UNIVERSITÀ DI PISA

Emotion Recognition
Intelligent Systems

ACADEMIC YEAR 2022-23

Erasmus+ Student
Maria Ewa Fabijan

Table of Contents

1. Introduction

The primary objective of this project is to develop a robust emotion recognition system. The aim is to construct a model capable of accurately classifying emotions from images, with applications spanning various domains such as human-computer interaction and mental health assessment. To achieve this goal, Convolutional Neural Networks (CNNs) were implemented, either from scratch or using pretrained networks.

2. Data

The dataset for the project is the OpenCV Emotion Images dataset from Kaggle. The dataset provides labeled images with emotions, enabling supervised learning.Each image is associated with an emotion label, such as Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. The dataset contains 32,298 images, allowing for a diverse and comprehensive training and testing process.

Link to dataset: https://www.kaggle.com/datasets/juniorbueno/rating-opencv-emotion-images

3. Why CNN for emotion recognition?

Emotion recognition, the ability to discern and interpret human emotions from visual cues, has emerged as a fascinating and multidisciplinary field with applications that transcend traditional boundaries. This captivating area of study finds relevance in diverse domains, including human-computer interaction, mental health assessment, and beyond. At the heart of any successful emotion recognition system lies a pivotal choice: selecting the most appropriate machine learning model. Convolutional Neural Networks (CNNs), a class of deep learning models, stand out as a compelling choice for this intricate task.

Emotion recognition is a nuanced undertaking. It demands not only an understanding of human emotions but also the ability to capture and decode subtle emotional cues embedded within visual data. To achieve this, a model must navigate through a labyrinth of facial expressions, considering minute details and nuanced variations. Such complexity necessitates a machine learning architecture that can autonomously distill meaningful information from raw data.

CNNs have demonstrated exceptional proficiency in extracting hierarchical features from complex data, particularly images. These networks are inherently designed to imitate the human visual system by simulating the processing that occurs in the human brain's visual cortex. In essence, they break down the input image into multiple layers of features, ranging from low-level features like edges and textures to high-level features like facial expressions. This hierarchical feature extraction capability aligns seamlessly with the requirements of emotion recognition, where understanding facial expressions is paramount.

The spatial arrangement of facial features is pivotal in conveying emotions. Subtle shifts in the positioning of eyes, nose, and mouth can profoundly alter the emotional context of an expression. CNNs are uniquely poised to preserve this spatial information during their training and inference. Through the use of convolutional layers and pooling operations, these networks maintain the relative positions of facial components, ensuring that the spatial context is factored into the emotion recognition process.

In real-world scenarios, emotions are expressed amidst a sea of variables, including changes in lighting conditions, facial pose, and background clutter. CNNs can be trained to be robust in the face of such variability. Techniques like data augmentation and regularization enable these networks to adapt to diverse and challenging image conditions, making them reliable instruments for emotion recognition in the wild.

CNNs offer an additional advantage: they seamlessly integrate with the concept of transfer learning. Pre-trained CNN models, which have previously acquired general features from massive datasets like ImageNet, can be fine-tuned for specific tasks like emotion recognition. This approach not only accelerates the model development process but also capitalizes on the wealth of knowledge embedded in these pre-trained architectures. As a result, it enhances the recognition accuracy and generalization capabilities of the model.

Emotion recognition often necessitates the use of large datasets and intricate models to achieve high accuracy. CNNs are highly scalable, offering researchers the flexibility to accommodate extensive datasets and construct complex neural architectures. This scalability is instrumental in pushing the boundaries of recognition accuracy and establishing state-of-the-art performance.

Understanding the decision-making process of an emotion recognition model is crucial for researchers and practitioners. CNNs provide valuable interpretability by highlighting which facial features or patterns are most influential in identifying specific emotions. This insight offers a deeper understanding of the model's inner workings and empowers users to trust and refine its predictions.

In conclusion, Convolutional Neural Networks are more than just mathematical models; they are powerful tools in the realm of emotion recognition. Their innate ability to autonomously extract relevant features, preserve spatial information, adapt to variability, and leverage transfer learning makes them exceptionally well-suited for this challenging endeavor. Furthermore, their scalability and interpretability contribute to their enduring appeal as a foundational technology in the field of emotion recognition. As we continue to unlock the mysteries of human emotions through the lens of CNNs, the applications and possibilities seem boundless.

4. System configuration

The dataset are already splitted in training and validation set, with each containing seven different classes.

4.1 Balancing the dataset

The dataset exhibits class imbalance in its distribution. The least numerous class is ‘Disgust’, which contains 436, while the most numerous class ‘Happy’ contains 7215 images. This inequality can result in suboptimal predictions for minority classes. To address this concern, undersampling techniques have been employed to balance the class distribution. After undersampling, each class consists 436 images.

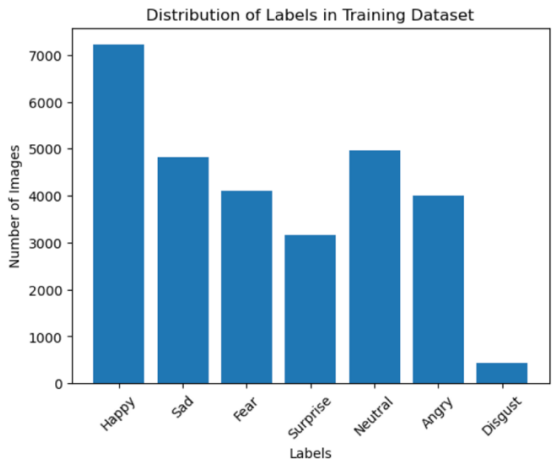

Figure 1: Distribution of class in training set

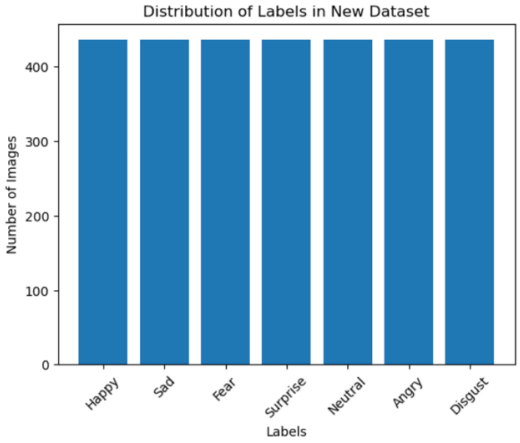

Figure 2: Distribution of class in training set after undersampling

4.2 CNN layers parameters

Design the Convolution Neural Network requires define several parameters, such as:

- **Kernel Size**
  It is also called filter size, which refers to window that moves over the input data. This window is responsible for the extraction features from the images. The choice of this parameter has a huge impact on the classification task. For example, small kernels can extract much more information from the input containing highly local functions. The smaller size of the kernel also leads to a smaller reduction in the dimensions of the layers, which allows for a deeper architecture. Conversely, a large kernel size extracts less information, leading to faster reduction of layer dimensions, often resulting in poorer performance. But sometimes we can gain a better generalization of the problem. Large kernels are better suited for extracting larger elements.
- **Padding**
  Padding defines how the swatch border is handled. This makes it possible to get the size of the output the same as the size of the input (assuming strides is shifted by one). This is achieved at the cost of adding additional (artificial) weights on the edges (usually with a value of zero). Zeros provide simplicity and computational efficiency.
- **Strides (steps)**
  The stride parameter indicates how many pixels the kernel should be moved at a time. In other words, it means a filter window shift step.
- **Activation features**
  One of the reasons why neural networks are able to achieve such enormous accuracy is their non-linearity. Non-linearity is necessary to produce non-linear decision limits so that the result cannot be written as a linear combination of the inputs.
  - ***ReLU*** (Rectified Linear Activation)
    ReLU applies the needed non-linearity to the model. It is quick to train due to the simplicity of its implementation. It is the most used activation function.
  - ***SoftMax***
    The softmax operation serves a key purpose: to make sure that the CNN scores add up to 1. For this reason, softmax operations are useful for scaling model outputs into probabilities.
    After passing through the softmax function, each class corresponds to a corresponding probability.
  - ***Flatten layer***
    This layer transforms our multidimensional layer in the network into a one-dimensional vector.

The softmax function is commonly used to classify features after extracting features from the input image through convolutional networks, which requires one-dimensional input data.

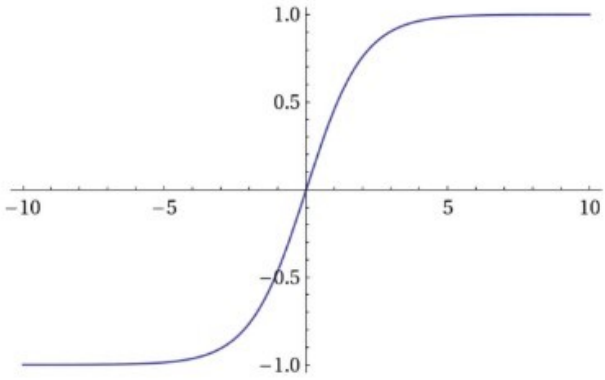

Figure 3: Softmax function

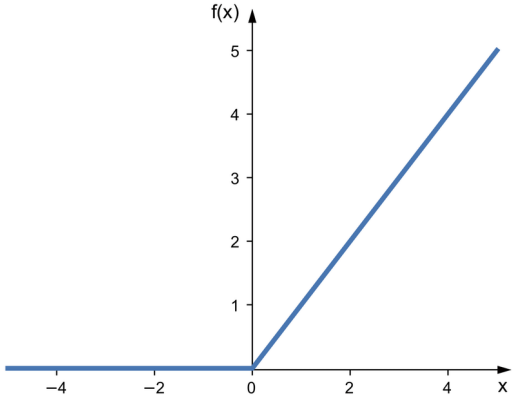

Figure 4: ReLu function

4.3 Pooling layers

The idea of pooling is map several pixels to 1 pixel. Two types of pooling are best known:

- max (maximum) – take the maximum value from defined number of pixels
- avg (avarage) – take the average value of defined number pixels.

4.4 Dropout layer

The most popular way to fight overtraining in the case of neural networks is dropout. Dropout is to randomly set the outgoing edges of hidden units to 0 each time you update your training phase. This method is very efficient because connections are randomly disconnected every time. Thanks to this, the neural network will not learn "by heart" too quickly, because the architecture changes slightly after the calculation by resetting random connections of neurons.

In this project Dropout I applied after the last convolution stage, and before the final fully connected layer with value of parameters 0.5.

4.5 Data Augmentation

The accuracy of models in supervised learning depends largely on the amount and variety of data available during training. In this case, I increase the amount of training data using data augmentation. When I train a new network using this data augmentation configuration, the network will never see twice the same input. This is another way to prevent overfitting.

In this project I used:

- horizontal flipping to a random 50% of the images
- rotates the input images by a random value in the range [–10%, +10%] (fraction of full circle [-36°, 36°])
- Zooms in or out of the image by a random factor in the range [-20%, +20%]

Figure 5: Examples of data augmentation

5. CNN from scratch

I attempted to create a custom neural network architecture from scratch to solve the emotion recognition detection problem. With numerous hyperparameters to choose from, I used a trial-and-error approach to find the optimal architecture.

5.1 First trail

The model's architecture consists:

Input Layer is tailored to accommodate data with dimensions of (None, 180, 180, 3). This indicates that the network can process images with a height and width of 180 pixels and three color channels (RGB).

Rescaling Layer that performs data preprocessing. This layer rescales the input data to a suitable range, ensuring that the pixel values are within a normalized range (between 0 and 1). It is crucial for enhancing the training process.

Convolutional Layers to capture essential features from the input images. The first convolutional layer, labeled "conv2d," has 32 filters and a kernel size of (3, 3), resulting in an output shape of (None, 178, 178, 32). Subsequently, there is a "max_pooling2d" layer that performs max-pooling operations, reducing the spatial dimensions by half, yielding an output of (None, 89, 89, 32). A similar pattern is repeated with subsequent convolutional and max-pooling layers, increasing the number of filters and further downsampling the feature maps.

- Flatten Layer to transform the 3D feature maps into a 1D vector. This flattening process prepares the data for the subsequent fully connected layers.
- Dense Layer with 7 output units, corresponding to the number of classes in the classification task. This layer computes the final class probabilities and facilitates the model's predictions.

The entire model architecture comprises a total of 1,066,311 trainable parameters, indicating the weights and biases that the model will adjust during training to optimize its performance.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2  (None, 89, 89, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 7)                 87815

=================================================================
Total params: 1066311 (4.07 MB)
Trainable params: 1066311 (4.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 6: Summary of first model

Results

The model indicates overfitting because the accuracy of the training set increases while the accuracy of the validation set remains constant.

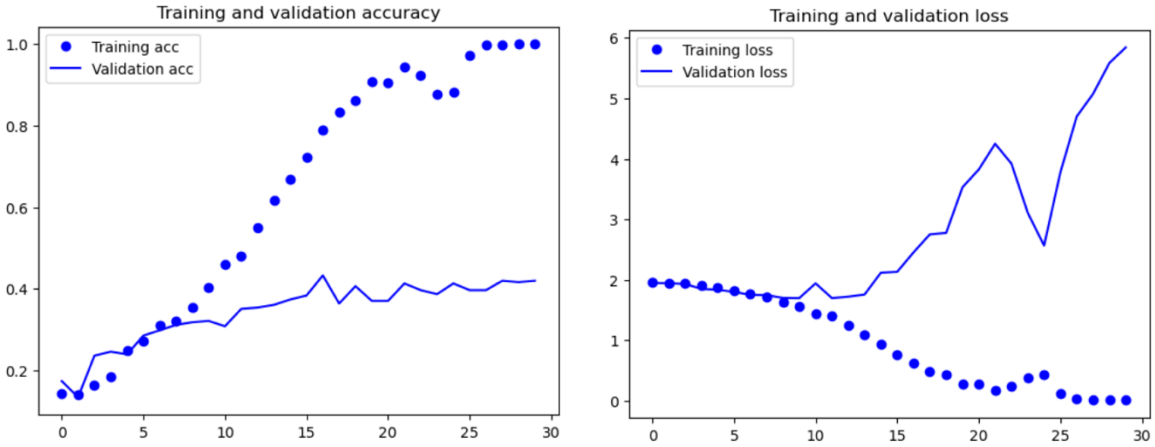

Figure 7: Plot of accuracy and loss

The accuracy of model is 0.3791, which is not so good results. The model has the best results for ‘Sad’ images (AUC=0.88) and the poorest results ‘Happy’ and ‘Angry’ images.

```
10/10 [==============================] - 1s 54ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.3077    0.3265    0.3168        49
           1     0.5102    0.6410    0.5682        39
           2     0.1778    0.2353    0.2025        34
           3     0.4500    0.4186    0.4337        43
           4     0.3696    0.3333    0.3505        51
           5     0.2571    0.2571    0.2571        35
           6     0.5897    0.4182    0.4894        55

    accuracy                         0.3791       306
   macro avg     0.3803    0.3757    0.3740       306
weighted avg     0.3943    0.3791    0.3824       306
```

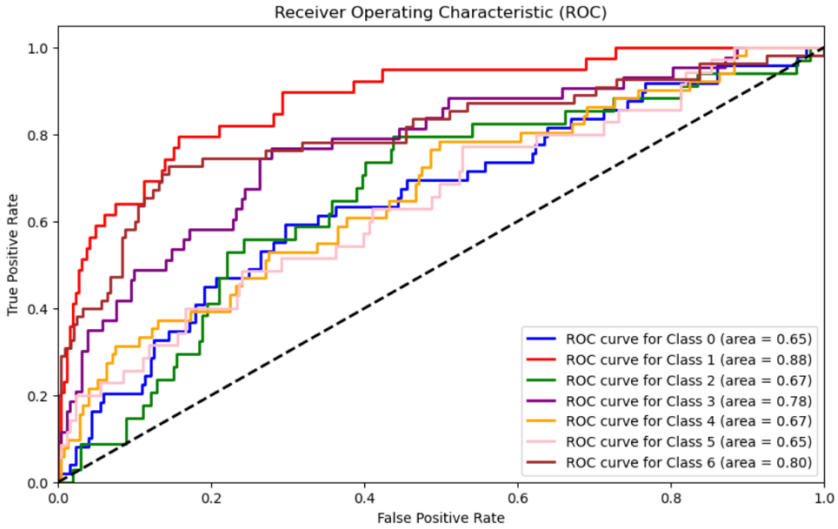

Figure 8: First model evaluation

5.2 Second trail – Early Stopping

The first technique to mitigate overfitting is Early Stopping. For the first model, I added callbacks with the EarlyStopping class to stop training when a monitored metric has stopped improving. The number of epochs with no improvement after which training will be stopped was set on 5 consecutive epochs

Results

The model stopped after 4 epoches.

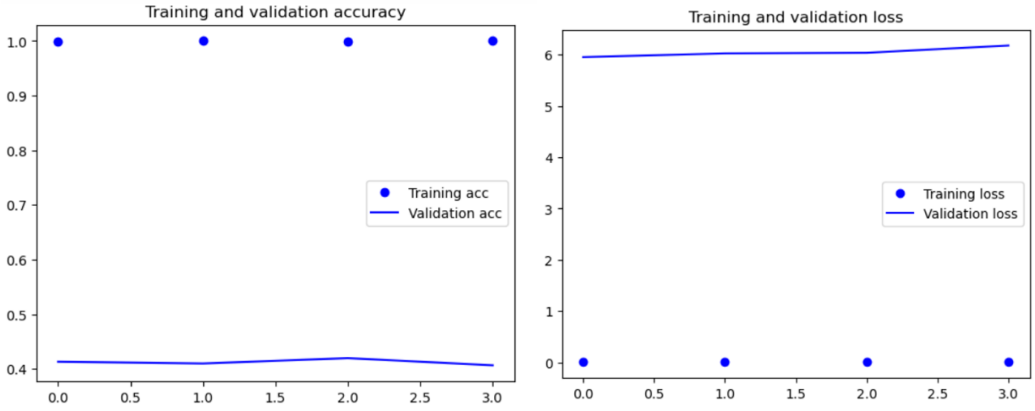

Figure 9: Plot of training and validation accuracy and loss

Some classes have benefited from Early Stopping by preventing overfitting, while others have seen a decrease in performance because the model stopped training before it had a chance to further improve on those classes.
The overall increase in accuracy to 0.3824 suggests that the model's ability to generalize to unseen data has improved, which is a positive outcome.

```
10/10 [==============================] - 0s 29ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.3077    0.3265    0.3168        49
           1     0.5625    0.6923    0.6207        39
           2     0.1778    0.2353    0.2025        34
           3     0.4222    0.4419    0.4318        43
           4     0.3750    0.2941    0.3297        51
           5     0.2105    0.2286    0.2192        35
           6     0.6316    0.4364    0.5161        55

    accuracy                         0.3824       306
   macro avg     0.3839    0.3793    0.3767       306
weighted avg     0.4001    0.3824    0.3858       306
```

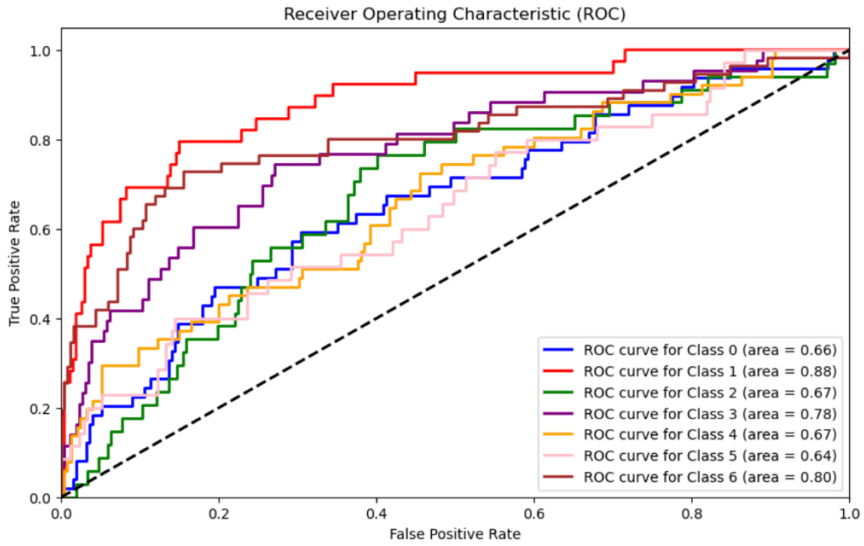

Figure 10: Evaluation of first model with Eearly Stopping

5.3 Third trail - Data Augumentation

The second technique to overcome overfitting was introducing data augmentation. The data augmentation was added as an additional layer in model.

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_5 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_4 (MaxPoolin  (None, 89, 89, 32)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_7 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_6 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_8 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_7 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 conv2d_9 (Conv2D)           (None, 7, 7, 256)         590080

 flatten_1 (Flatten)         (None, 12544)             0

 dense_1 (Dense)             (None, 7)                 87815

=================================================================
Total params: 1066311 (4.07 MB)
Trainable params: 1066311 (4.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 11: Second model with data augmentation

Results

The data augmentation was not prevent overfitting as was expected.

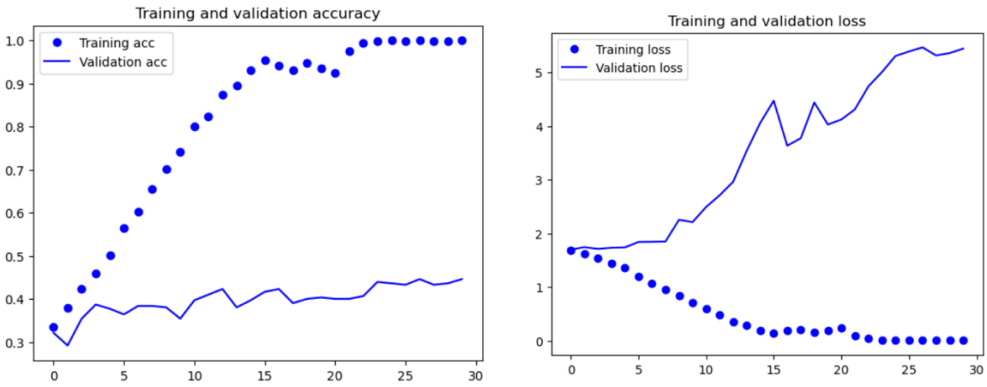

Figure 12: Plot of accuracy and loss training and validation sets

```
10/10 [==============================] - 0s 25ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.1957    0.1837    0.1895        49
           1     0.4902    0.6410    0.5556        39
           2     0.1190    0.1471    0.1316        34
           3     0.3571    0.3488    0.3529        43
           4     0.3077    0.3137    0.3107        51
           5     0.1892    0.2000    0.1944        35
           6     0.7778    0.5091    0.6154        55

    accuracy                         0.3431       306
   macro avg     0.3481    0.3348    0.3357       306
weighted avg     0.3699    0.3431    0.3500       306
```

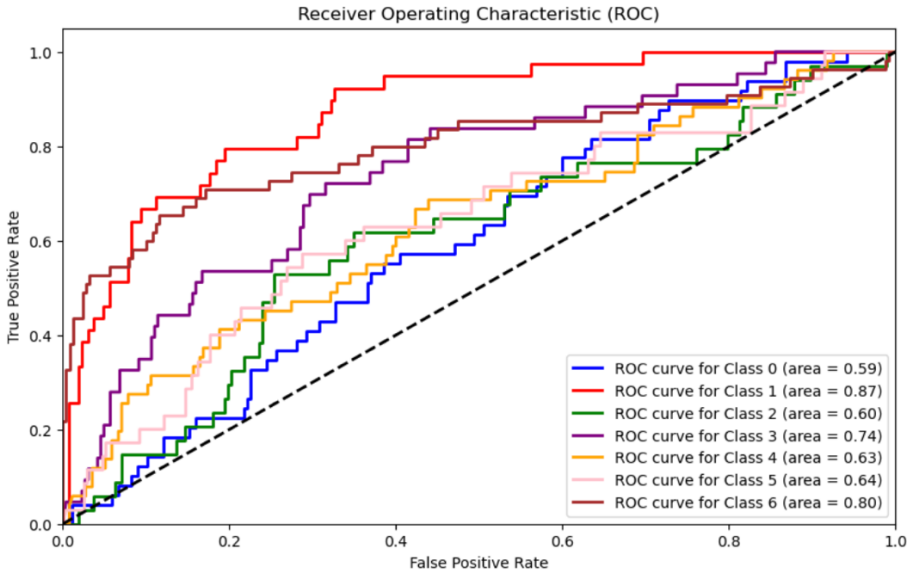

Figure 13: Second model evaluation

5.4 Fourth trial - Dropout layer

The last technique used to improve model, was adding the dropout layer.

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 180, 180, 3)]     0

 sequential (Sequential)     (None, 180, 180, 3)       0

 rescaling_3 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_15 (Conv2D)          (None, 178, 178, 32)      896

 max_pooling2d_12 (MaxPooli  (None, 89, 89, 32)        0
 ng2D)

 conv2d_16 (Conv2D)          (None, 87, 87, 64)        18496

 max_pooling2d_13 (MaxPooli  (None, 43, 43, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 41, 41, 128)       73856

 max_pooling2d_14 (MaxPooli  (None, 20, 20, 128)       0
 ng2D)

 conv2d_18 (Conv2D)          (None, 18, 18, 256)       295168

 max_pooling2d_15 (MaxPooli  (None, 9, 9, 256)         0
 ng2D)

 conv2d_19 (Conv2D)          (None, 7, 7, 256)         590080

 flatten_3 (Flatten)         (None, 12544)             0

 dropout_1 (Dropout)         (None, 12544)             0

 dense_3 (Dense)             (None, 7)                 87815

=================================================================
Total params: 1066311 (4.07 MB)
Trainable params: 1066311 (4.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 14: Third model with dropout layer

Results

The plot of validation and loss of training and validation sets shows that Dropout layer overcome with overfitting.

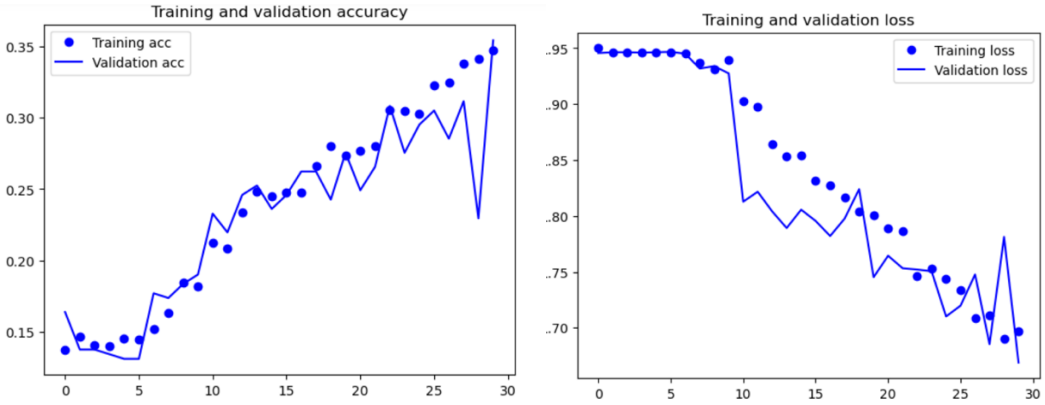

Figure 15: Plot of accuracy and loss of training and validation sets

```
10/10 [==============================] - 1s 32ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.2727    0.1224    0.1690        49
           1     0.3148    0.4359    0.3656        39
           2     0.1538    0.1765    0.1644        34
           3     0.4028    0.6744    0.5043        43
           4     0.4000    0.1961    0.2632        51
           5     0.1731    0.2571    0.2069        35
           6     0.6667    0.5091    0.5773        55

    accuracy                         0.3431       306
   macro avg     0.3406    0.3388    0.3215       306
weighted avg     0.3638    0.3431    0.3341       306
```

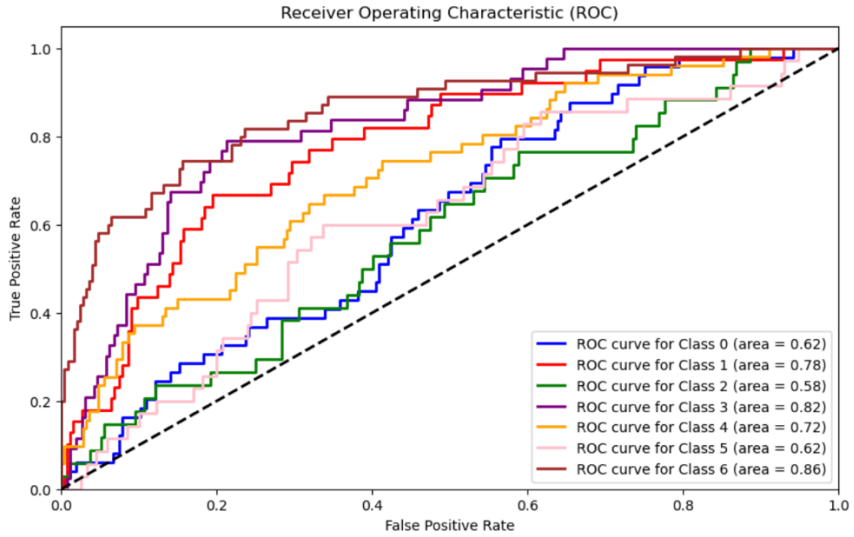

Figure 16: Third model evaluation

5.5 Fifth trail - L2 regularization

Dropout layers with a dropout rate of 0.2 are added after each convolutional layer to prevent overfitting during training. L2 regularization is applied to the dense layer with a regularization strength (L2 coefficient) of 0.01.

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 180, 180, 3)]     0

 sequential (Sequential)     (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 178, 178, 32)      896

 dropout (Dropout)           (None, 178, 178, 32)      0

 max_pooling2d_3 (MaxPoolin  (None, 89, 89, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 87, 87, 64)        18496

 dropout_1 (Dropout)         (None, 87, 87, 64)        0

 max_pooling2d_4 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 41, 41, 128)       73856

 dropout_2 (Dropout)         (None, 41, 41, 128)       0

 max_pooling2d_5 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_6 (Conv2D)           (None, 18, 18, 256)       295168

 dropout_3 (Dropout)         (None, 18, 18, 256)       0

 max_pooling2d_6 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 conv2d_7 (Conv2D)           (None, 7, 7, 256)         590080

 dropout_4 (Dropout)         (None, 7, 7, 256)         0

 flatten_4 (Flatten)         (None, 12544)             0

 dropout_5 (Dropout)         (None, 12544)             0

 dense_7 (Dense)             (None, 512)               6423040

 dense_8 (Dense)             (None, 7)                 3591

=================================================================
Total params: 7405127 (28.25 MB)
Trainable params: 7405127 (28.25 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 17: Summary model with L2 regularization

Results

L2 regularization mitigate overfitting, but the performance of model is low.

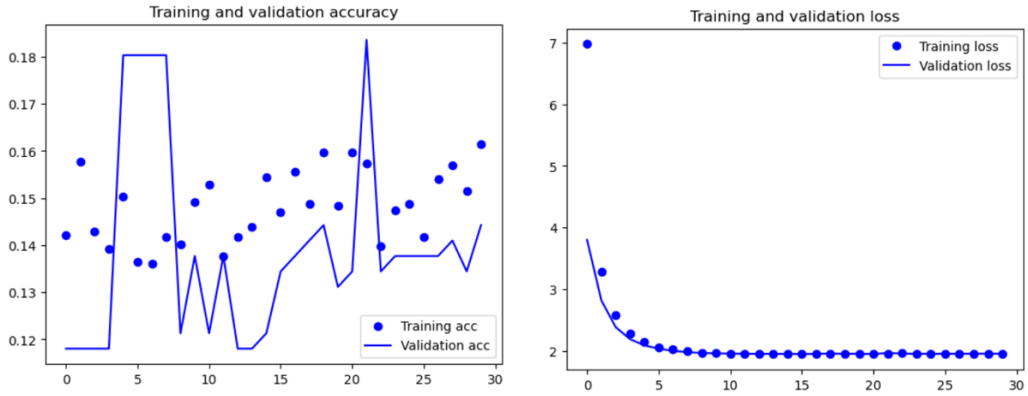

Figure 18: Plot of loss and accuracy of training and validation set

```
10/10 [==============================] - 1s 44ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000        49
           1     0.1667    0.0256    0.0444        39
           2     0.1124    0.8529    0.1986        34
           3     0.0000    0.0000    0.0000        43
           4     0.1951    0.1569    0.1739        51
           5     0.0000    0.0000    0.0000        35
           6     0.0000    0.0000    0.0000        55

    accuracy                         0.1242       306
   macro avg     0.0677    0.1479    0.0596       306
weighted avg     0.0663    0.1242    0.0567       306
```

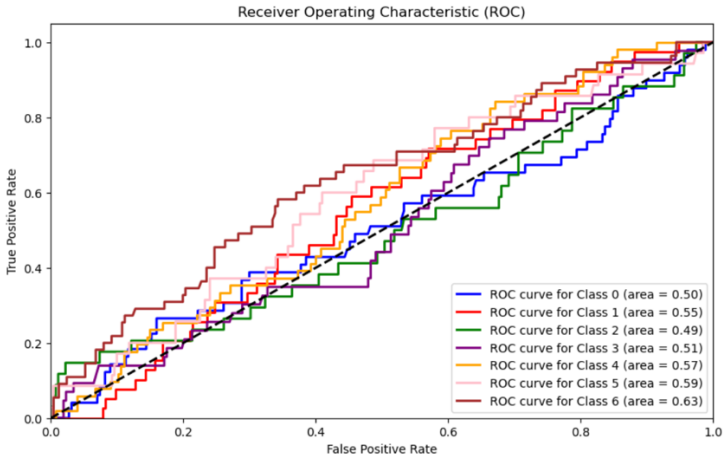

Figure 19: Evaluation model with reguralization L2

6. Pretrained CNN

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision and image processing. These neural networks are highly effective at tasks such as image classification. In this project the **VGG16 model** was used to improve the performance of model.

6.1 Feature extraction

Feature extraction using VGG16 involves using the pre-trained network as a fixed feature extractor. Instead of training the entire network from scratch, the fully connected layers was removed (the top part of the network) and use the remaining layers to extract features from dataset.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 sequential (Sequential)     (None, 180, 180, 3)       0

 tf.__operators__.getitem (  (None, 180, 180, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 180, 180, 3)       0
 )

 vgg16 (Functional)          (None, 5, 5, 512)         14714688

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 256)               3277056

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 17993543 (68.64 MB)
Trainable params: 3278855 (12.51 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Figure 20: Summary of first pretrained model

Results

The VGG16 model is not overfitted.

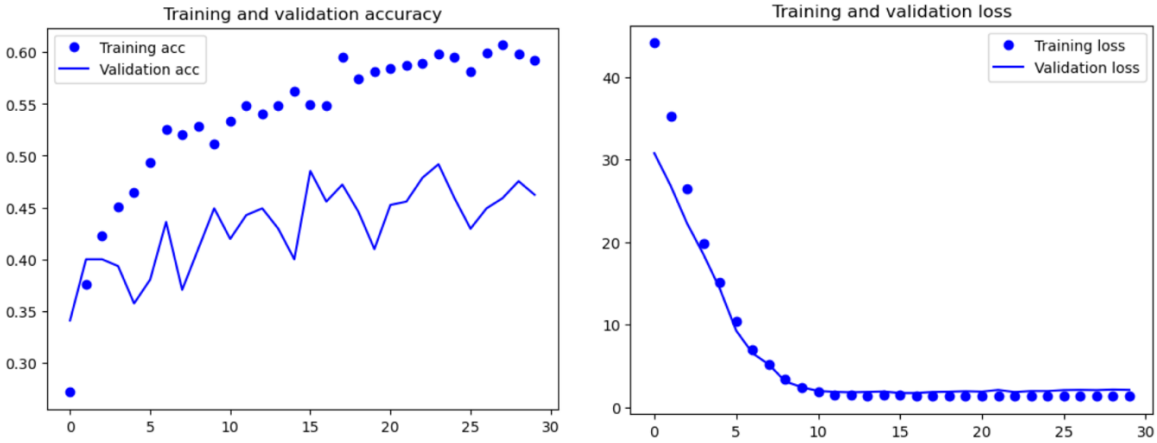

Figure 21: Plot of accuarcy and loss training and validation set

The accuracy is the highest from all previous models around 0.4641. The lowest metrices are for 2 class, which represents images labeled as ‘Fear’. However, for the others class the AUC is high.

```
10/10 [==============================] - 2s 241ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.2973    0.2245    0.2558        49
           1     0.5000    0.7179    0.5895        39
           2     0.2500    0.2059    0.2258        34
           3     0.6111    0.5116    0.5570        43
           4     0.6176    0.4118    0.4941        51
           5     0.2535    0.5143    0.3396        35
           6     0.7955    0.6364    0.7071        55

    accuracy                         0.4641       306
   macro avg     0.4750    0.4603    0.4527       306
weighted avg     0.4999    0.4641    0.4677       306
```

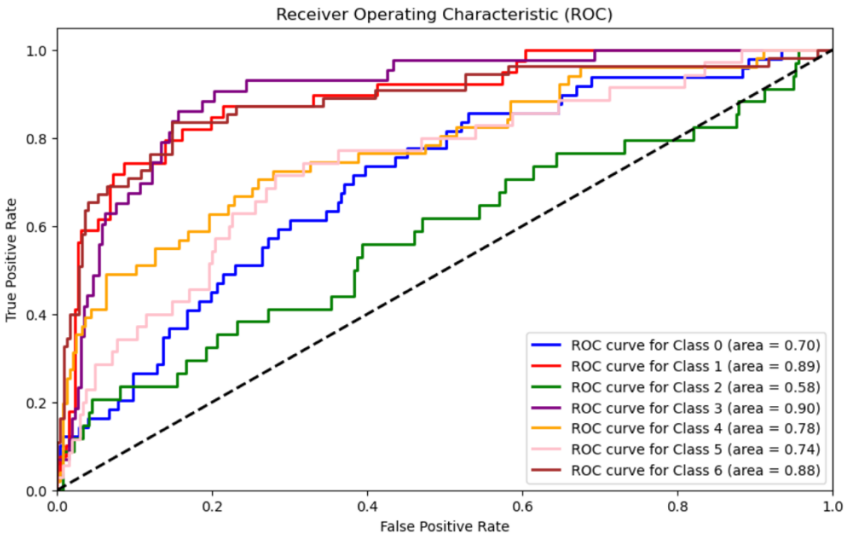

Figure 22: Evaluation of feature extraction of VGG16 model

6.2 Fine-tuning

Fine-tuning is a crucial technique in transfer learning that allows to adapt a pretrained model VGG16 to a specific task while retaining some of the knowledge it has gained from its original training. Fine-tuning involves making slight adjustments to the model's weights, typically in the deeper layers, to improve its performance on a new, related task.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 sequential (Sequential)     (None, 180, 180, 3)       0

 tf.__operators__.getitem (  (None, 180, 180, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 180, 180, 3)       0
 )

 vgg16 (Functional)          (None, 5, 5, 512)         14714688

 flatten (Flatten)           (None, 12800)             0

 dense (Dense)               (None, 256)               3277056

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 17993543 (68.64 MB)
Trainable params: 10358279 (39.51 MB)
Non-trainable params: 7635264 (29.13 MB)
_________________________________________________________________
```

Figure 23: Summary of VGG-16 model with fine-tuning

Results

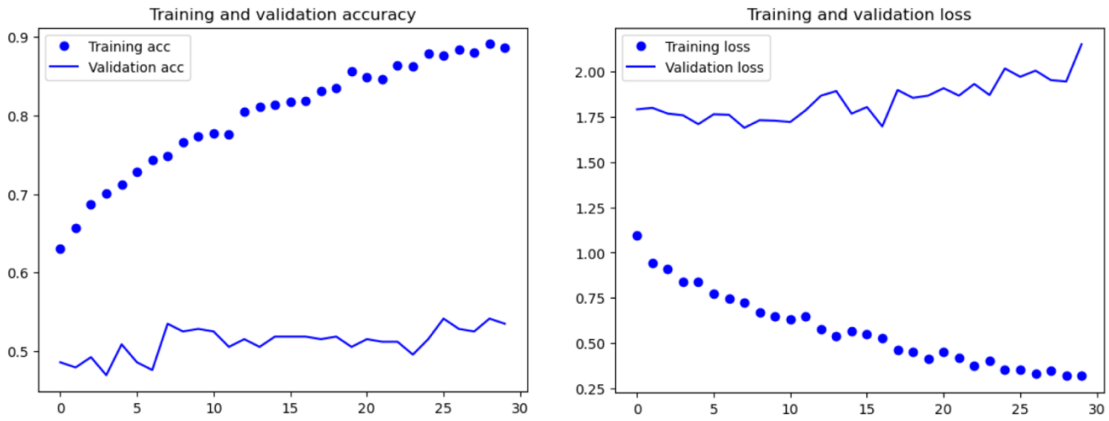

Figure 24: Plot of accuracy and loss of training and validation sets

The VGG-16 with fine-tuning achieved the highest accuracy 0.5033 and good metrices for each class.

```
10/10 [==============================] - 5s 560ms/step
Classification report:
              precision    recall  f1-score   support

           0     0.2857    0.1633    0.2078        49
           1     0.5918    0.7436    0.6591        39
           2     0.2963    0.2353    0.2623        34
           3     0.7097    0.5116    0.5946        43
           4     0.4762    0.5882    0.5263        51
           5     0.3273    0.5143    0.4000        35
           6     0.7358    0.7091    0.7222        55

    accuracy                         0.5033       306
   macro avg     0.4890    0.4951    0.4818       306
weighted avg     0.5029    0.5033    0.4933       306
```

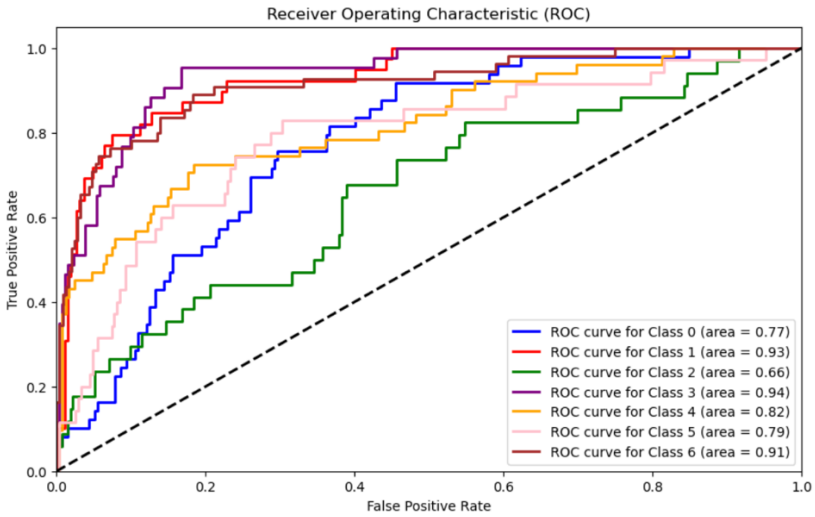

Figure 25: Evaluation of VGG-16 model with fine-tunning

7. Summary

In this project, the aim was to build a robust image classification model for a dataset with imbalanced class distributions. The dataset initially posed challenges related to class imbalance, overfitting, and the need for improved model performance. I employed various techniques and explored different model architectures to address these challenges.

To mitigate overfitting, I implemented the Early Stopping technique. To further combat overfitting, I introduced data augmentation as an additional layer in the model.
The best results in the project were achieved by fine-tuning a pretrained VGG-16 model. This highlighted the challenges of building a CNN from scratch and the benefits of leveraging

pretrained models. Pretrained models, trained on extensive datasets, provide a strong foundation for building models with superior performance.

In conclusion, this project demonstrated that building an effective image classification model can be challenging, especially with imbalanced datasets. The use of undersampling, regularization techniques, early stopping, and leveraging pretrained models played pivotal roles in improving the model's performance. The results emphasize the advantage of fine-tuning pretrained models, showcasing their ability to outperform models built from scratch when handling complex tasks in computer vision.