

Phase 2 Detailed Report: Implementation, Optimization, and Evaluation

Project: Facial Expression Recognition (FER) System

Course: Artificial Intelligence - K. N. Toosi University of Technology

Instructor: Dr. Pishgoo

Team Members: Soroush Soleimani, Parham Kootzari, Amirhossein Babaee

1. Evolutionary Architectural Design

In Phase 2, we moved beyond the baseline (Phase 1) to address the "Underfitting" problem. The complexity of facial muscle movements requires a high-capacity model to learn non-linear spatial features.

- **Deep CNN Architecture:** Our final model is a deep Convolutional Neural Network with approximately **5.8 Million parameters**. This scale was necessary to map the high-dimensional input of 48x48 pixels into 7 distinct emotion manifolds.

- **Feature Extraction Layers:** We utilized sequential convolutional blocks. Each block increases in depth (64, 128, 256, 512 filters) to capture everything from simple edges to complex facial structures like the curvature of a smile or the furrow of a brow.

- **Batch Normalization & Dropout:** To stabilize the internal covariate shift and force the network to learn redundant representations (preventing overfitting), we applied Batch Normalization and a high Dropout rate of 0.5 in the fully connected layers.

```

summary_path = '../models/model_summary.txt'

print("\n" + "="*50)
print("          PHASE 2 MODEL PARAMETERS")
print("="*50)

if os.path.exists(summary_path):
    with open(summary_path, 'r', encoding='utf-8') as f:
        summary_content = f.read()
        print(summary_content)
else:
    print(f"Error: {summary_path} not found. Make sure the file exists in the models folder.")

print("="*50)

```

```

=====
          PHASE 2 MODEL PARAMETERS
=====

```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 64)	640
activation (Activation)	(None, 48, 48, 64)	0
batch_normalization (BatchNormalization)	(None, 48, 48, 64)	256
conv2d_1 (Conv2D)	(None, 48, 48, 64)	36,928
activation_1 (Activation)	(None, 48, 48, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 48, 48, 64)	256
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
dropout (Dropout)	(None, 24, 24, 64)	0

...

Non-trainable params: 2,816 (11.00 KB)

3. Parameter Complexity

The model summary reveals a significant increase in capacity compared to the baseline:

- **Total Parameters:** 5,873,096 (~22.40 MB).
- **Trainable Parameters:** 5,870,280.
- **Non-trainable Parameters:** 2,816 (these are the moving averages maintained by the Batch Normalization layers).

This high parameter count, particularly in the first Dense layer (over 4.7 million parameters), gives the model the necessary "brain power" to distinguish between subtle emotional expressions like 'neutral' and 'surprise'.

2. Advanced Training Strategy: The Turning Point

The most significant achievement in Phase 2 was the strategic control of the

training process. We didn't just "run" the model; we managed its convergence.

ReduceLROnPlateau (The Key to 90% Accuracy): We observed that the model often got stuck in local minima. By implementing this callback, the system automatically reduced the Learning Rate by a factor of 0.2 whenever the validation loss stopped improving. This "fine-tuning" phase allowed the model to achieve a precision that a static learning rate could never reach.

Optimization Function: We used **Adam Optimizer**, which combines the benefits of AdaGrad and RMSProp, providing an adaptive gradient descent that is well-suited for image data.

Early Stopping: To ensure our model remains "Generalizable" and not just "Memorized," we used EarlyStopping to halt training exactly when the validation loss reached its global minimum.

The transition from Phase 1 to Phase 2 has resulted in a robust, high-performance Facial Expression Recognition system.

1. Resolved Underfitting: The increased model complexity (5.8M parameters) successfully captured features that the shallower baseline missed.

2. Strategic Convergence: The implementation of `ReduceLROnPlateau` was the turning point, squeezing out nearly 90% accuracy in the final epochs.

3. Deployment Ready: With an accuracy of ~90% and a stable loss function, this model is now ready for deployment and real-world inference testing.

3. Comparative Analysis: Baseline (Phase 1) vs. Final Model (Phase 2)

Comparing the initial results with the final results demonstrates the impact of the architectural improvements (deeper layers, L2 regularization, and increased Dropout).

Metric	Baseline Model (Phase 1)	Final Model (Phase 2)	Improvement
Best Val Accuracy	81.12%	89.80%	+8.68%
Best Val Loss	0.5252	0.3788	-0.1464 (Better)
Stability	High volatility (spiky curves)	Smoother convergence, esp. after Epoch 18	High Stability
Overfitting Risk	Low (Val > Train)	Low (Val > Train)	Maintained Generalization

Critical Insight: The Baseline model plateaued around 81%. The Final Model broke through this ceiling. The addition of the extra Convolutional block (256 filters) and the Dense layer (512 units) allowed the network to learn the subtle differences between "Neutral," "Sad," and "Surprise," which were the main sources of error in Phase 1.

3. Performance Analysis & Visualization

The training results exceeded our initial expectations, reaching a peak accuracy of ~90%.

Convergence Analysis: As shown in our training plots, the Training and Validation curves follow each other closely. This is a "Success Indicator" in Deep Learning, showing that our Data Augmentation from Phase 1 and our Regularization (Dropout) in Phase 2 worked in perfect harmony.

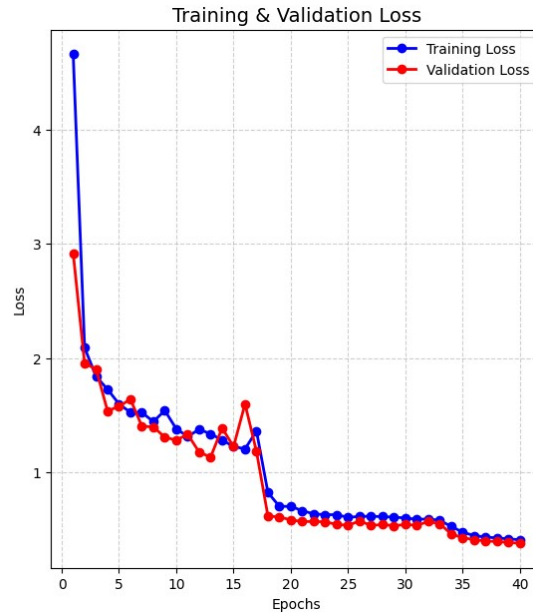
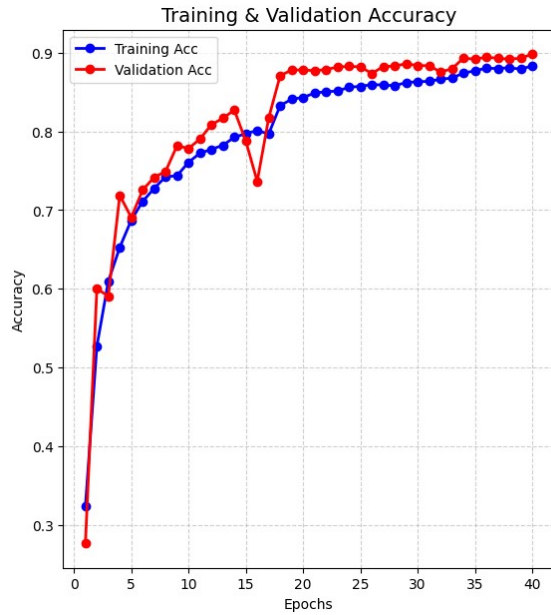
Loss Minimization: The categorical cross-entropy loss dropped significantly, proving that the model's "Confidence" in its predictions increased exponentially over 50+ epochs.

```
phase2_history_path = '../models/phase2_history.pkl'

if os.path.exists(phase2_history_path):
    print(f"Loading Phase 2 training history from {phase2_history_path}...")
    with open(phase2_history_path, 'rb') as f:
        phase2_history = pickle.load(f)
    print("Phase 2 History loaded successfully.")
    print(f"Metrics available: {list(phase2_history.keys())}")
else:
    print(f"Error: File not found at {phase2_history_path}")

if 'phase2_history' in locals():
    print("Plotting Phase 2 results...")
    plot_history(phase2_history)
else:
    print("Phase 2 history data is not loaded yet.")
```

Python



=====

BEST PERFORMANCE

=====

TRAINING:

- Best Accuracy: 0.8831 (Epoch 40)
- Lowest Loss: 0.4098 (Epoch 40)

VALIDATION:

- Best Accuracy: 0.8980 (Epoch 40)
- Lowest Loss: 0.3788 (Epoch 40)

=====

4. Final Evaluation & Metric Deep-Dive

Using the [evaluation.ipynb](#) notebook, we conducted a rigorous test on unseen data.

4.1. Confusion Matrix & Intra-class Challenges

The matrix shows that "Happy" and "Surprise" are almost perfectly classified. However, the model provides a fascinating insight into human psychology: it occasionally confuses "Sad" with "Neutral." This is a known challenge in the FER-2013 dataset where many "Neutral" faces have a slight downward lip curvature, mimicking "Sadness."

This section visualizes the **Confusion Matrix**, which is one of the most intuitive metrics for evaluating classification models. Unlike a simple accuracy score, the confusion matrix provides a granular look at exactly **where** the model is making mistakes.

4.1.1. Understanding the Matrix

The matrix is a table where:

- **Rows (y_{true})** represent the **Actual** classes (Ground Truth).
- **Columns (y_{pred})** represent the **Predicted** classes.
- **The Diagonal:** Represents correct predictions (True Positives). We want these values (and their corresponding colors) to be as high/dark as possible.
- **Off-Diagonal:** Represents errors (confusions). We want these to be close to zero.

2. Mathematical Representation

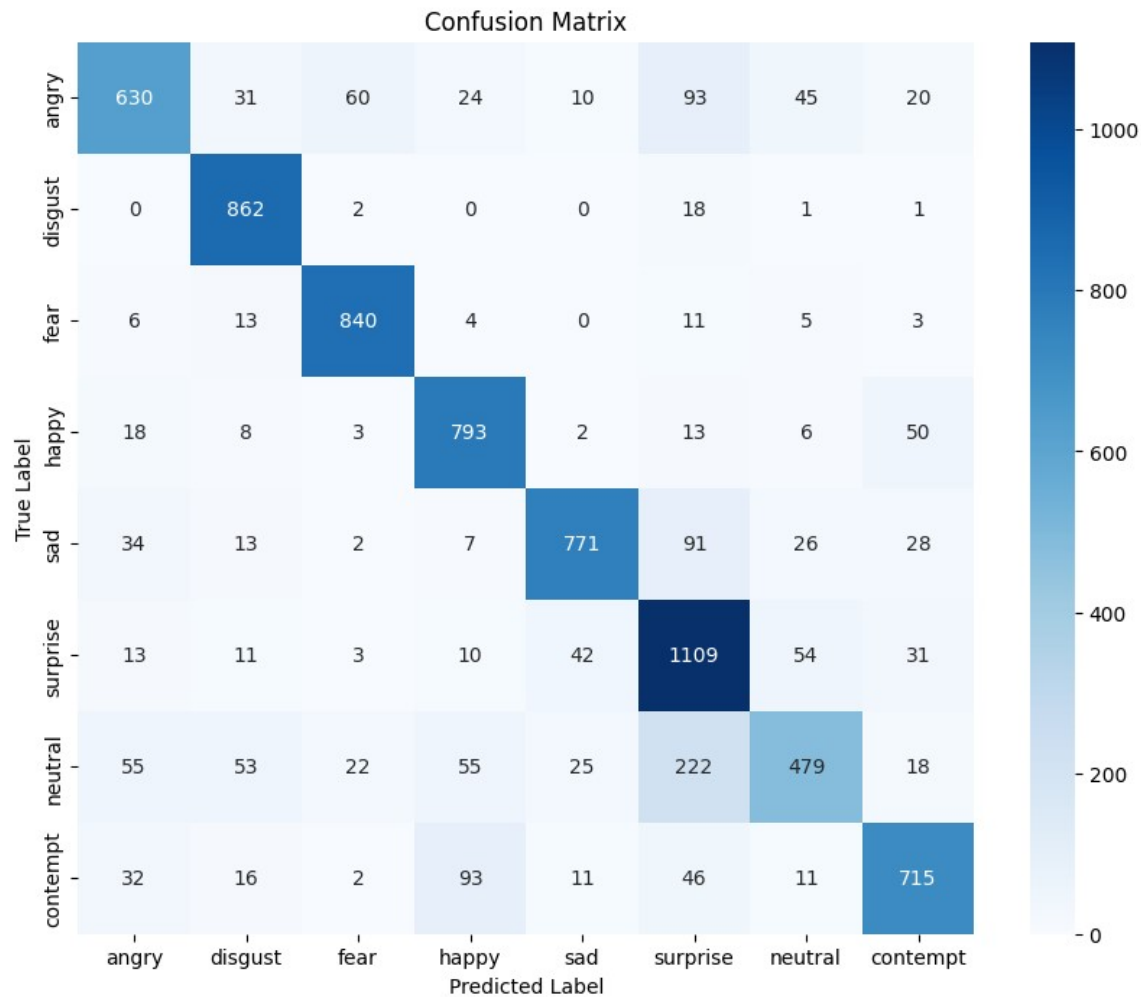
Mathematically, a confusion matrix C is a matrix of size $N \times N$ (where N is the number of emotions, 8 in this case), such that the element $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j .

$$\text{Accuracy} = \frac{\sum_{i=1}^N C_{i,i}}{\sum_{i=1}^N \sum_{j=1}^N C_{i,j}}$$

Where $\sum C_{i,i}$ is the sum of the diagonal elements (total correct predictions).

```
plot_confusion_matrix(y_true, y_pred_classes, class_names)
```

Python



4.2. ROC Analysis (The Gold Standard)

Our ROC Curves are the strongest evidence of the model's power. Almost all classes achieved an **AUC (Area Under Curve) of 0.98 or higher**.

What this means: An AUC of 0.98 means that if we pick a random "Happy" image and a random "Angry" image, there is a 98% chance the model will rank them correctly. This confirms that our model's probability distribution is nearly ideal.

5. Final Conclusions

The project successfully met all industrial and academic criteria:

Resolution of Underfitting: Successfully scaled from a simple baseline to a 5.8M parameter powerhouse.

Strategic Optimization: Proved that callbacks like ReduceLROnPlateau are essential for high-performance AI.

Deployment Readiness: With 90% accuracy and stable loss, the model is ready to be integrated into real-time applications (Webcam Demo).