

Iterative Development of a Deep Learning Model for Facial Emotion Recognition

Giorgio Gobbin*

November 18, 2025

Abstract

This report details the end-to-end data science process for building a high-accuracy facial emotion classifier. Starting from the *Human Face Emotions* dataset, I performed an exploratory data analysis (EDA) that revealed a significant class imbalance. All computational experiments were conducted using the Kaggle cloud environment, leveraging an NVIDIA Tesla T4 GPU to accelerate model training. I then iteratively developed and evaluated multiple deep learning models, systematically comparing architectures (Custom CNN, MobileNetV2, EfficientNetB0, ResNet50V2) and balancing techniques (Class Weights, Data Augmentation). My findings demonstrate that geometric and photometric augmentations were detrimental to performance. The winning strategy combined Transfer Learning with a **ResNet50V2** backbone, **Class Weighting** to handle imbalance, and **Fine-Tuning**. This champion model (V13) achieved **87% accuracy** on the internal validation set and a robust **91% accuracy** on an external, unseen test dataset (FER-2013), validating its generalization capabilities.

*Student/University of Pisa

Contents

1	Introduction	3
1.1	Problem Statement	3
1.2	Dataset and Exploration	3
2	Methodology: An Iterative Approach	3
2.1	Experiment 1: Baseline Custom CNN (V1)	3
2.2	Experiment 2: The Failure of Augmentation (V8-V11)	4
2.3	Experiment 3: Architecture Comparison (V5, V7, V13)	4
2.3.1	MobileNetV2 (V5)	4
2.3.2	EfficientNetB0 (V7)	5
2.3.3	ResNet50V2 (V12-V13)	5
3	The Champion Model (V13): ResNet50V2	5
3.1	Training Strategy	5
3.2	Performance on Internal Validation (87%)	5
4	External Validation Real-World Limitations	7
4.1	Performance on External Test Set (91%)	7
4.2	Limitations: The Out-of-Distribution (OOD) Problem	7
5	Conclusion	8
A	Key Code Snippets	9
A.1	V13 Model Definition (Keras)	9
A.2	Single Image Prediction Function	9

1 Introduction

1.1 Problem Statement

The goal of this project is to develop a robust machine learning model capable of accurately classifying human facial expressions into five distinct categories: Angry, Fear, Happy, Sad, and Surprise. This task is a classic computer vision challenge with applications in human-computer interaction, mental health analysis, and user experience research.

1.2 Dataset and Exploration

The primary dataset used for training was the *Human Face Emotions* dataset from Kaggle, located at [/kaggle/input/human-face-emotions/Data](https://www.kaggle.com/c/human-face-emotions).

An initial Exploratory Data Analysis (EDA) revealed a significant class imbalance. The 'Happy' class was heavily over-represented, while classes like 'Surprise' and 'Fear' were under-represented. This imbalance was a critical challenge that needed to be addressed in the modeling strategy.

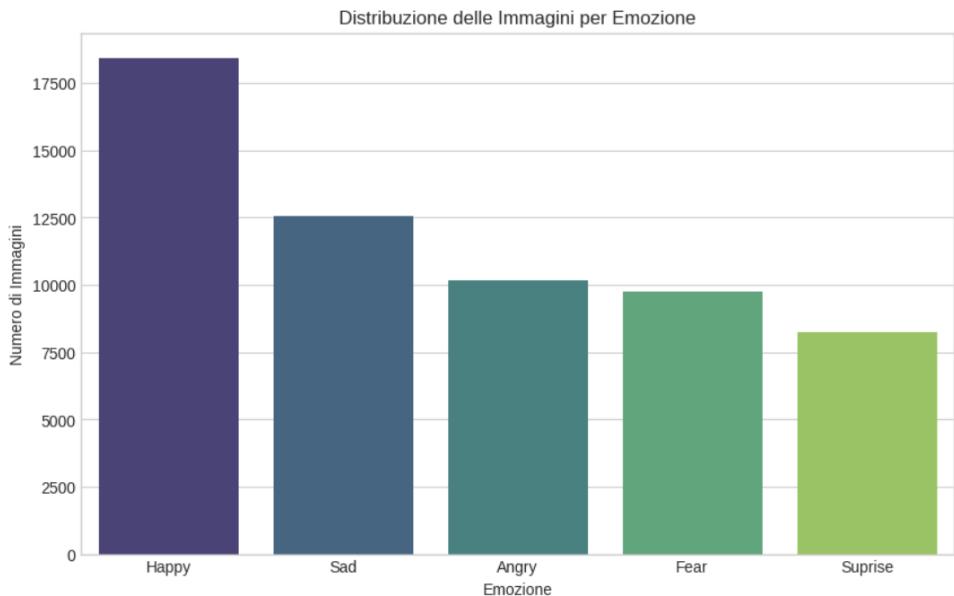


Figure 1: Class distribution of the training dataset, showing significant imbalance.

2 Methodology: An Iterative Approach

I followed a rigorous, iterative data science methodology, treating each model (V1-V13) as a controlled experiment. All training sessions were executed on a Kaggle Notebook environment equipped with an **NVIDIA Tesla T4 GPU (16GB VRAM)** and 2-core Intel Xeon CPU, ensuring efficient processing of the image datasets.

2.1 Experiment 1: Baseline Custom CNN (V1)

The first baseline established was a simple custom Convolutional Neural Network (CNN). While it achieved $\sim 68\%$ accuracy, the validation loss diverged significantly from the training loss, indicating severe overfitting.

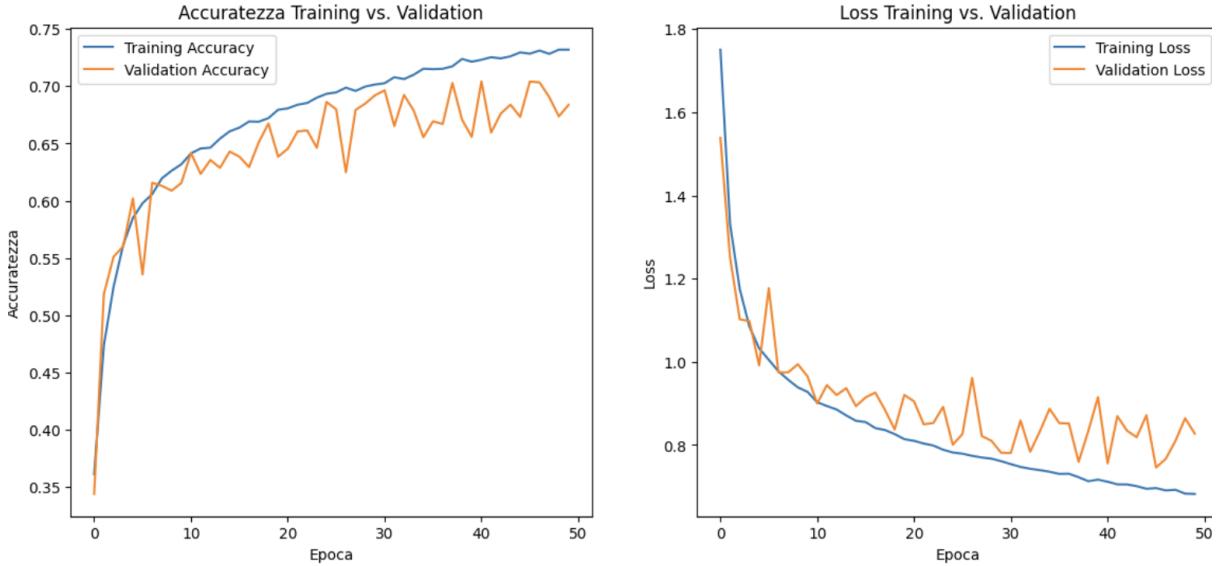


Figure 2: Training vs. Validation metrics for the Baseline Model (V1). The divergence between training loss (decreasing) and validation loss (increasing) clearly demonstrates overfitting.

2.2 Experiment 2: The Failure of Augmentation (V8-V11)

A primary hypothesis was that data augmentation would improve generalization. I tested two types:

- **Geometric (V8, V9):** Rotations and zooms.
- **Photometric (V10, V11):** Brightness, contrast, and noise.

As shown in Table 1, both augmentation strategies caused a catastrophic drop in performance, especially when combined with class weights. I concluded that for this dataset, augmentation added noise that confused the model rather than helping it generalize.

Table 1: Comparison of Augmentation Strategies (Internal Validation Results)

Model	Configuration	Accuracy
V4	MobileNetV2 + Class Weights (Baseline)	76%
V8	MobileNetV2 + Class Weights + Geometric Aug.	56%
V9	MobileNetV2 + Geometric Aug. (No Weights)	57%
V11	MobileNetV2 + Class Weights + Photometric Aug.	49%

2.3 Experiment 3: Architecture Comparison (V5, V7, V13)

I tested three different Transfer Learning architectures, applying the most stable strategy: Class Weighting (to fix imbalance) and Fine-Tuning.

2.3.1 MobileNetV2 (V5)

This lightweight model served as the second, much stronger baseline, achieving **79% accuracy**. It performed well but struggled with the 'Fear' class (0.69 F1-score).

2.3.2 EfficientNetB0 (V7)

This modern architecture performed worse than the baseline, reaching only 70% accuracy after fine-tuning.

2.3.3 ResNet50V2 (V12-V13)

This architecture proved to be the clear winner. The combination of ResNet's deep residual features, class weighting, and fine-tuning yielded the best performance by a significant margin.

3 The Champion Model (V13): ResNet50V2

The champion model (V13) was built using the ResNet50V2 backbone, trained in two phases.

3.1 Training Strategy

1. **Phase 1 (Head Training):** I froze the ResNet backbone and trained only the top classification layers (GlobalAveragePooling, Dense, Dropout) for 30 epochs, using a learning rate of 0.0001 and the calculated `class_weight` dictionary.
2. **Phase 2 (Fine-Tuning):** I unfroze the top 50 layers (from 140 onwards) of the ResNet backbone and continued training with a very low learning rate of 1e-5. `EarlyStopping` (patience=3) was used to halt training at the optimal point.

3.2 Performance on Internal Validation (87%)

The V13 model achieved an accuracy of 87% on the internal validation set. The accuracy and loss curves show a stable training process, with the fine-tuning phase providing a final performance boost.

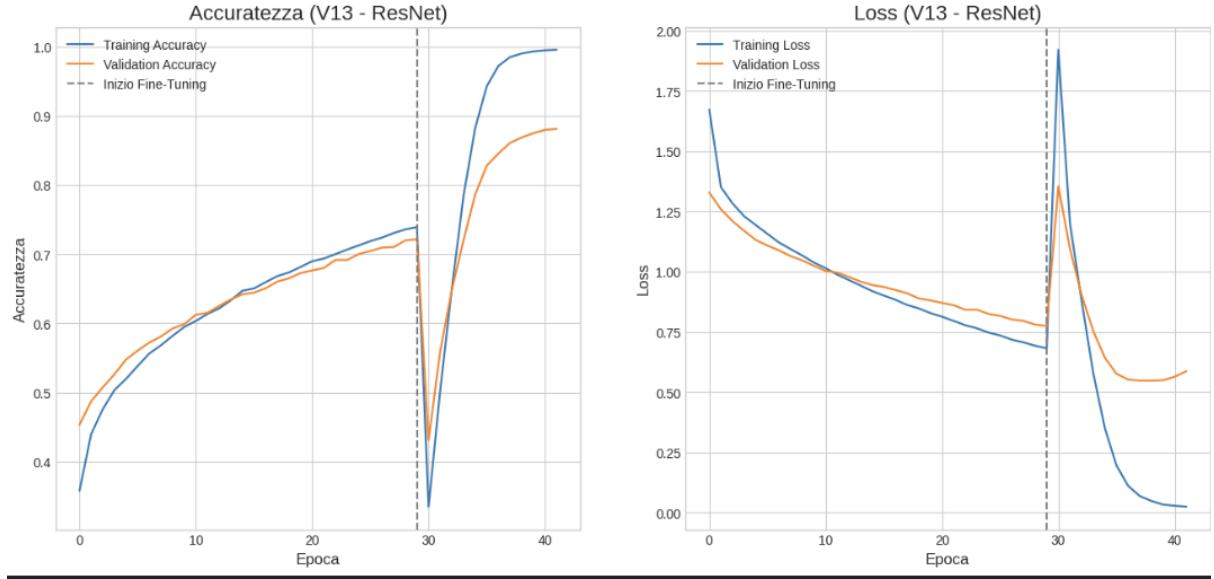


Figure 3: Accuracy and Loss curves for V13 (ResNet50V2), showing the start of fine-tuning.

The classification report (Table 2) shows excellent, balanced performance across all classes, including the previously difficult 'Fear' class.

Table 2: Classification Report for V13 (ResNet) on Internal Validation Set

	precision	recall	f1-score	support
Angry	0.82	0.86	0.84	2058
Fear	0.83	0.80	0.81	1981
Happy	0.93	0.90	0.92	3648
Sad	0.82	0.86	0.84	2503
Surprise	0.92	0.90	0.91	1629
accuracy			0.87	11819
macro avg	0.86	0.86	0.86	11819
weighted avg	0.87	0.87	0.87	11819

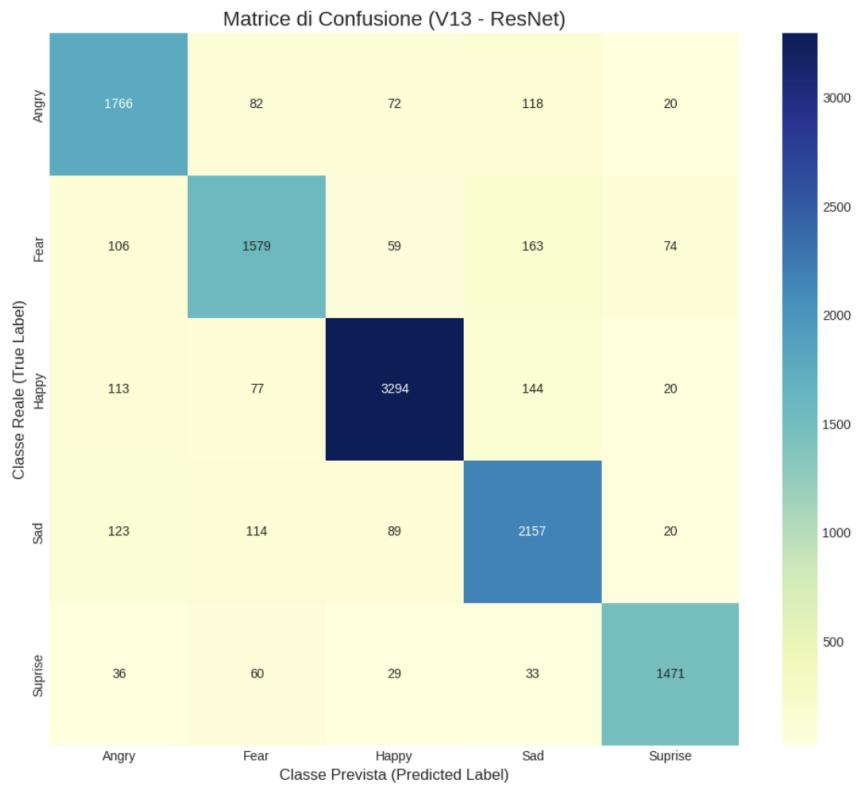


Figure 4: Confusion Matrix for V13 (ResNet) on Internal Validation Set.

4 External Validation Real-World Limitations

4.1 Performance on External Test Set (91%)

To verify the model's true generalization power, I tested it on an external, unseen dataset: the **FER-2013** dataset. The model achieved an outstanding 91% accuracy, higher than its internal validation score. This confirms the model is extremely robust and not overfitted to the original dataset.

Table 3: Classification Report for V13 (ResNet) on External Test Set (FER-2013)

	precision	recall	f1-score	support
Angry	0.87	0.92	0.89	958
Fear	0.90	0.79	0.84	1024
Happy	0.96	0.93	0.95	1774
Sad	0.86	0.94	0.90	1247
Surprise	0.93	0.95	0.94	831
accuracy			0.91	5834
macro avg	0.91	0.91	0.91	5834
weighted avg	0.91	0.91	0.91	5834

4.2 Limitations: The Out-of-Distribution (OOD) Problem

Despite its high accuracy on test data, the model exhibited high-confidence failures when tested on real-world photos (e.g., from a webcam). These images represent "Out-of-Distribution" (OOD) data. The model was trained on low-resolution, pre-cropped, grayscale-like images, and it struggled to interpret high-resolution photos with complex backgrounds and lighting.

A key failure case was the misclassification of 'Surprise' as 'Angry' (Figure 5). The model likely over-weighted the "open mouth" feature and, confused by shadows, misclassified the eyes.



Figure 5: OOD Failure Case: A real-world 'Surprise' photo confidently misclassified as 'Angry' (94.01%).

5 Conclusion

Through a systematic, iterative process, I successfully developed a high-performance emotion classifier (V13) with 87% internal and 91% external accuracy. I determined that the optimal strategy was a **ResNet50V2** backbone, balanced with **Class Weights**, and optimized with **Fine-Tuning**.

I also concluded that common data augmentation techniques were detrimental to this specific task. The model's primary limitation is its vulnerability to OOD data, which could be addressed in future work by re-training the model on a more diverse dataset that includes these "failed" real-world images (Hard Negative Mining).

A Key Code Snippets

A.1 V13 Model Definition (Keras)

```
1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Rescaling, GlobalAveragePooling2D, Dense,
4     Dropout
5 from tensorflow.keras.applications import ResNet50V2
6 from tensorflow.keras.optimizers import Adam
7
8 # Define input shape and num_classes
9 input_shape = (96, 96, 3)
10 num_classes = 5
11
12 # 1. Create the base model
13 base_model_v12 = ResNet50V2(
14     input_shape=input_shape,
15     include_top=False,
16     weights='imagenet'
17 )
18 base_model_v12.trainable = False # Start as frozen
19
20 # 2. Build the full model
21 model_v12 = Sequential([
22     Rescaling(1./127.5, offset=-1, input_shape=input_shape),
23     base_model_v12,
24     GlobalAveragePooling2D(),
25     Dense(128, activation='relu'),
26     Dropout(0.5),
27     Dense(num_classes, activation='softmax')
28 ])
29
30 # 3. Compile for head training
31 model_v12.compile(
32     optimizer=Adam(learning_rate=0.0001),
33     loss='categorical_crossentropy',
34     metrics=['accuracy']
35 )
36 # (After head training, model is unfrozen and re-compiled for fine-tuning)
```

Listing 1: Python (Keras) code for defining the V13 model architecture.

A.2 Single Image Prediction Function

```
1 from tensorflow.keras.utils import load_img, img_to_array
2 import numpy as np
3
4 # Load the champion model
5 model = load_model('emotion_model_V13_final_87pct.keras')
6 CLASS_NAMES = ['Angry', 'Fear', 'Happy', 'Sad', 'Surprise']
7 IMG_HEIGHT = 96
8 IMG_WIDTH = 96
9
10 def predict_single_image(image_path):
11     # a. Load and resize the image
12     img = load_img(
13         image_path,
14         target_size=(IMG_HEIGHT, IMG_WIDTH),
15         color_mode='rgb',
16     )
```

```
17
18 # b. Convert image to array
19 img_array = img_to_array(img)
20
21 # c. Create a batch of 1
22 img_batch = np.expand_dims(img_array, axis=0)
23
24 # d. Predict (Rescaling is built into the model)
25 predictions = model.predict(img_batch)
26 scores = predictions[0]
27
28 # e. Format results
29 score_list = sorted(zip(CLASS_NAMES, scores), key=lambda x: x[1], reverse=True)
30
31 return score_list
32
33 # Example usage:
34 # results = predict_single_image('my_photo.jpg')
35 # print(results)
```

Listing 2: Python function to preprocess and predict a single new image.