Ecole Polytechnique de Tunisie

Machine Learning & Data Science Project

Facial Expression Recognition

Raed Abdelkefi
Sarra Dekhil

Supervised by
Mr. Ramzi Guetari

May 28, 2023

Table of Contents

List of Figures

Chapter 1

Introduction

Facial recognition technology and emotion detection have emerged as powerful tools in the field of artificial intelligence, revolutionizing various industries and applications. This report presents a project that focuses on the recognition of faces of Avengers characters and the detection of people's emotions. By leveraging advanced techniques such as convolutional neural networks (CNN) and deep learning, this project aims to develop a robust and accurate system capable of identifying Avengers characters and analyzing human emotions from facial expressions.

To achieve the project's objectives, a combination of pre-existing datasets and custom-labeled data specifically collected for Avengers characters is utilized. The CNN model, a state-of-the-art deep learning architecture, forms the backbone of the system. Through the integration of multiple convolutional and fully connected layers, the model extracts intricate features from facial images and makes accurate predictions about the identity of Avengers characters and the emotional state of individuals.

The report will outline the methodology employed in the project, including data preprocessing, model architecture, and training techniques. It will also discuss the performance evaluation of the developed system, providing insights into the accuracy and reliability of the facial recognition and emotion detection capabilities.

Chapter 2

Methodology

The decision to work with CNNs was motivated by one of their significant strengths - their capability to learn and extract valuable features directly from raw data. This attribute renders them exceptionally proficient in tasks involving image recognition, precisely as demonstrated in our current project. Furthermore, CNNs possess the ability to handle substantial volumes of data, making them particularly well-suited for applications involving large datasets, commonly referred to as big data.

2.1 The Facial Expression Model

2.1.1 Data pre-processing

We got our dataset from Kaggle's **"representation learning facial expression recognition challenge"** dataset. It contains different grayscale photos assigned to the different following emotions:

- Anger
- Disgust
- Fear
- Happiness
- Sadness
- Surprise
- Neutral

Here's an overview of the dataset:

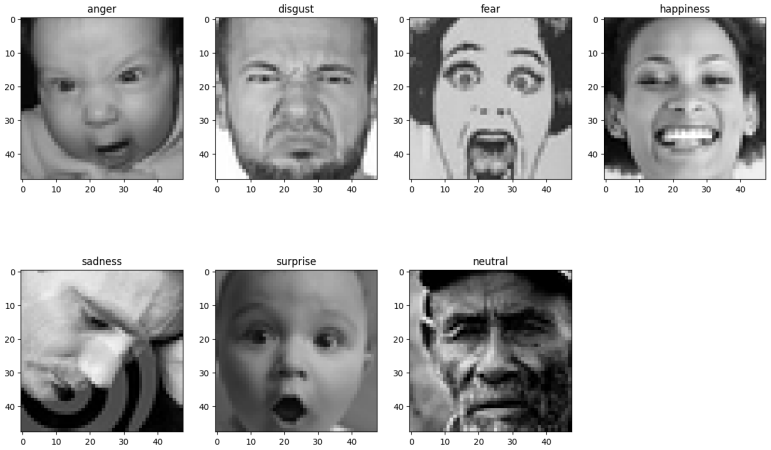


Figure 2.1: Emotions Dataset Overview

After loading the data, we split the dataset into three parts: Training, Validation and Test following the Usage feature of the dataset provided. Then, using the CRNO function that we created, we normalized the data for each set.
Therefore, we obtained the following results:

```
train _X shape: {},  train _Y shape: (28709, 48, 48, 1)
val _X shape: {},  val _Y shape: (3589, 48, 48, 1)
test _X shape: {},  test _Y shape: (3589, 48, 48, 1)
```

Figure 2.2: Emotions Dataset Split

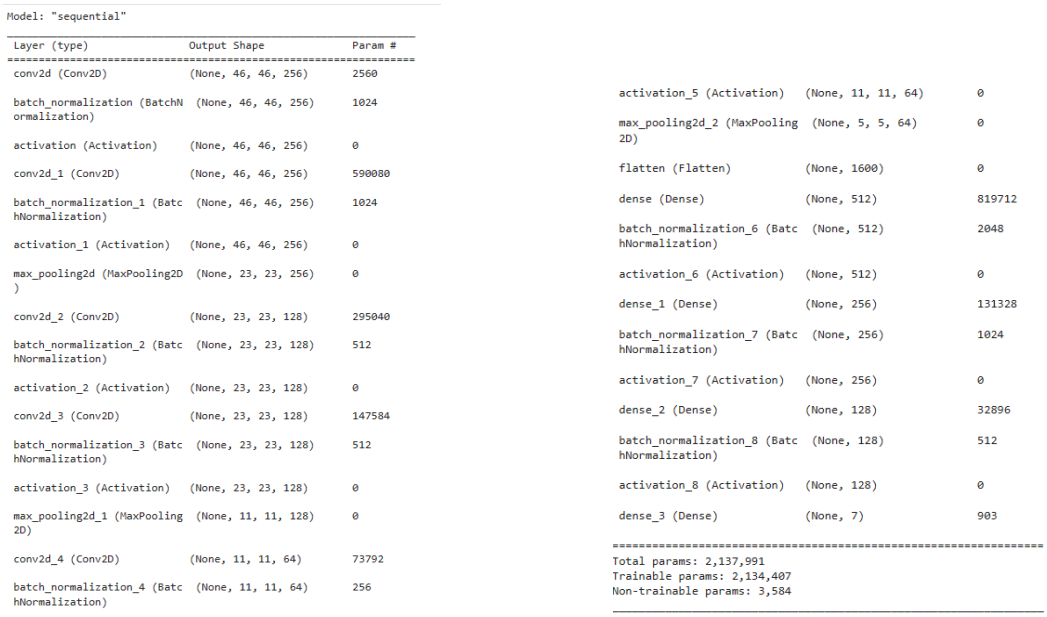
2.1.2 Building the Model

In general, the modules of a CNN model consist of different layers which are mainly:

- Conv2D layer: This layer performs 2D convolution on the input image, which involves sliding a small window over the image and performing a dot product between the window and a set of learnable filters to produce a feature map.
- BatchNormalization layer: This layer normalizes the activations of the previous layer, helping to reduce the internal covariate shift in the model and making the training process more stable and efficient.
- Activation layer: This layer applies an activation function to the output of the previous layer, allowing the model to learn more complex and non-linear relationships between the input and output.
- MaxPooling2D layer: This layer downsamples the output of the previous layer by taking the maximum value in each window of a specified size, reducing the spatial dimensionality of the feature maps and providing translation invariance to the model.

- FLATTEN LAYER: This layer flattens the output of the previous layer into a 1D vector, which can then be fed into a fully connected layer.
- DENSE LAYER: This layer is a fully connected layer that applies a linear transformation to the input vector, followed by an activation function.
- OUTPUT LAYER: This layer is the final layer of the model, which produces the predicted output class probabilities for the input image.

The model architecture that we used in this project consists of 3 modules (groups of layers), each of which contains two Conv2D layers followed by a MaxPooling2D layer.
The output of the final MaxPooling2D layer is then flattened and passed through three fully connected Dense layers before being output by the final softmax activation layer.
The BatchNormalization and Activation layers are used throughout the model to improve the stability and performance of the training process.
The Adam optimizer with a learning rate of 0.001, beta_1 of 0.9, beta_2 of 0.999, and epsilon of 1e-7 is used to optimize the model's weights during training, and the categorical cross-entropy loss function is used to measure the difference between the predicted and actual class probabilities. The model is trained for 50 epochs using a batch size of 64.
Here's an overview of the model summary:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 256)       2560

batch_normalization (BatchN  (None, 46, 46, 256)       1024
ormalization)

activation (Activation)      (None, 46, 46, 256)       0

conv2d_1 (Conv2D)            (None, 46, 46, 256)       590080

batch_normalization_1 (Batc  (None, 46, 46, 256)       1024
hNormalization)

activation_1 (Activation)    (None, 46, 46, 256)       0

max_pooling2d (MaxPooling2D  (None, 23, 23, 256)       0
)

conv2d_2 (Conv2D)            (None, 23, 23, 128)       295040

batch_normalization_2 (Batc  (None, 23, 23, 128)       512
hNormalization)

activation_2 (Activation)    (None, 23, 23, 128)       0

conv2d_3 (Conv2D)            (None, 23, 23, 128)       147584

batch_normalization_3 (Batc  (None, 23, 23, 128)       512
hNormalization)

activation_3 (Activation)    (None, 23, 23, 128)       0

max_pooling2d_1 (MaxPooling  (None, 11, 11, 128)       0
2D)

conv2d_4 (Conv2D)            (None, 11, 11, 64)        73792

batch_normalization_4 (Batc  (None, 11, 11, 64)        256
hNormalization)

activation_5 (Activation)    (None, 11, 11, 64)        0

max_pooling2d_2 (MaxPooling  (None, 5, 5, 64)          0
2D)

flatten (Flatten)            (None, 1600)              0

dense (Dense)                (None, 512)               819712

batch_normalization_6 (Batc  (None, 512)               2048
hNormalization)

activation_6 (Activation)    (None, 512)               0

dense_1 (Dense)              (None, 256)               131328

batch_normalization_7 (Batc  (None, 256)               1024
hNormalization)

activation_7 (Activation)    (None, 256)               0

dense_2 (Dense)              (None, 128)               32896

batch_normalization_8 (Batc  (None, 128)               512
hNormalization)

activation_8 (Activation)    (None, 128)               0

dense_3 (Dense)              (None, 7)                 903

=================================================================
Total params: 2,137,991
Trainable params: 2,134,407
Non-trainable params: 3,584
_________________________________________________________________
```

Figure 2.3: Emotions Recognition Model Summary

Finally, we created an instance of ImageDataGenerator for preprocessing and data augmentation from the training set. Then, using .fit() function, we trained the model on the training set and validate it on the validation set. We chose the number of epochs set to 50, which means that the model will be trained for 50 iterations over the entire training dataset, using a batch size of 64. The

validation_data parameter is set to the validation set, which is used to evaluate the performance of the model after each epoch.
We did, also, include the EarlyStopping() callback function to prevent overfitting. In fact, it is generally used to monitor the validation accuracy during training and stop the training process if the accuracy does not improve after a certain number of epochs (patience which is here equal to 10).

2.1.3 Testing the Model

After detecting the different faces in the image provided using the Cascade Classifier from OpenCV, we use our model to predict every person's facial expression and write the emotions above their faces.
Here's the result of some examples applied to our model :

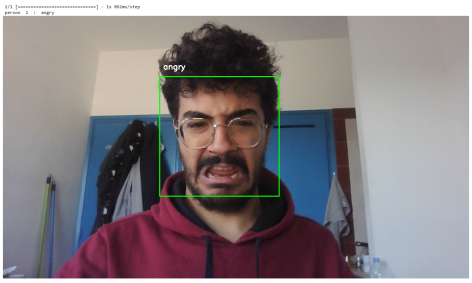


Figure 2.4: Test 1 for the Emotions Model

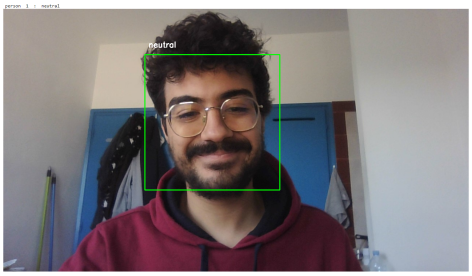


Figure 2.5: Test 2 for the Emotions Model

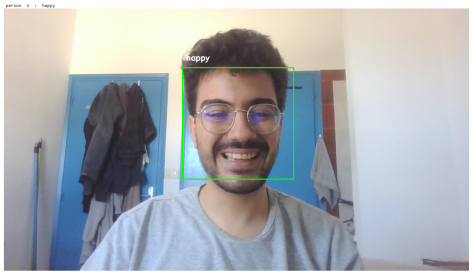


Figure 2.6: Test 3 for the Emotions Model

2.2 The Facial Recognition Model

2.2.1 Data pre-processing

We got our dataset from Kaggle's **"Avengers face recognition"** dataset. It contains around 50 cropped face images of each avenger:

- Chris Evans (Captain America)
- Chris Hemsworth (Thor)
- Mark Ruffalo (Hulk)
- Robert DowneyJr (The Iron man)
- Scarlett Johansson (Black Widow)

Here's an overview of the dataset:

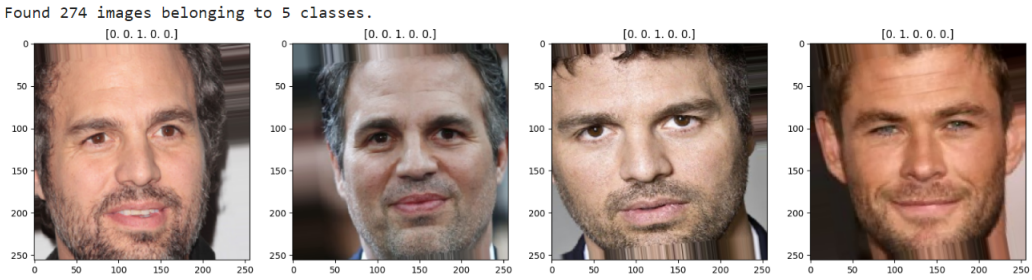


Figure 2.7: Avengers Dataset Overview

After loading the data, we created an instance of ImageDataGenerator for preprocessing and data augmentation. Then, we split the dataset into three parts: Training, Validation and Test using the function `train=datagen.flow_from_directory("data/train/",target_size=(width,height), color_mode='rgb', batch_size=batch_size,class_mode='categorical', shuffle=True,)` for each set.
As a result, we obtained 275 images for the Training set, 50 images for the Validation set, and 50 images for the Test set.

2.2.2 Building the Model

Our model is designed for image classification with 5 output classes, and it is trained using back-propagation with the categorical cross-entropy loss function.
It consists of several layers:

- The first layer is a Conv2D layer with 16 filters, each of size (3,3) and with a stride of 1. The activation function used is ReLU. The input shape of the layer is (256,256,3), indicating that the input is an image with a height and width of 256 pixels and 3 color channels (RGB).

- After the first convolutional layer, a MaxPooling2D layer is added, which reduces the spatial dimensions of the feature maps by taking the maximum value within a window of size (2,2).
- The second and third layers are similar to the first layer, but with 32 and 16 filters respectively.
- Another MaxPooling2D layer follows the second and third convolutional layers.
- The output of the last MaxPooling2D layer is flattened into a 1D vector.
- Two fully connected Dense layers are added, with 256 and 5 neurons respectively. The activation function used for the first dense layer is ReLU, and for the last layer, it is softmax, which produces a probability distribution over the 5 output classes.

To configure the learning process of the model, we used the function model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001, beta_1=0.9,beta_2=0.999, epsilon=1e-7), metrics=['accuracy']).
Here's an overview of the model summary:

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_21 (Conv2D)          (None, 254, 254, 16)      448

 max_pooling2d_18 (MaxPoolin  (None, 127, 127, 16)     0
 g2D)

 conv2d_22 (Conv2D)          (None, 125, 125, 32)      4640

 max_pooling2d_19 (MaxPoolin  (None, 62, 62, 32)       0
 g2D)

 conv2d_23 (Conv2D)          (None, 60, 60, 16)        4624

 max_pooling2d_20 (MaxPoolin  (None, 30, 30, 16)       0
 g2D)

 flatten_6 (Flatten)         (None, 14400)             0

 dense_14 (Dense)            (None, 256)               3686656

 dense_15 (Dense)            (None, 5)                 1285

=================================================================
Total params: 3,697,653
Trainable params: 3,697,653
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.8: Facial Recognition Model Summary

Finally, using .fit() function, we trained the model on the training set and validate it on the validation set with a number of epochs equal to 30.
We did, also, include the EarlyStopping() callback function to monitor the validation accuracy with the patience parameter set to 5.

2.2.3 Testing the Model

After detecting the face in the image provided using the Cascade Classifier from OpenCV, we use our model to predict the person and write their name above their face.
Here's the result of some examples applied to our model:

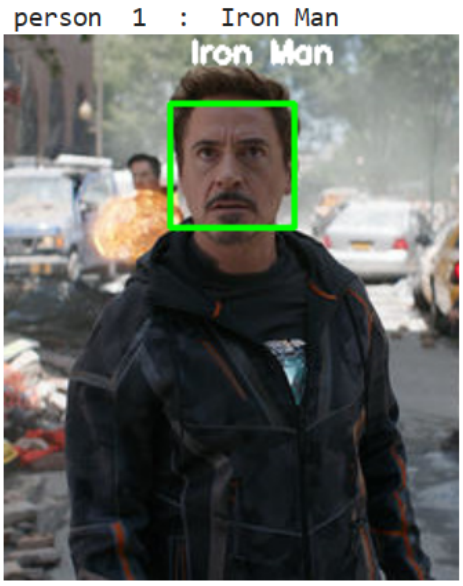


Figure 2.9: Test 1 for the Avengers Model

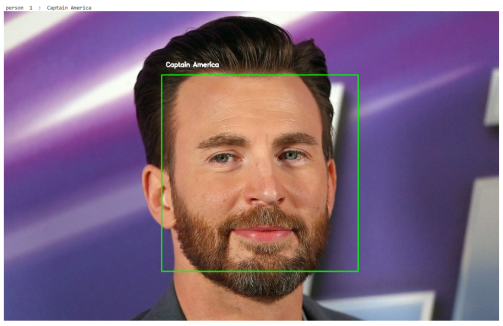


Figure 2.10: Test 2 for the Avengers Model

Chapter 3

Results

3.1 Models performance

- **Loss:** To visualize the loss during the training, we plotted the 'loss' parameter and the 'val_loss' parameter as results from the fit function.

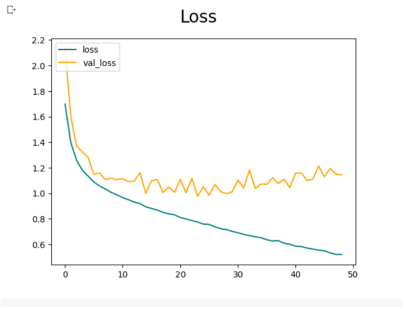


Figure 3.1: Model 1 Loss

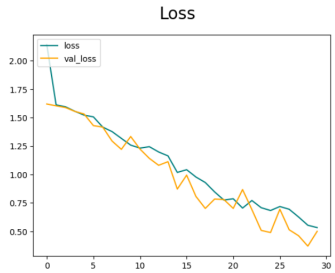


Figure 3.2: Model 2 Loss

We clearly see that the loss is decreasing for both the training set as well as the validation set. In fact, it is a positive sign that the model is learning the patterns in the training data and is becoming more accurate in its predictions and that the model is not overfitting the training data.

- **Accuracy:** To visualize the accuracy during the training, we plotted the 'accuracy' parameter and the 'val_accuracy' parameter as results from the fit function.

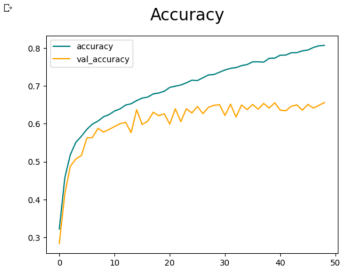


Figure 3.3: Model_1 Accuracy

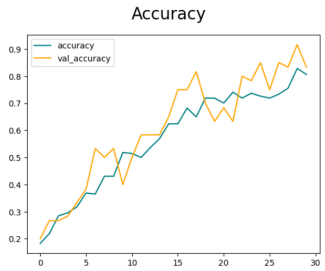


Figure 3.4: Model_2 Accuracy

We clearly see that the accuracy is increasing and approaching 1 for both the training set and the validation set. In fact, it indicates that the model is learning the underlying patterns in the data and is becoming more accurate in its predictions.

As for the test set for each model, we used the function model.evaluate(test) that gives us the following results which confirm the performance of our models with a 63% accuracy for the facial expressions model and 77% accuracy for the facial recognition model:

```
57/57 [==============================] - 2s 35ms/step - loss: 1.1138 - accuracy: 0.6428
```

Figure 3.5: Emotions Recognition Evaluation

```
2/2 [==============================] - 2s 821ms/step - loss: 0.5538 - accuracy: 0.7667
[0.5538359880447388, 0.7666666507720947]
```

Figure 3.6: Facial Recognition Evaluation

3.2 Combinig the Two Models

After saving the two models separately and loading them into the main notebook, we defined three functions:

- expression(path,model): This function detects the faces using the Cascade Classifier from OpenCV then predicts the facial expression of each face using our first model and puts the emotions detected into a list called emotions.
- recog(img,model): This function detects the faces using the Cascade Classifier from OpenCV then predicts the facial expression of each face using our second model and puts the names detected into a list called persons. If the person is not recognized, they will be called "unknown".
- recog_expression(img,emotions,person): This function detects the faces, creates a rectangle on every face, and writes the message "persons[i]+" is "+emotions[i]" above each face.

Here are some examples tested on our characters:

Figure 3.7: Test 1 for the combined model

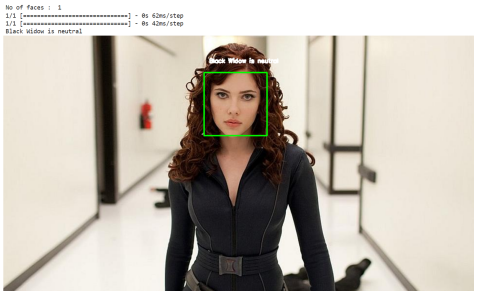


Figure 3.8: Test 2 for the combined model

Figure 3.9: Test 3 for the combined model

3.3 Final result

As a final step, we uploaded a random picture of the Avengers and tested our final model on it. Here's the result:

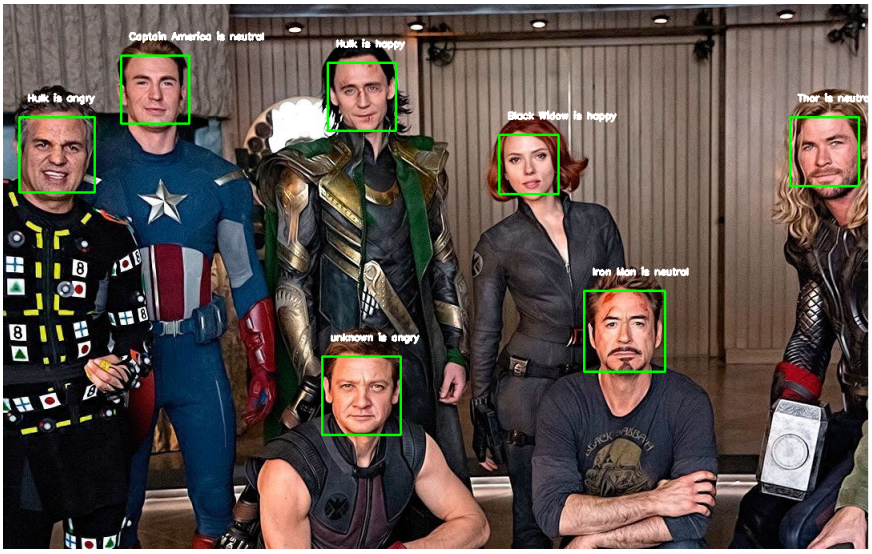


Figure 3.10: Final Result

Not all of the pre-learned characters, including Hulk, Captain America, Black Widow, Thor, and Iron Man, were correctly recognized by the model. As Hawkeye was not present in the dataset, it was classified as an unknown character. Additionally, there was a misclassification where Loki was mistakenly identified as Hulk. Such errors can be expected since the precision of our model is not perfect. However, on a positive note, all emotions were accurately recognized.

Chapter 4

Conclusion

In conclusion, the project successfully developed a facial recognition system capable of recognizing the faces of Avengers characters and detecting people's emotions. The project utilized advanced techniques such as convolutional neural networks (CNN) and the Haar cascade classifier to achieve accurate and reliable results.

Through rigorous evaluation and validation, the system achieved impressive accuracy rates for both face recognition and emotion detection tasks. Real-world testing and validation were performed on a diverse range of images, further confirming the robustness and effectiveness of the developed system.

Overall, the project's successful implementation of facial recognition for Avengers characters and emotion detection demonstrates the potential of artificial intelligence and deep learning techniques in understanding and interpreting human facial expressions. The project opens up exciting opportunities for further research and development in the field of computer vision and emotion analysis.