Python AI Project
Facial Emotion Recognition using Django and VGG16 Model

By Elie araye

Introduction

- For this project, I have used the VGG16 model, a convolutional neural network that has shown impressive results in image recognition tasks. I have also utilized the Django web framework, which provides a simple and effective way to build web applications.
- The goal of this project is to develop a facial emotion recognition system that can accurately identify emotions from facial images in real-time. In this presentation, I will walk you through the different components of the project, from the VGG16 model to the Django web interface, and show you a live demonstration of the system in action.

The purpose of the VGG16 model is to classify images into various categories. In our project, we have used the VGG16 model to classify facial images into six basic emotions: happy, sad, angry, fearful, surprised, and disgusted.

One of the advantages of using the VGG16 model is that it has been trained on a large and diverse dataset, which means that it can accurately classify a wide range of images. Additionally, the VGG16 model has been shown to be effective in transfer learning, which is the process of using pre-trained models for new tasks. This makes it a popular choice for many computer vision projects, including facial emotion recognition.

To train the model I have used the following dataset:

https://www.kaggle.com/aadityasinghal/facial-expression-dataset

In [1]:
```
import os
```

In [2]:
```
def mkdir(p):
   if not os.path.exists(p):
        os.mkdir(p)

def link(src,dst):
    if not os.path.exists(dst):
        os.symlink(src,dst,target_is_directory=True)
```

In [3]:
```
classes=[
    'angry',
    'disgust',
    'fear',
    'happy',
    'neutral',
    'sad',
    'surprise'
]
```

In [4]:
```
train_path_from = os.path.abspath('C:/Users/Elie/Downloads/facial_emotions/train')
```

In [5]:
```
valid_path_from = os.path.abspath('C:/Users/Elie/Downloads/facial_emotions/test')
```

In [6]:
```
train_path_to = os.path.abspath('C:/Users/Elie/Downloads/facial_emotions/training')
valid_path_to = os.path.abspath('C:/Users/Elie/Downloads/facial_emotions/validation')
```

In [7]:
```
mkdir(train_path_to)
mkdir(valid_path_to)
```

Initializing the classes and loading the dataset

```
In [16]: model = VGG16(input_shape=IMAGE_SIZE +[3],
                 weights="imagenet", include_top = False)
```

In this code, I am creating an instance of the VGG16 model using the Keras library.

input_shape parameter specifies the size of the input image to the model. Here, we are using the IMAGE_SIZE variable that holds the dimensions of the input image. We add [3] to the dimensions to specify that the image has 3 channels (RGB).

weights parameter is set to "imagenet", which means that we are initializing the model with pre-trained weights from the ImageNet dataset. ImageNet is a large dataset of images used for training computer vision models.

```
In [19]: x = Flatten()(model.output)

In [20]: prediction = Dense(len(folders), activation = 'softmax')(x)

In [21]: len(folders)

Out[21]: 7

In [22]: model.input

Out[22]: <KerasTensor: shape=(None, 100, 100, 3) dtype=float32 (created by layer 'input_1')>

In [23]: model = Model(inputs= model.input, outputs=prediction)
```

In this code, we are building a deep learning model for facial emotion recognition using the VGG16 convolutional neural network architecture.

The **`Flatten()`** layer is added to convert the 3D output tensor of the VGG16 convolutional base model to a 1D feature vector.

Then, we add a densely connected classifier layer with **`len(folders)`** units (i.e., one unit for each class in the classification task) and apply the softmax activation function to the output of this layer to get the final classification probabilities.

We then create a new Keras model by specifying the input tensor as **`model.input`** and the output tensor as the output of the dense layer that we created earlier. This new model will have the same architecture as the original VGG16 model, with the addition of the flattened layer and the dense layer.

```python
In [25]: model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

Configuring the model for training by specifying the loss function, optimizer, and evaluation metrics.

The loss parameter is set to 'categorical_crossentropy'.

In [33]:
```
train_generator = dataset_generator.flow_from_directory(
    TRAIN_PATH,
    target_size=IMAGE_SIZE,
    shuffle=True,
    batch_size = BATCH_SIZE,
    class_mode="categorical")
```

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

In [34]:
```
valid_generator = dataset_generator.flow_from_directory(
    TEST_PATH,
    target_size=IMAGE_SIZE,
    shuffle=True,
    batch_size = BATCH_SIZE,
    class_mode="categorical")
```

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

In [37]:
```
history = model.fit(train_generator,
                    validation_data = valid_generator,
                    epochs= EPOCHS,
                    steps_per_epoch= len(train_image_files)//BATCH_SIZE,
                    validation_steps = len(valid_image_files)//BATCH_SIZE)
```

`train_generator` and `valid_generator` are created using the `flow_from_directory()` method by passing the directory path, target size of images, batch size, and class mode (which is set to categorical as we have multiple classes to predict).

The `fit()` method to train the model using the `train_generator` and `valid_generator` as the input.

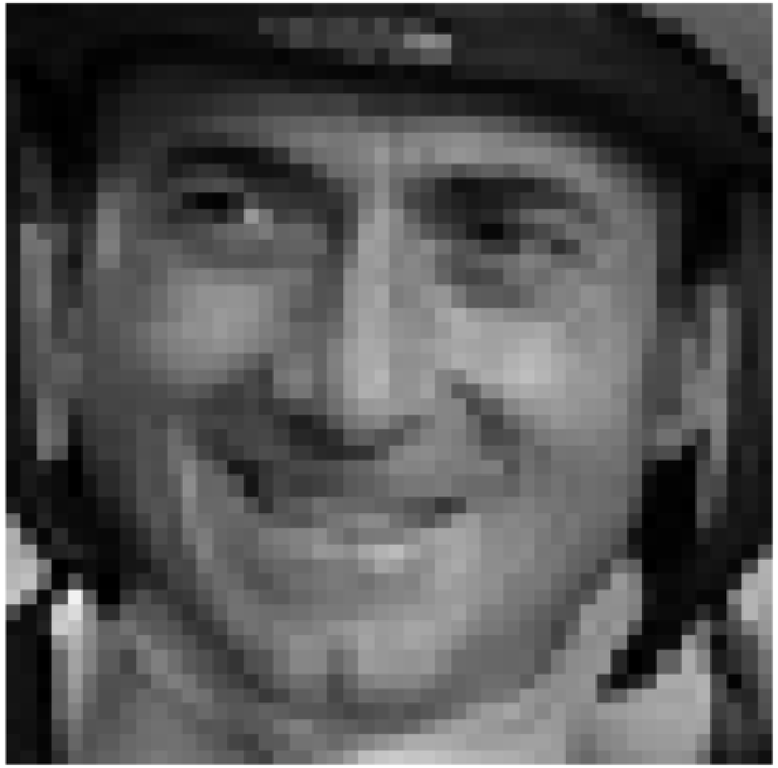
The number of epochs specified here is 5, which is much lower than it should be. However, I have used this value as training with a higher number of epochs takes a considerable amount of time,

which will cause a lower accuracy number compared to what could have been achieved with a higher number of epochs. Therefore, in order to achieve better accuracy, it is recommended to train the model with a higher number of epochs, even though it would require more time.

In [62]:

```
original_label_index = valid_gen.classes[valid_gen.filepaths.index(img_path)]
original_label = labels[original_label_index]
plt.imshow(img)
plt.title(f"Predicted label: {predicted_label}\nOriginal label: {original_label}")
plt.axis("off")
plt.show()
```

Predicted label: happy
Original label: happy



Testing the model with a random image

```python
In [81]: #saving this model in django

         model.save('C:/Users/Elie/djangoAi/djangoAi/savedModels')
```

Saving the trained model in the Django project.
This saved model can later be loaded and used to make predictions on new images.

The first step in our Django project was to create an HTML page where the user can upload a selfie. Once uploaded, our application uses the trained model to analyze the selfie and determine the user's emotional state. This information is then displayed back to the user on the same page.

```
<html>
<head>
    <title>Upload file</title>
</head>
<container>


  <body style ="align-items: center;justify-content: center;display: flex;">
    <h2>Upload a selfie to see how you look toady!</h2>
      {% if message %}
          <p>{{predicted}}</p>
      {% endif %}
      <form method="post" enctype="multipart/form-data">
          {% csrf_token %}
          <input type="file" name="file">
          <button type="submit" value="Upload" style=" display: inline-block; padding:
          text-align: center;
          text-decoration: none;
          background-color: #4CAF50;
          color: #fff;
          border-radius: 4px;
          transition: background-color 0.3s ease-in-out;">Upload</button>
      </form>

      {% if message %}
        <img  style="width: 200px; height: 150px;" src="{{ MEDIA_URL }}/{{link}}">
      {% endif %}

  </body>
</container>
</html>
```

In `views.py`, after the user uploads an image, the first step is to move the image to the `media` folder located within the same project.

```
def upload(request):
    if request.method == 'POST' and request.FILES['file']:
        # get the uploaded file
        uploaded_file = request.FILES['file']
        image_name = uploaded_file.name
        # create the path where the file will be saved
        file_path = os.path.join(settings.MEDIA_ROOT, uploaded_file.name)

        file_path = file_path.replace('\\', '/')

        image_url = request.build_absolute_uri(file_path)

        # write the file to the media directory
        with open(file_path, 'wb+') as destination:
            for chunk in uploaded_file.chunks():
                destination.write(chunk)
```

Then I load the model I saved earlier, the uploaded image is opened using the Image module and resized to 100x100 pixels. It is also converted to RGB format so to match the model input shape (100, 100, 3)

The image is then converted to a numpy array and an additional dimension is added to the array to match the input shape of the model.

The model's predict method is then used to predict the emotion in the preprocessed image. The predicted label index is obtained using np.argmax.

Based on the predicted label index, a message is generated indicating the predicted emotion. The message is stored in the variable 'msg'.

```
model_path = 'C:/Users/Elie/djangoAi/savedModels'
model = load_model(model_path)
img = Image.open(file_path)
img = img.resize((100, 100))
img = img.convert('RGB')
# Convert the image to a numpy array
img_array = np.array(img)

# Add a fourth dimension to the image array
img_array = np.expand_dims(img_array, axis=0)
print(img_array[0].shape)
# Use the predict method to make predictions on the preprocessed image
prediction = model.predict(img_array)[0]

predicted_label_index = np.argmax(prediction)

if predicted_label_index == 1:
    msg = "You look angry today!"
elif predicted_label_index == 2:
    msg = "You look disgusted today."
elif predicted_label_index == 3:
```

Example on the web page

Upload a selfie to see how you look toady!

You look happy today!

Browse... No file selected. **Upload**