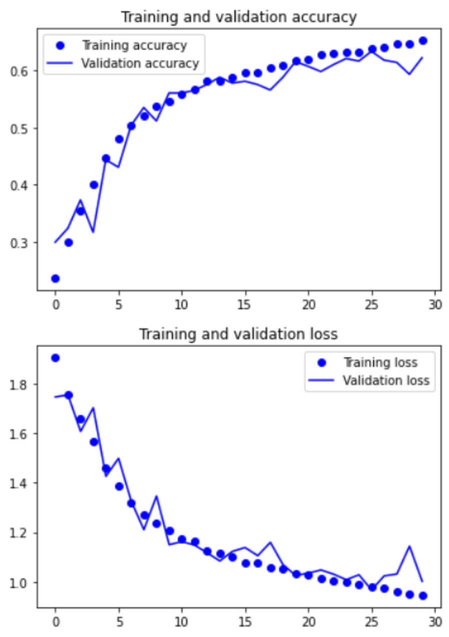
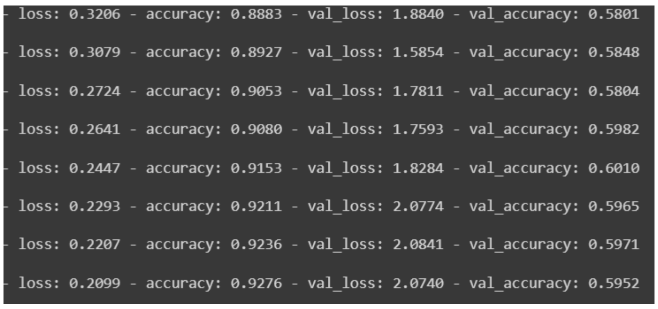


Figure 27plots of the outputs

You can face many problems before having a good accuracy:

<u>Overfitting:</u>

When the accuracy is very high compared to the validation accuracy. (figure28)



```
- loss: 0.3206 - accuracy: 0.8883 - val_loss: 1.8840 - val_accuracy: 0.5801
- loss: 0.3079 - accuracy: 0.8927 - val_loss: 1.5854 - val_accuracy: 0.5848
- loss: 0.2724 - accuracy: 0.9053 - val_loss: 1.7811 - val_accuracy: 0.5804
- loss: 0.2641 - accuracy: 0.9080 - val_loss: 1.7593 - val_accuracy: 0.5982
- loss: 0.2447 - accuracy: 0.9153 - val_loss: 1.8284 - val_accuracy: 0.6010
- loss: 0.2293 - accuracy: 0.9211 - val_loss: 2.0774 - val_accuracy: 0.5965
- loss: 0.2207 - accuracy: 0.9236 - val_loss: 2.0841 - val_accuracy: 0.5971
- loss: 0.2099 - accuracy: 0.9276 - val_loss: 2.0740 - val_accuracy: 0.5952
```

Figure 28:Overfitting

Proposed solutions:

-Add a Dropout layer or even increase its coefficient.

-Reduce the model complexity by reducing the size of the model: remove some layers or decrease the number of neurons

-Decrease the number of epochs

<u>Fluctuation:</u>

When the accuracy isn't stable(figure29)

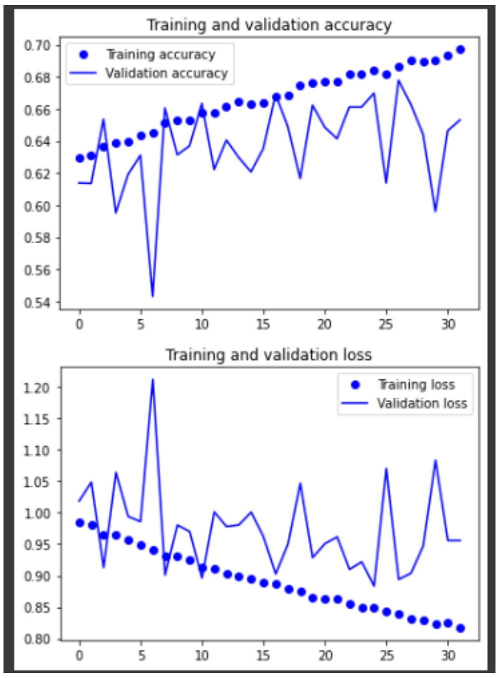


Figure 29: Fluctuations

Proposed solutions:

-Decrease learning rate

-Increase batch size

To analyze epochs and batch size influence on accuracy, we did many tests. Table1 shows the results of the tests.

Table 1: Accuracies results

	Epochs		
Batch size	Accuracy	50	100
	32	65.22%	59%(Overfitting)
	64	66.31%	66%(Overfitting)
	128	67.76%	66%(Overfitting)
	256	64.12%	67%(Overfitting)

After these measurements, we concluded that 100 epochs are the reason for overfitting., thus, many epochs lead to the overfitting of our model for the training images. And from a batch size of 256, accuracy starts decreasing.

III- Trying and evaluate your model

The efficiency is calculated based on your test database. But the model can only accept faces as input, you can't expect it to predict some facial expression from a random picture or a real-time video. As a solution, you can use OpenCV to detect the face from any image, and then you can try to predict the emotion of the detected face as explained in Figure 30. You can even draw a rectangle and write the expression on the picture (Figure31).

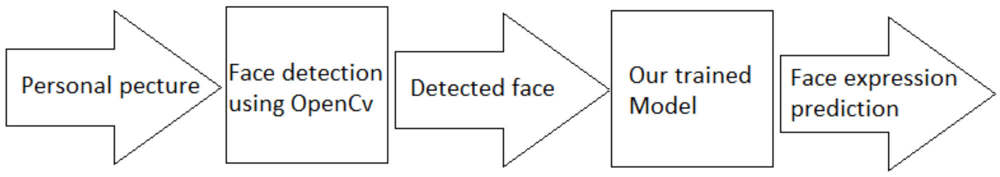


Figure 30: sequential approach for face expression prediction

```
plt.imshow(cv2.cvtColor(frame,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fafef8f0690>
```

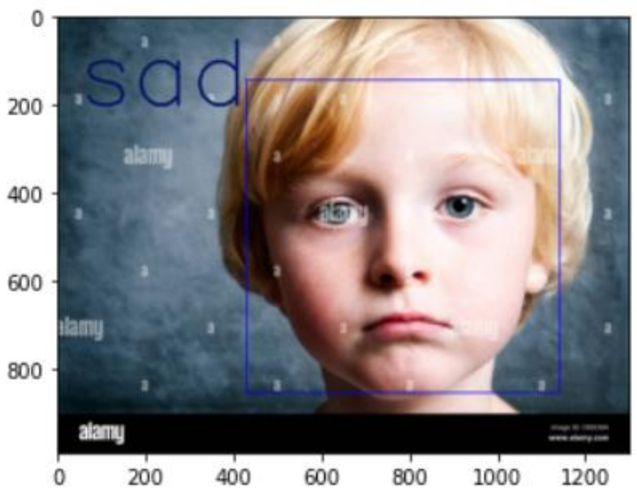


Figure 31:predict facial emotion from a random picture

Not to mention that the prediction should be applied to a suitable input for the model., That's why you need to convert your image into a NumPy array (Figure32)

```
img =tf.keras.utils.load_img('/content/drive/My Drive/Colab Notebooks/happy.jpg',
                             grayscale=True, target_size=(48, 48))
x = tf.keras.preprocessing.image.img_to_array(img)
x=x.astype(float)
x= x.reshape(1,48,48,1)
x=x/255
prediction = model.predict(x)
print(prediction)
```

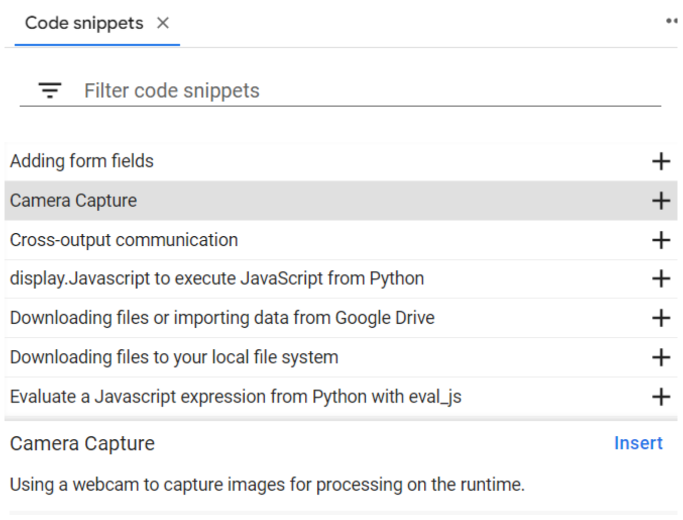
Figure 32: Personal picture prediction

To try your model after your run time disconnection, you can load weights already saved (Figure33)

```
model.load_weights(checkpoint_path)
```

Figure 33: Load weights

Also, to experience a reel time emotion detection, Colab allows you to have a video stream from your local webcam. Code snippets (Figure34) are a good option to add predefined codes:


Code snippets
Filter code snippets
Adding form fields
Camera Capture
Cross-output communication
display.Javascript to execute JavaScript from Python
Downloading files or importing data from Google Drive
Downloading files to your local file system
Evaluate a Javascript expression from Python with eval_js
Camera Capture
Insert

Using a webcam to capture images for processing on the runtime.

Figure 34:Code snippets for camera capture

Conclusion:

To recognize facial expressions, Google Colab is suggested in this study. After training various models, we were able to increase accuracy by making changes to the datasets, choosing the optimum batch size and epochs and modifying various layers.

References:

[1] Panarav KB and Manikandan "Design and Evaluation of a Real-Time Face Recognition System using Convolutional neural Network" third international conference on computing and Network Communication (CoCoNet'19)

[2] Siyue Xie and Haifeng Hu "Facial Expression Recognition Using Hierarchical

Features With Deep Comprehensive Multipatches Aggregation Convolutional Neural Networks” IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 21, NO. 1, JANUARY 2019

[3]https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

[4]https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[5]https://medium.com/@ml_kid/how-to-save-our-model-to-google-drive-and-reuse-it-2c1028058cb2

[6]https://towardsdatascience.com/4-reasons-why-you-should-use-google-colab-for-your-next-project-b0c4aaad39ed

[7]https://www.geeksforgeeks.org/

[8]https://stackoverflow.com/

[9]https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/

[10]https://iecscience.org/jpapers/46

[11]https://www.freecodecamp.org/news/improve-image-recognition-model-accuracy-with-these-hacks/

[12]https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

[13]https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/

[14]https://datascience.stackexchange.com/questions/55545/in-cnn-why-do-we-increase-the-number-of-filters-in-deeper-convolution-layers-fo

[15]https://datascience.stackexchange.com/search?q=improve+accuracy

[16]https://datascience.stackexchange.com/questions/102629/how-to-improve-accuracy-bert