Facial Expression Classifier

Irene Silvestro

February 2025

Introduction

This project was developed with the idea of building a classifier for facial expressions. Considering that the task was quite challenging, the goal was not to achieve a network with excellent performance, but rather to analyze what the network "looks at" and, above all, where it fails to classify correctly.
In the following sections, different approaches to the problem will be presented: in some cases, a classifier with 7 output classes (the original number of classes in the dataset) will be implemented; in others, only 5 classes will be used (by discarding those with fewer samples); and finally, the problem will be binarized by asking the network to simply distinguish between "Negative" and "Positive" expressions.

Dataset

Dataset Splitting and Visualization

The dataset was downloaded from the following link: https://www.kaggle.com/msambare/fer2013

There are 7 emotion classes: *Angry, Disgust, Fear, Happy, Neutral, Sad, Surprise*. The images are in *gray_scale* format and have a resolution of 48x48 pixels.
In the original dataset, only a Training and Test set were provided. Therefore, the dataset was further split so that, for each class, the data were divided as follows: 70% Training Set, 20% Validation Set, and 10% Test Set.

Let us visualize a few sample images from the dataset — five per class:

In each section, I will show the dataset distribution, which is not well balanced among the different classes. Therefore, several strategies were adopted to give equal weight to each class:
1) **Balancing the dataset** by randomly removing samples from the overrepresented classes so that each class has the same number of elements.
2) Using the ***class_weight*** parameter during the training process, to balance the classes during learning (as will be explained later).
3) **Removing from the dataset** the class with the smallest number of samples and the one that, according to previous analyses, "confuses" the network the most.
These approaches are sometimes combined, and the specific configuration will be detailed in each section.

Data Augmentation, Normalization, and Minibatch

Next, data augmentation was applied only to the training data, and it was used consistently across all experiments described in the following sections. The operations applied to the training images include: a random horizontal shift up to 10% of the image width, a random vertical shift up to 10% of the image height, and a random horizontal flip.
Finally, a normalization step was performed so that the value of each feature (pixel) lies between 0 and 1. The dataset is fed to the network in minibatches of size **batch_size** = 64.

Model Selection

Given the complexity of the task and the fact that the dataset is not particularly optimal, I decided to use a network with convolutional layers from the start.

Model Architecture

The model architecture was determined through an empirical process, balancing a trade-off between achieved accuracy and precision, and the training time required for the most complex task. Specifically, in order to improve accuracy, I aimed to include the maximum possible number of layers, provided that each epoch required a reasonable amount of training time (in the model trained on the full dataset, the average epoch duration was about 180–200 seconds). It is important to note that the number of layers could only be increased to improve accuracy, as there was never a risk of overfitting due to the complexity of the task.

Layer	Type	Output Size	Activation	Parameters
1	Conv I (32 filters, 3x3)	46x46x32	ReLU	320
2	BatchNormalization	46x46x32	-	128
3	Conv II (64 filters, 3x3)	44x44x64	ReLU	18,496
4	BatchNormalization	44x44x64	-	256
5	MaxPooling (2x2)	22x22x64	-	0
6	Dropout (0.25)	22x22x64	-	0
7	Conv III (128 filters, 3x3)	20x20x128	ReLU	73,856
8	BatchNormalization	20x20x128	-	512
9	Conv IV (128 filters, 3x3)	18x18x128	ReLU	147,584
10	BatchNormalization	18x18x128	-	512
11	MaxPooling (2x2)	9x9x128	-	0
12	Dropout (0.25)	9x9x128	-	0
13	Conv V (256 filters, 3x3)	7x7x256	ReLU	295,168
14	BatchNormalization	7x7x256	-	1,024
15	Conv VI (256 filters, 3x3)	5x5x256	ReLU	590,080
16	BatchNormalization	5x5x256	-	1,024
17	MaxPooling (2x2)	2x2x256	-	0
18	Dropout (0.25)	2x2x256	-	0
19	Flatten	1024	-	0
20	FC I (256 neurons)	256	ReLU	262,400
21	BatchNormalization	256	-	1,024
22	Dropout (0.5)	256	-	0
23	FC II (7 neurons)	7	Softmax	1,799

Tabella 1: Neural network architecture

In the six convolutional layers, the default parameters of the *Conv2D* function were kept, namely stride = 1 and padding = 0.

The total number of parameters is 1,394,185. This network architecture will be used throughout all the analyses presented in the following sections, with only the number of neurons in the final layer being adjusted depending on the number of output classes. Consequently, the total number of parameters will vary only in the final layer.

Loss Function and Hyperparameters

After defining the network architecture, the following settings were chosen:
- **Loss** = *categorical_crossentropy*

- **Optimizer** = *ADAM*
- **Learning Rate** = 0.0001
- **Training Metric** = *accuracy*
- **Steps per epoch** = total number of samples divided by the batch size, *batch_size* = 64
- **Stopping criterion** = fixed number of epochs, set to 40

As anticipated, in some training cases the ***class_weight*** parameter is used, which assigns weights to each class to adjust the importance of their examples during the computation of the average loss. The weights are computed according to the following formula: $w_c = \frac{N}{K \times n_c}$, where w_c is the weight associated with class c, N is the total number of samples, K is the total number of classes, and n_c is the number of examples in class c.

Model Evaluation Metrics

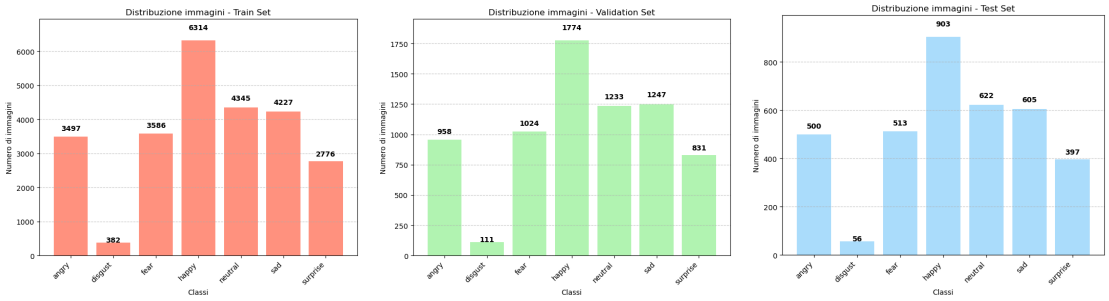
After the training phase, the model's performance was analyzed on the test data using the following metrics: **Confusion Matrix**, **Accuracy**, **Precision**, **ROC Curves and AUC**, and **Macro-Averaged ROC and AUC**. Additionally, **Saliency Maps** were used to visualize which parts of the input images were most relevant for the network in discriminating between classes.

Results

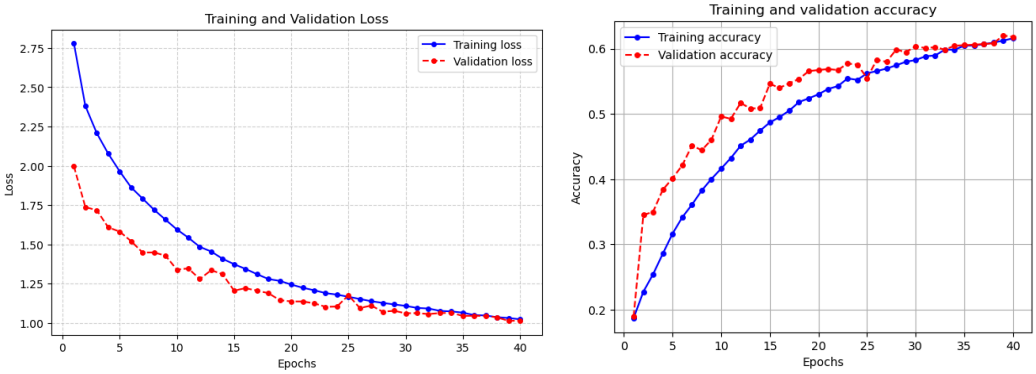
In the following sections, I will report the various results obtained by addressing the problem using different approaches. In some cases, certain "mistakes" were made intentionally (for instance, training with an unbalanced dataset) in order to compare those outcomes with results obtained using more refined approaches (such as training on a balanced dataset or using *class_weight*).

1) Full, Unbalanced Dataset — 7 Classes

As an initial attempt, I used the complete dataset with all 7 classes, without applying any balancing, to later compare its performance with the approaches discussed in subsequent sections. The dataset was split as follows:

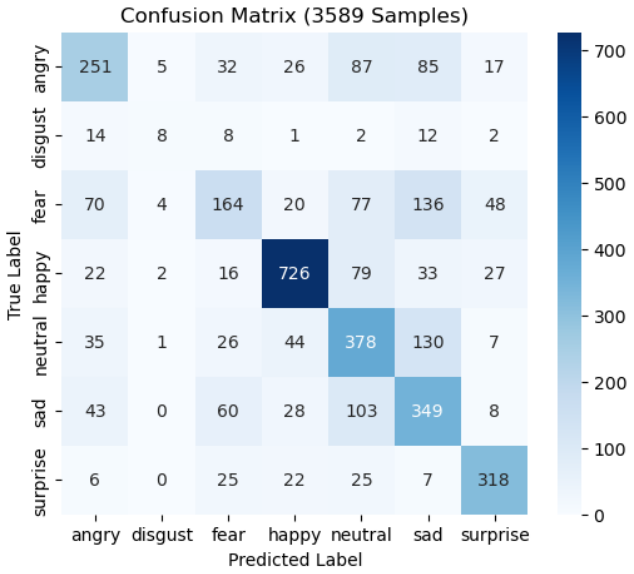


A strong imbalance can be immediately observed. In particular, it is expected that the *Disgust* class will not be effectively learned by the network, while the *Happy* class will likely perform much better. Proceeding with training, the resulting plots are as follows:



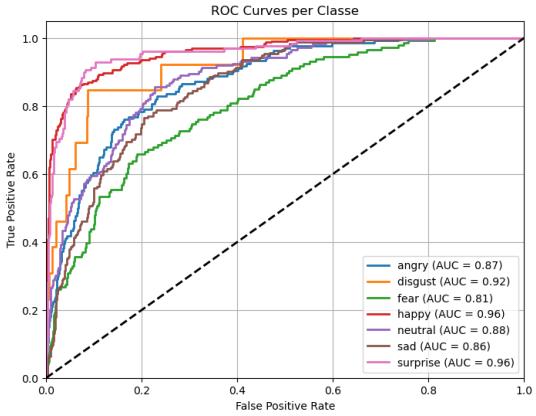
The final accuracy values obtained on the training and validation sets are $acc\_train = 0.62$ and $acc\_val = 0.62$.

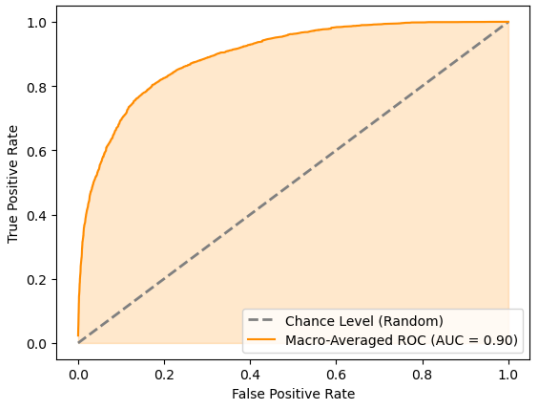
We now begin evaluating the model's performance on the Test Set using the metrics previously introduced. First, I examined the Confusion Matrix and also printed the precision, recall, and f_1-score values using the *classification_report* function. As expected, the class on which the network performs best is *Happy*, while its worst performance is observed for *Disgust*. Analyzing the Confusion Matrix, we can also observe that the network may confuse the classes *Sad*, *Neutral*, and *Fear*; this observation will be discussed further later on.




```
Classification Report (3589 Samples):
              precision    recall  f1-score   support

       angry       0.57      0.50      0.53       503
     disgust       0.40      0.17      0.24        47
        fear       0.50      0.32      0.39       519
       happy       0.84      0.80      0.82       905
     neutral       0.50      0.61      0.55       621
         sad       0.46      0.59      0.52       591
    surprise       0.74      0.79      0.77       403
```

As a final metric, I plotted the ROC curves for each class and the Macro-Averaged ROC, together with the corresponding AUC values. The results are reported in the legends of the following figures:

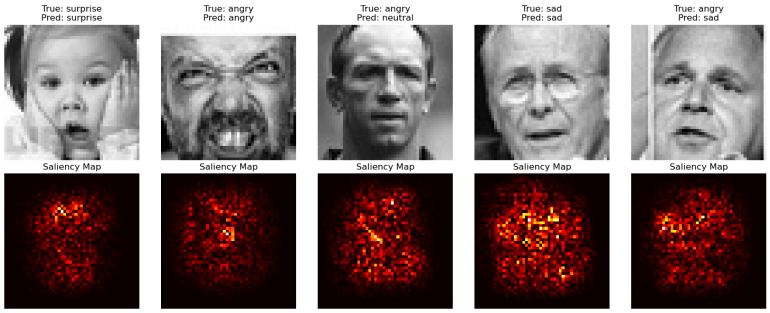




We can observe that the lowest areas under the curve correspond to the classes *Fear* and *Sad*, as anticipated. This suggests that the network is often uncertain when assigning those labels. This can also be confirmed from the confusion matrix, where many misclassified samples with *true_label* = *Fear* are predicted as *predicted_label* = *Sad*.

The global performance metrics on the test set are as follows: $accuracy = 0.61$ and $precision = 0.57$. These overall results are not particularly satisfactory; therefore, in the next sections, I will modify certain experimental conditions to explore ways to improve performance.

Finally, to verify that the network is focusing on the relevant parts of the images, I plotted the saliency maps for a random sample of 5 test images, indicating both the true and predicted labels.



Observation: In the following sections, to avoid redundancy, certain plots not essential for the analysis will be omitted, such as the accuracy-over-epochs plot, the Macro-Averaged ROC plot, and the saliency maps, except for specific cases of particular interest.

2) Balanced Dataset — 7 Classes

As a second attempt, I balanced the dataset while still including the minority class *Disgust*, by reducing the number of images in all other classes to match it. This resulted in each class containing 382 images in the training set, 111 in the validation set, and 56 in the test set. As expected, the network did not have enough examples to properly learn to distinguish among the classes, achieving a final validation accuracy of $acc\_val = 0.38$. Therefore, this approach does not warrant further analysis or detailed discussion of its results.

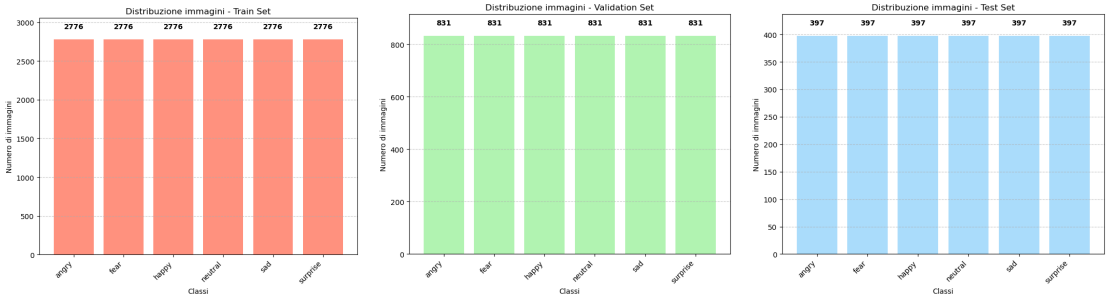
3) Full Dataset, Balanced via *class_weight* — 7 Classes

To begin improving the model's performance, I applied the *class_weight* function, which balances the contribution of images from classes with different sample sizes, as previously explained. The dataset distribution was the same as in Section 1).

However, the overall performance did not improve. The minority class received an excessively high weight in the loss computation compared to the others. As a consequence, the model attempted to minimize the error for that class at the expense of the others, resulting in an overall performance degradation.

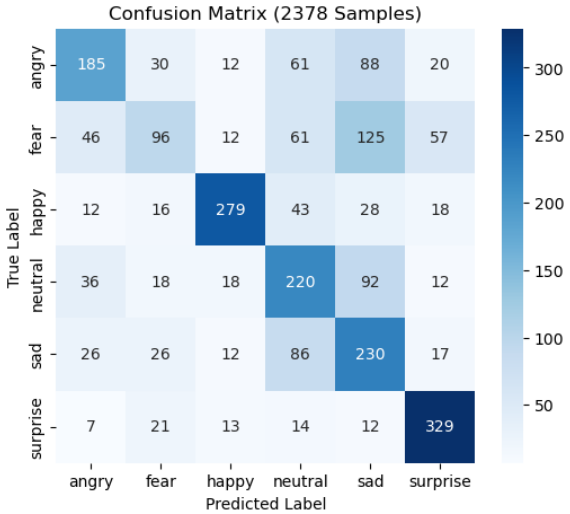
4) Dataset Without *Disgust* Class — Balanced, 6 Classes

At this stage, I decided to exclude the *Disgust* class, which contained very few samples, and balanced the remaining classes at the level of the *Surprise* class. The dataset was distributed as follows:

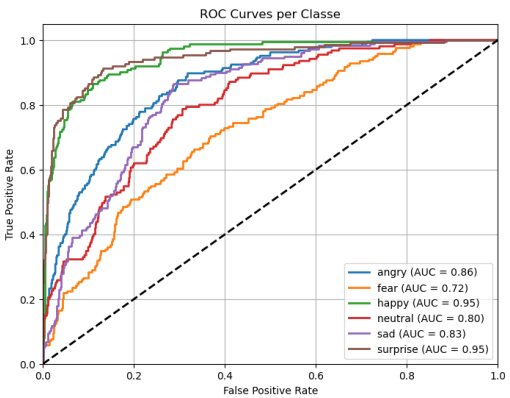


After training, the following plot shows the loss trend. The final accuracy values at the last epoch were $acc\_train = 0.56$ and $acc\_val = 0.57$.

Given that these results are reasonably similar to those of Section 1), it can be hypothesized that the network's difficulties were partially caused by the *Disgust* class. However, balancing the dataset by reducing the number of samples for other classes such as *Surprise* did not lead to improvements. This suggests that the challenge in achieving higher performance lies mainly in the intrinsic complexity of the task itself, which I will attempt to simplify in the next steps.

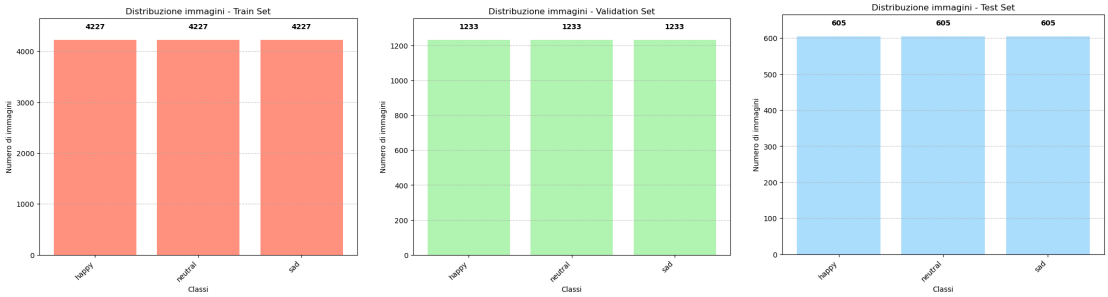



```
Classification Report (2378 Samples):
              precision    recall  f1-score   support

       angry       0.59      0.47      0.52       396
        fear       0.46      0.24      0.32       397
       happy       0.81      0.70      0.75       396
     neutral       0.45      0.56      0.50       396
         sad       0.40      0.58      0.47       397
    surprise       0.73      0.83      0.78       396
```

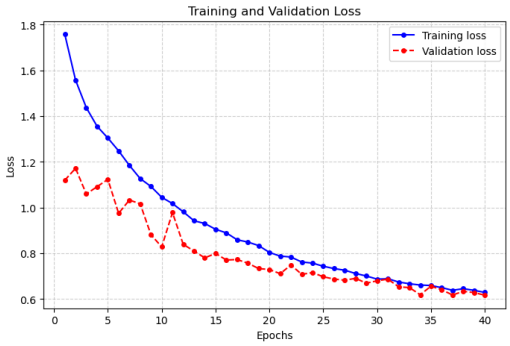


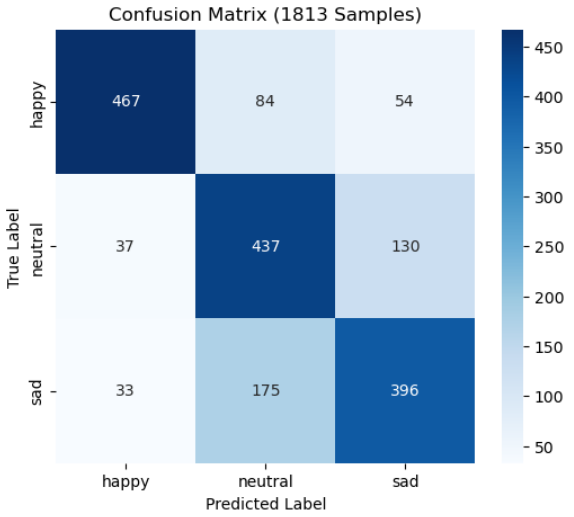
5) Dataset with Only *Happy, Neutral, Sad* Classes — Balanced, 3 Classes

To reduce the complexity of the task, I limited the dataset to three classes: *Happy*, *Sad*, and *Neutral*. The dataset is now distributed as follows:

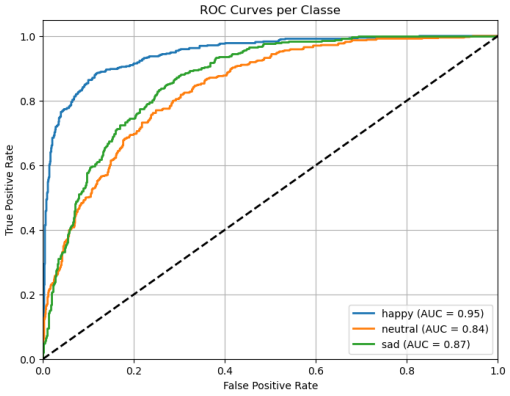


After training, the following plot shows the loss trend, while the final accuracy values at the last epoch were $acc\_train = 0.72$ and $acc\_val = 0.73$.
Subsequently, the performance metrics were computed.
It can be observed that the loss decreased and the accuracy increased compared to Section 1). Moreover, the precision improved for each class. However, the confusion matrix shows that the network often confuses the labels *Neutral* and *Sad*.
The overall performance on the test set is: $precision = 0.73$, $accuracy = 0.72$, and AUC (Macro-Averaged ROC) $= 0.88$.

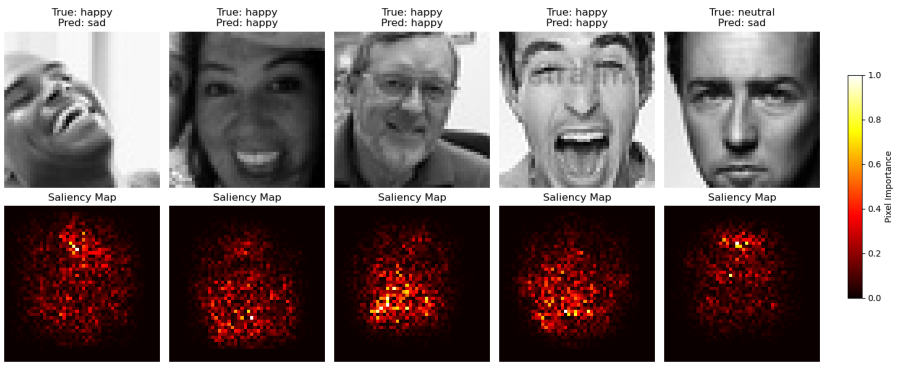





```
Classification Report (1813 Samples):
              precision    recall  f1-score   support

       happy       0.87      0.77      0.82       605
     neutral       0.63      0.72      0.67       604
         sad       0.68      0.66      0.67       604
```

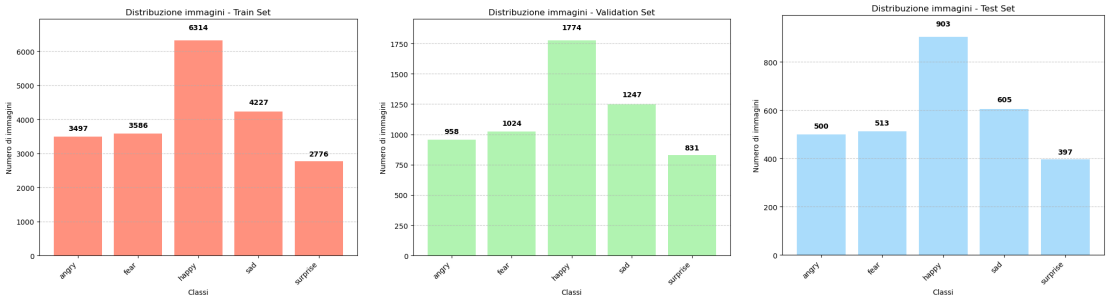


It is interesting to note that in the first and last image the network misclassifies the input by predicting *Sad*.

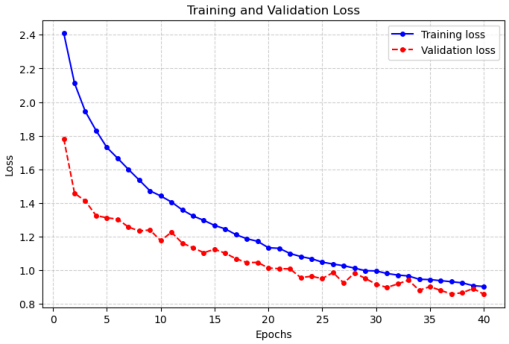


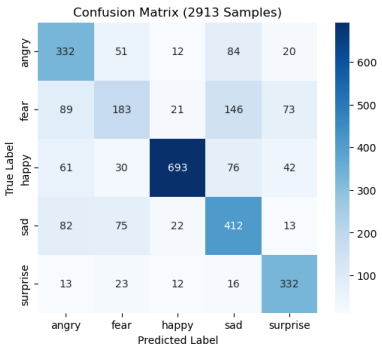
6) Dataset Without *Disgust* and *Neutral* — Balanced via *class_weight*, 5 Classes + Post-Training Binarization

Since *Neutral* appeared to be a confusing class for the network (as suggested both by visual inspection of the dataset and by the results in the previous section), I decided to retrain the model on 5 classes, removing *Disgust* and *Neutral*, and applying *class_weight* during training to balance the dataset dynamically. After training, I performed classification binarization, in order to compare these results with those obtained when the dataset was binarized prior to training. The dataset distribution is shown below:

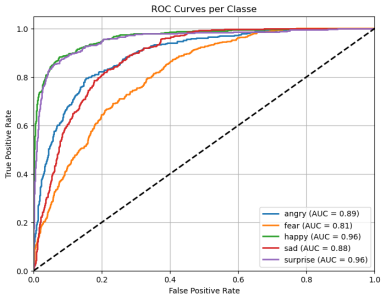


After training, the following plot shows the loss curve, while the final accuracy values at the last epoch were $acc\_train = 0.66$ and $acc\_val = 0.66$.
Subsequently, the evaluation metrics were computed.
The overall performance on the test set was: $precision = 0.65$, $accuracy = 0.67$, and AUC (Macro-Averaged ROC) $= 0.88$.
This represents the best result obtained when training the network with fine-grained classes. At this point, I wanted to analyze what happens when combining the output probabilities of multiple classes to transform the problem into a binary one.

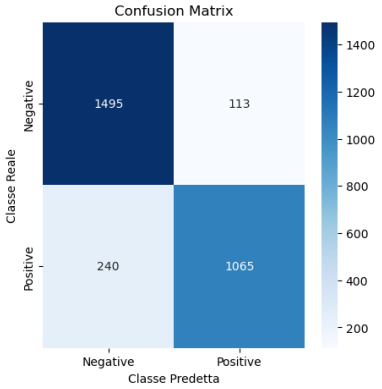




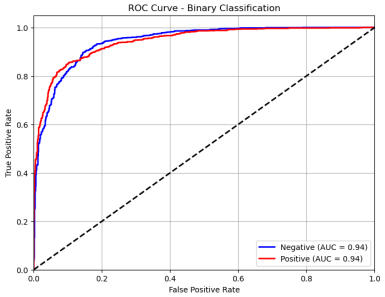

```
Classification Report (2913 Samples):
              precision    recall  f1-score   support

       angry       0.58      0.67      0.62       499
        fear       0.51      0.36      0.42       512
       happy       0.91      0.77      0.83       902
         sad       0.56      0.68      0.62       604
    surprise       0.69      0.84      0.76       396
```



I therefore summed the output probabilities of the classes *Happy* and *Surprise*, merging them under the label *Positive*, and grouped *Angry*, *Fear*, and *Sad* under the label *Negative*. I then set the decision threshold to 0.5 to determine which of the two classes each test image should belong to, and re-evaluated the model's performance:



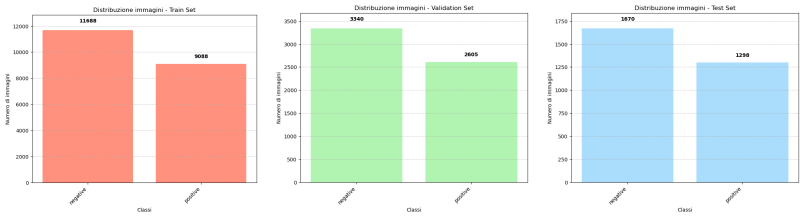

```
Classification Report (2913 Samples):
              precision    recall  f1-score   support

    Negative       0.86      0.93      0.89      1608
    Positive       0.90      0.82      0.86      1305
```



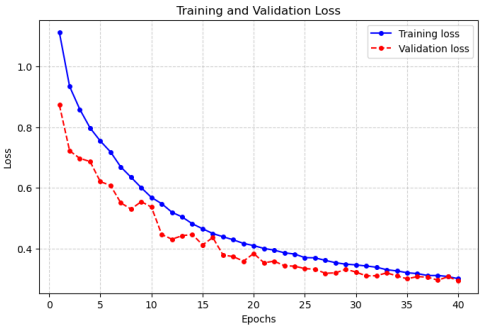
The overall performance on the test set was: *precision* = 0.90, *accuracy* = 0.87, and AUC (Macro-Averaged ROC) = 0.94.

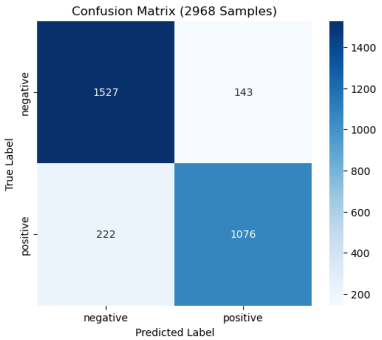
7) Dataset with Two Redefined Classes: *Negative* and *Positive*, Balanced via *class_weight*, 2 Classes

Next, I restructured the pre-processed dataset by merging the classes *Happy* and *Surprise* under the *Positive* label, and *Angry*, *Fear*, *Disgust*, and *Sad* under the *Negative* label, resulting in the following distribution:



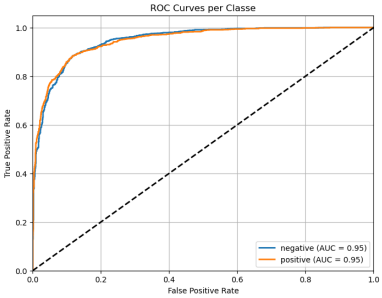
Since the classes were slightly imbalanced, I again employed the *class_weight* parameter. After training, the loss curve is shown below, and the final accuracy values at the last epoch were *acc_train* = 0.87 and *acc_val* = 0.88. The evaluation metrics were then computed. The overall performance on the test set was: *precision* = 0.87, *accuracy* = 0.87, and AUC (Macro-Averaged ROC) = 0.95. It can be observed that these results are very similar to those obtained in the previous section after binarizing the problem.



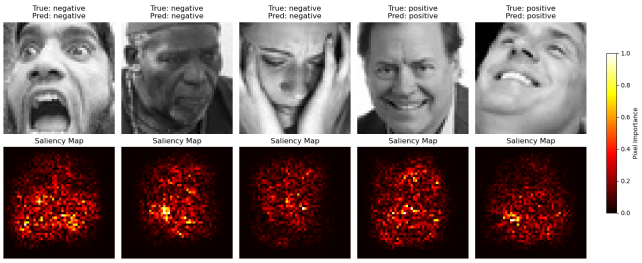


Classification Report (2968 Samples):

	precision	recall	f1-score	support
negative	0.87	0.91	0.89	1670
positive	0.88	0.83	0.85	1298



It is interesting to observe that the network is now able to correctly predict even on challenging samples such as the last image.



8) Dataset with Two Redefined Classes: *Negative* and *Positive*, Balanced, 2 Classes

I repeated the same procedure as in the previous section, but this time the dataset was balanced in order to verify whether any significant differences in performance could be observed. The obtained results are as follows:

- *loss_train* = 0.31 and *loss_val* = 0.31
- *acc_train* = 0.86 and *acc_val* = 0.87
- Overall performance on the test set: *precision* = 0.86, *accuracy* = 0.86, and AUC (Macro-Averaged ROC) = 0.94.

It can be observed that the results obtained are very similar to those in the previous section.

Classification Report (2968 Samples):

	precision	recall	f1-score	support
negative	0.87	0.91	0.89	1670
positive	0.88	0.83	0.85	1298