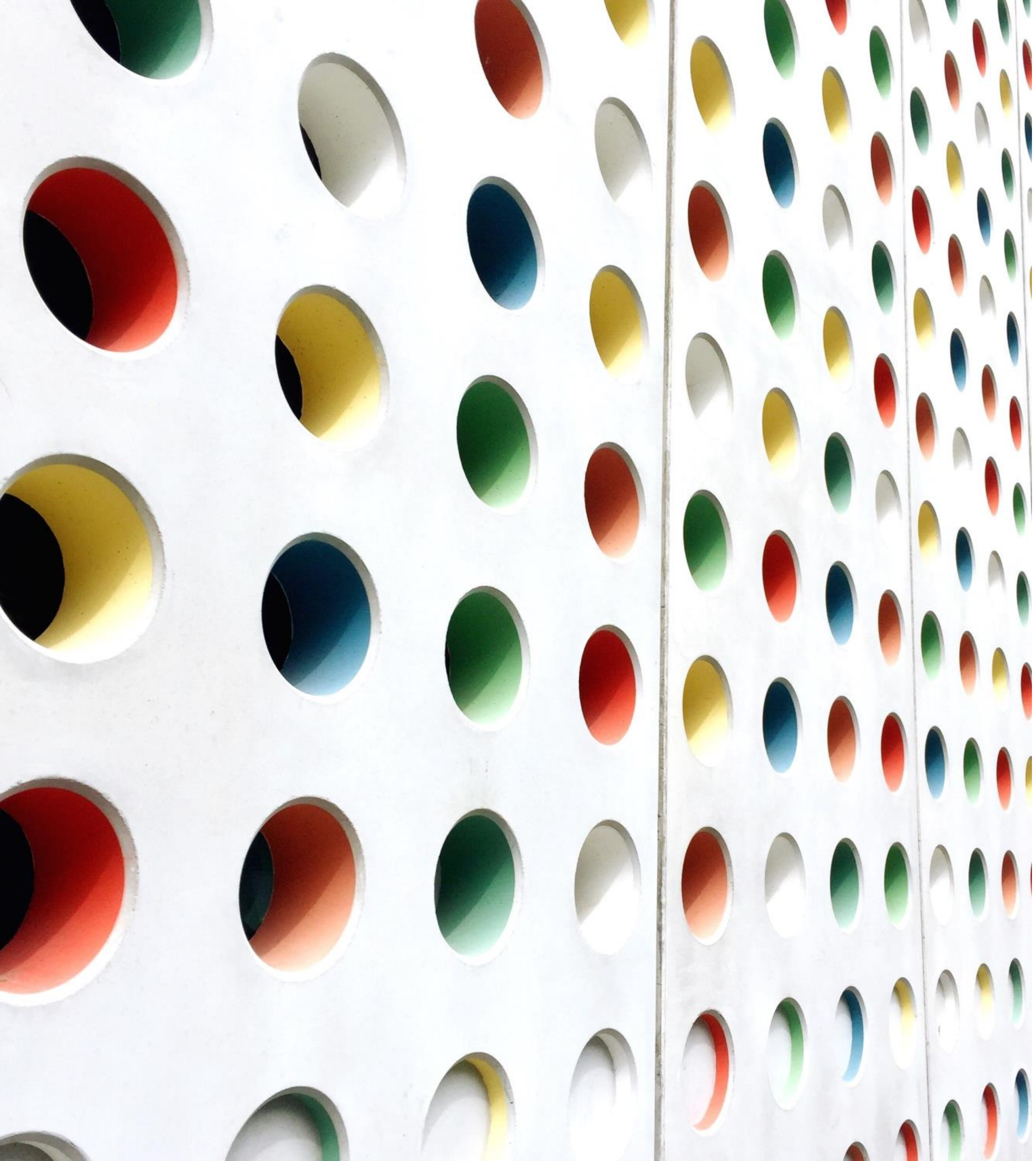


Computer Vision

FACIAL EXPRESSION RECOGNITION CHALLENGE

Submitted by
Ramya Mercy Rajan
AA.SC.P2MCA2107434
MCA in Artificial Intelligence
Amrita Vishwa Vidyapeetham

Project Summary

This case study aims to develop a face emotion recognition detector using computer vision techniques. Facial expression for emotion detection has always been an easy task for humans, but achieving the same task with a computer algorithm is quite challenging. With the recent advancement in computer vision and machine learning, it is possible to detect emotions from images. This case study is focussed on facial emotion recognition using convolutional neural networks. This requires a camera to take picture of our expression in order to detect.

In this project, I have used dataset from Kaggle for a case study and then by the end of the project a live facial expression is shown to detect the kind of expression using webcam.

This dataset consists of **48x48 pixel grayscale images of faces**.

The **training set consists of 28,709** examples and the public **test set consists of 3,589** examples. In this project, I have focussed on **building and training a convolutional neural network (CNN) in Keras** to recognize facial expressions. The **objective is to classify each face based on the emotion** shown in the facial expression into one of **seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).**

OpenCV is used to automatically detect faces in images and draw bounding boxes around them. Once the model trained, saved, and exported the CNN, the trained model predictions is directly served to a web interface and perform real-time facial expression recognition on video.

Methodology

1. Performing EDA and importing necessary libraries. Import essential modules and helper functions from NumPy, Matplotlib, and Keras. Checked for class imbalance problems in the training data.
2. Generate Training and Validation Batches- Generate batches of tensor image data with real-time data augmentation. Specified paths to training and validation image directories and generates batches of augmented data.
3. Created a Convolutional Neural Network (CNN) Model-Designed a convolutional neural network with 4 convolution layers and 2 fully connected layers to predict 7 types of facial expressions. Used Adam as the optimizer, categorical crossentropy as the loss function, and accuracy as the evaluation metrics.
4. Train and Evaluate Model -Train the CNN by invoking the model.fit() method. Use ModelCheckpoint() to save the weights associated with the higher validation accuracy. Observing training loss and accuracy plots in Jupyter Notebook for Keras.
5. Save and Serialize Model as JSON String - Used to_ json(), which uses a JSON string, to store the model architecture.
6. Create a Flask App to Serve Predictions

CODING

Task 1: Import Libraries

In [1]:
```
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.inputs.tf_keras import PlotLossesCallback
import tensorflow as tf
print("Tensorflow version:", tf.__version__)
```

```
Tensorflow version: 2.7.0
```

```
In [4]: for expression in os.listdir("train/"):
            print(str(len(os.listdir("train/" + expression))) + " " + expression + "images")
```

```
3995 angryimages
436 disgustimages
4097 fearimages
7215 happyimages
4965 neutralimages
4830 sadimages
3171 surpriseimages
```

we have relatively imbalance dataset except for the disgust class images

Task 2: Generate Training and Validation Batches

By using data generators and the ImageDataGenerator class, we can efficiently load and augment images in batches during the training and evaluation processes. Data augmentation helps increase the variety and generalization of the training data by applying random transformations to the images. This can improve the model's ability to generalize and perform well on unseen data.

In [5]:

```
img_size = 48
batch_size = 64
datagen_train = ImageDataGenerator(horizontal_flip = True)

train_generator = datagen_train.flow_from_directory( "train/" ,
                                                    target_size=(img_size, img_size),
                                                    color_mode= "grayscale",
                                                    batch_size= batch_size,
                                                    class_mode="categorical",
                                                    shuffle = True)
datagen_validation = ImageDataGenerator(horizontal_flip = True)
validation_generator = datagen_validation.flow_from_directory( "test/",
                                                    target_size=(img_size, img_size),
                                                    color_mode= "grayscale",
                                                    batch_size= batch_size,
                                                    class_mode="categorical",
                                                    shuffle = True)
```

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

Task 3: Create CNN Model

Defines a CNN model with convolutional, pooling, dropout, and fully connected layers. It follows a common pattern for image classification tasks. Adjust the model architecture and hyperparameters as needed for facial expression recognition

Applying multiple convolutional layers helps the model learn increasingly complex and abstract features as the information flows deeper into the network. The initial layers capture low-level features like edges and textures, while the deeper layers capture higher-level features such as shapes, patterns, and facial expressions.

In [6]:

```
model=Sequential()

#1 - conv

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))

#2 -conv

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))

#3 - conv

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))
```

```
#4 - conv

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))

model.add(Flatten())

model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax'))


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

model.summary() #displays a summary of the model architecture
```

Model: "sequential"

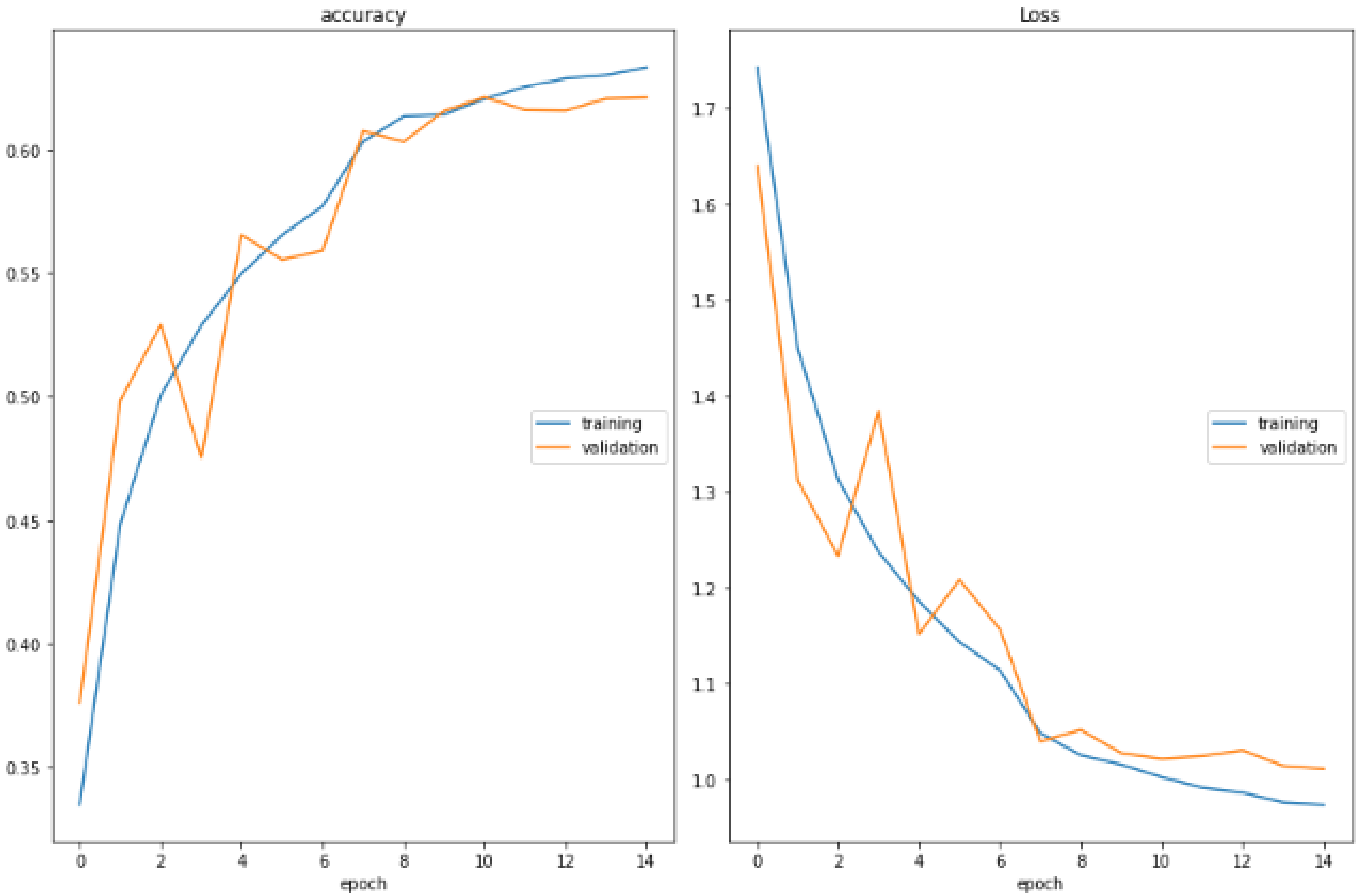
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 64)	640
batch_normalization (BatchNormalization)	(None, 48, 48, 64)	256
activation (Activation)	(None, 48, 48, 64)	0
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0
conv2d_1 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 128)	512
activation_1 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_1 (Dropout)	(None, 12, 12, 128)	0
conv2d_2 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 512)	2048
activation_2 (Activation)	(None, 12, 12, 512)	0
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 512)	0
dropout_2 (Dropout)	(None, 6, 6, 512)	0
conv2d_3 (Conv2D)	(None, 6, 6, 128)	1638528
batch_normalization_3 (BatchNormalization)	(None, 6, 6, 128)	512
activation_3 (Activation)	(None, 6, 6, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 128)	0
dropout_3 (Dropout)	(None, 3, 3, 128)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 256)	295168
batch_normalization_4 (BatchNormalization)	(None, 256)	1024

```
 activation_4 (Activation)   (None, 256)               0

 dropout_4 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 2,735,751
Trainable params: 2,733,575
Non-trainable params: 2,176
_________________________________________________________________
```

Total params: 2,735,751 Trainable params: 2,733,575 (this will be updated during traing) Non-trainable params: 2,176 (this will not be updated during training)

Task 4: Train and Evaluate Model

In [7]:
```
epochs=15
steps_per_epoch= train_generator.n//train_generator.batch_size
validation_steps= validation_generator.n//validation_generator.batch_size

checkpoint=ModelCheckpoint("model_weights.h5", monitor="val_accuracy",
                            save_weights_only=True, mode='max', verbose=1)

reduce_learning_rate= ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, min_learning_rate=0.00001, model='auto')


callbacks = [PlotLossesCallback(), checkpoint, reduce_learning_rate]


history = model.fit(x=train_generator,steps_per_epoch= steps_per_epoch,epochs=epochs,validation_data=validation_generator,validat
```


accuracy
0.60
0.55
0.50
0.45
0.40
0.35
0
2
4
6
8
10
12
14
epoch
training
validation
Loss
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
2
4
6
8
10
12
14
epoch
training
validation

```
accuracy
	training	(min:	0.335, max:	0.633, cur:	0.633)
	validation	(min:	0.376, max:	0.621, cur:	0.621)
Loss
	training	(min:	0.974, max:	1.743, cur:	0.974)
	validation	(min:	1.011, max:	1.640, cur:	1.011)

Epoch 00015: saving model to model_weights.h5
448/448 [==============================] - 1495s 3s/step - loss: 0.9740 - accuracy: 0.6333 - val_loss: 1.0115 - val_accuracy:
0.6210 - lr: 5.0000e-06
```

the training accuracy is 63.33% (0.633) and the validation accuracy is 62.10% (0.621).

the training and validation accuracy values are 63.33% and 62.10% respectively, which are relatively close.

This indicates that the model is performing similarly on both the training and validation data.

lower loss values indicate better performance

the training loss is 0.974 and the validation loss is 1.011. Comparing these values, we can see that the training loss is

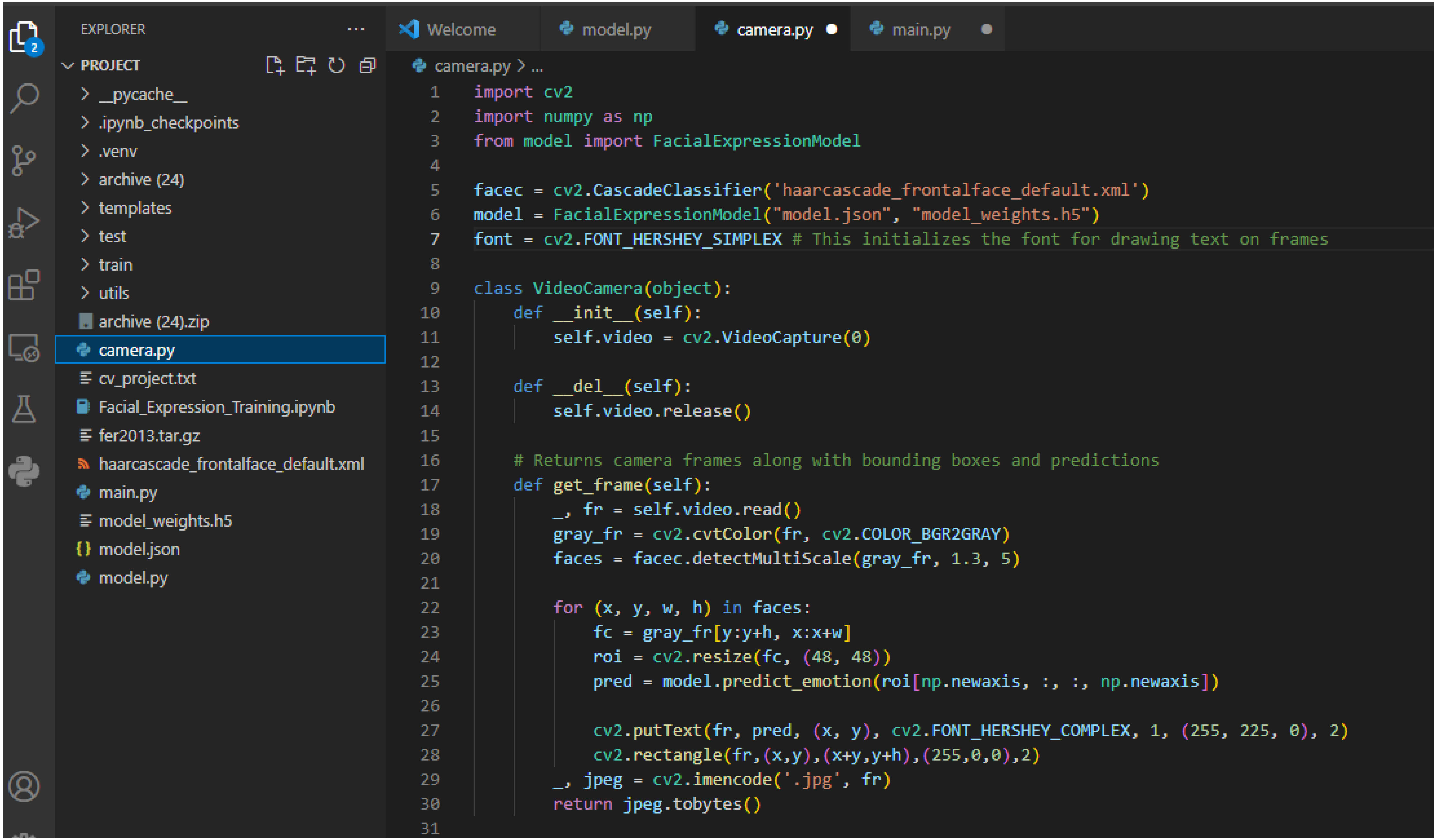
lower than the validation loss,

which suggests that the model is performing relatively better on the training data compared to the validation data.

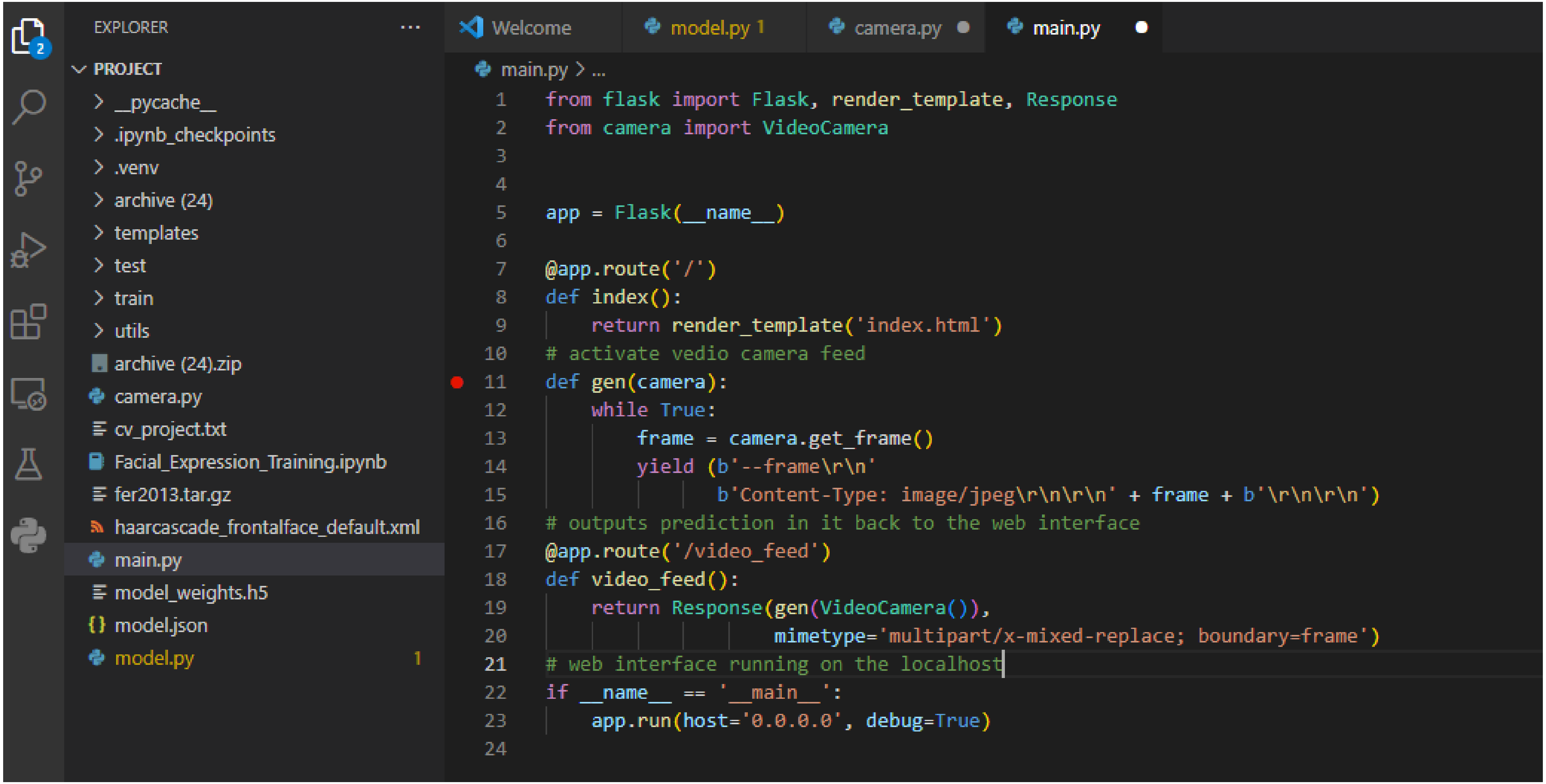
Task 5: Represent Model as JSON String

Saves the model architecture (in JSON format) to a file named "model.json"

In [9]:
```
model_json = model.to_json()
with open("model.json","w") as json_file:
    json_file.write(model_json)
```


EXPLORER
PROJECT
__pycache__
.ipynb_checkpoints
.venv
archive (24)
templates
test
train
utils
archive (24).zip
camera.py
cv_project.txt
Facial_Expression_Training.ipynb
fer2013.tar.gz
haarcascade_frontalface_default.xml
main.py
model_weights.h5
model.json
model.py
Welcome
model.py
camera.py
main.py
camera.py > ...
import cv2
import numpy as np
from model import FacialExpressionModel
facec = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
model = FacialExpressionModel("model.json", "model_weights.h5")
font = cv2.FONT_HERSHEY_SIMPLEX # This initializes the font for drawing text on frames
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), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 225, 0), 2)
cv2.rectangle(fr,(x,y),(x+y,y+h),(255,0,0),2)
_, jpeg = cv2.imencode('.jpg', fr)
return jpeg.tobytes()

```python

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

config = tf.compat.v1.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.15
session = tf.compat.v1.Session(config=config)

class FacialExpressionModel(object):
    EMOTIONS_LIST = ["Angry", "Disgust", "Fear", "Happy", "Neutral", "Sad", "Surprise"]

    def __init__(self, model_json_file, model_weights_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)

        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)]

```


EXPLORER
PROJECT
__pycache__
.ipynb_checkpoints
.venv
archive (24)
templates
test
train
utils
archive (24).zip
camera.py
cv_project.txt
Facial_Expression_Training.ipynb
fer2013.tar.gz
haarcascade_frontalface_default.xml
main.py
model_weights.h5
model.json
model.py 1
Welcome
model.py 1
camera.py
main.py
main.py > ...
from flask import Flask, render_template, Response
from camera import VideoCamera
app = Flask(__name__)
@app.route('/')
def index():
return render_template('index.html')
# activate vedio camera feed
def gen(camera):
while True:
frame = camera.get_frame()
yield (b'--frame\r\n'
b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
outputs prediction in it back to the web interface
@app.route('/video_feed')
def video_feed():
return Response(gen(VideoCamera()),
mimetype='multipart/x-mixed-replace; boundary=frame')
# web interface running on the localhost
if __name__ == '__main__':
app.run(host='0.0.0.0', debug=True)