Facial Emotion Recognition: A Custom CNN Approach

Dinar Orazgaliyev 7056420, Mestay Murzabayev 7047290

High-Level Computer Vision, Saarland University

github.com/dinar-orazgaliyev/fer_project

July 23, 2025

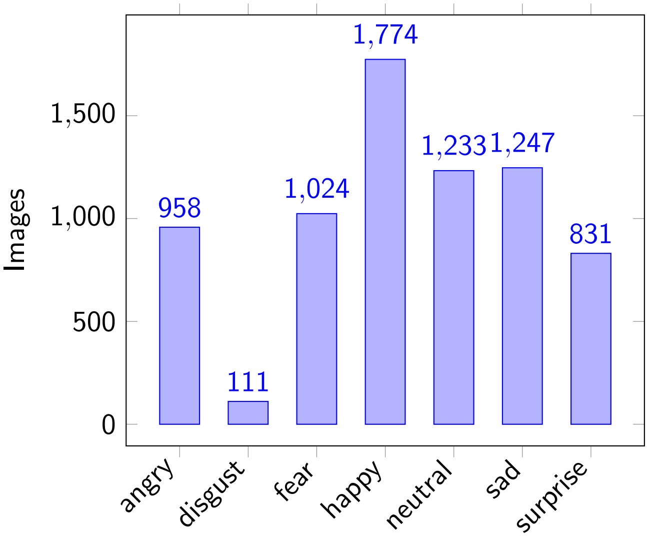
Outline

The Challenge: Recognizing Emotions from Faces

- **Goal:** To build and train a model to classify facial expressions into one of seven categories (Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral).
- **Dataset:** FER-2013
  - **Input:** 48x48 grayscale images.
  - **Challenges:** Low resolution, noisy data (poor lighting, obstructions), and class imbalance.

The Challenge: Recognizing Emotions from Faces

Dataset Analysis: A Major Challenge



Our Approach: A Two-Pronged Strategy

- **1: Build from Scratch**
  - Design and train a custom Convolutional Neural Network (CNN).
  - **Goal:** Establish a strong performance baseline and understand the problem's complexity.
- **2: Fine-Tune a Pre-trained Model**
  - Adapt a powerful, pre-trained ResNet-18 model.
  - **Question:** Can leveraging features learned from ImageNet yield better performance for FER-2013?

Architecture: Custom VGG-style CNN

Key Features:

- **Deep & Narrow:** Multiple blocks of stacked 3x3 convolutions.
- **Progressive Deepening:** Channel width increases (64 $\rightarrow$ 128 $\rightarrow$ 256 $\rightarrow$ 512) as spatial dimensions decrease.
- **Heavy Regularization:**
  - `BatchNorm` after every convolution for stability.
  - `Dropout(p=0.3)` in convolutional blocks.
  - `Dropout(p=0.5)` in the final classifier to prevent overfitting.

Architecture: Custom VGG-style CNN

4 blocks with a pair of convolution layers each

```
# --- Conv Block 1: 48x48 -> 24x24 ---
# in_channels=1, out_channels=64
nn.Conv2d(self.input_channels, 64, kernel_size=3, padding=1),
nn.BatchNorm2d(64),
nn.ReLU(inplace=True),
nn.Conv2d(64, 64, kernel_size=3, padding=1),
nn.BatchNorm2d(64),
nn.ReLU(inplace=True),
nn.MaxPool2d(kernel_size=2, stride=2),
nn.Dropout(self.dropout),
```

Architecture: Custom VGG-style CNN

1 classifier block with 3 linear layers

```
# After the conv layers, the feature map is (B, 512, 3, 3).
# We need to calculate the flattened size for the linear layer.
# Flattened size = 512 * 3 * 3 = 4608
self.classifier = nn.Sequential(
    nn.Linear(4608, 512),
    nn.BatchNorm1d(512),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),

    nn.Linear(512, 256),
    nn.BatchNorm1d(256),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),

    nn.Linear(256, self.num_classes)
)
```

Data Preprocessing

- Image preprocessing
- Label smoothing
  [0.02, 0.9, 0.02, 0.02...] instead of [0, 1, 0, 0, ...]

Data Preprocessing

Image Preprocessing

```python
transform = transforms.Compose(
    [   transforms.ToPILImage(),
        transforms.RandomHorizontalFlip(p=0.5),  # Flip images horizontally
        transforms.ColorJitter(brightness=0.2, contrast=0.2),  # Adjust brightness/contrast
        transforms.RandomRotation(degrees=10),  # Slight random rotation
        transforms.RandomCrop(48, padding=4),  # Random crop with padding
        transforms.ToTensor(),  # Convert to tensor (if not already)
        transforms.RandomErasing(p=0.5, scale=(0.02, 0.2), ratio=(0.3, 3.3), value=0), # RandomErasing is applied to the tensor
        transforms.Normalize(mean=[0.5], std=[0.5]),  # Normalize
    ]
)
```

Data Augmentation: Building a Robust Model

To prevent overfitting and improve generalization, a strong data augmentation pipeline was essential. The following transforms were applied to each training image:

Geometric Augmentations:

- `RandomHorizontalFlip`
  - Simulates viewing faces from different sides.
- `RandomRotation`
  - Accounts for slight head tilts ($\pm 10^\circ$).
- `RandomCrop`
  - Simulates small zoom and translation effects.

Data Augmentation: Building a Robust Mode

Photometric & Regularization:

- `ColorJitter`
  - Simulates various lighting conditions (brightness/contrast).
- `RandomErasing`
  - A key regularization technique.
  - Forces the model to learn from incomplete features (e.g., recognize emotion from eyes if the mouth is occluded).

Normalization:

- Images were normalized with a mean and standard deviation of 0.5, suitable for a model trained from scratch on grayscale data.

Training Details

- **Focal loss** as a loss function. Helps to focus training on hard, misclassified examples, while down-weighting the influence of easy, well-classified examples
- **Adam optimizer** with default learning rate and weight decay parameters

Training Details

Focal loss

```python
class FocalLoss(nn.Module):
    def __init__(self, weight=None, gamma=2.0, reduction='mean'):
        super().__init__()
        self.gamma = gamma
        self.weight = weight
        self.reduction = reduction

    def forward(self, input, target):
        log_prob = F.log_softmax(input, dim=-1)
        ce_loss = -log_prob.gather(1, target.unsqueeze(1)).squeeze()
        focal_loss = (1 - torch.exp(-ce_loss)) ** self.gamma * ce_loss

        if self.weight is not None:
            focal_loss = focal_loss * self.weight[target]

        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        else:
            return focal_loss
```

Custom CNN: Results & Learnings

61%

Peak Validation Accuracy

Key Learnings:

- A well-structured custom CNN can effectively learn the task.
- Aggressive data augmentation and regularization can be helpful.

The Promise of Transfer Learning

Why ResNet-18?

- Proven, powerful architecture.
- Pre-trained on ImageNet, providing a rich "dictionary" of low-level features (edges, textures, shapes).

The Modification:

- Replaced the final fully-connected layer with a custom classifier head (`num_filters -> 256 -> 7`).
- **Initial Expectation:** This should easily surpass our 61% baseline.

The Great Challenge: Overfitting

Initial Results:

- **Training Accuracy:** around 65%
- **Validation Accuracy:** stuck at 35-40%

Diagnosis: Domain Mismatch & Overfitting

- The model was memorizing the training set.
- The pre-trained ImageNet features (for cars, dogs, etc.) were not translating well to the new domain of faint, grayscale faces.

The Debugging Journey: A Lesson in Persistence

- **Hypothesis 1: Training Strategy is Wrong.**
  - **Attempted:** Complex warm-up stages, freezing/unfreezing layers, multiple schedulers (`ReduceLROnPlateau`, `CosineAnnealingLR`).
  - **Result:** No significant improvement. The model was fundamentally stuck.

The Debugging Journey: A Lesson in Persistence

- **Hypothesis 1: Training Strategy is Wrong.**
  - **Attempted:** Complex warm-up stages, freezing/unfreezing layers, multiple schedulers (`ReduceLROnPlateau`, `CosineAnnealingLR`).
  - **Result:** No significant improvement. The model was fundamentally stuck.
- **Hypothesis 2: The Data Pipeline is Corrupted.**
  - **Attempted:** A sanity-check script to visualize the exact images being fed to the model.
  - **Result:** The images were visually correct. This ruled out data corruption.

The Breakthrough?

The Key Observation:

- The input images from FER-2013 are very faint and have low contrast.

The New Hypothesis:

- The model isn't failing because it's too complex; it's failing because the **input signal is too weak**.

The Solution: Aggressive Contrast Enhancement

- Added two powerful transforms to the data augmentation pipeline: `RandomEqualize` and strong `ColorJitter`.

The Final Fine-Tuning Strategy

- **Model:** Refactored `ResNet-18` to cleanly separate the feature extractor and classifier.
- **Training:** A simple, end-to-end strategy from epoch 1.
  - No warm-up stage.
  - Single `Adam` optimizer with differential learning rates.
- **Regularization:** Strong `weight_decay`, `dropout`, and `label_smoothing`.
- **Data:** The new pipeline with aggressive contrast enhancement.

Final Results

Model	Peak Validation Accuracy
Custom CNN (Baseline)	61%
Fine-Tuned ResNet-18 (Final)	**41%**

Conclusion: Unfortunately, we were not able to successfully leverage the pre-trained ResNet-18 model to outperform our custom CNN.

Key Learnings

- **Data Quality and Model Complexity:** Improving the input signal with contrast enhancement was more impactful than any complex training strategy.
- **Transfer Learning is Not Automatic:** Pre-trained models can suffer from "domain mismatch" and require careful, methodical fine-tuning.
- **Debugging is Iterative:** When stuck, form a hypothesis, design a simple experiment to test it, and use the result to inform your next step.

New Tools and Skills

This project involved a number of engineering practices **Project Structuring:**

- **Modular Code:** Separating logic into distinct components (‘datasets‘, ‘models‘, ‘trainers‘) made the project manageable and easy to debug.
- **Configuration Files:** Using config files (e.g., `config.yaml`) to manage all hyperparameters, separating configuration from code.
- **Reproducibility:** A clear project structure ensures that any experiment can be reliably reproduced.

Robust Logging:

- Implemented a `BaseTrainer` to handle systematic logging to both files and the console, creating a permanent record of every run.

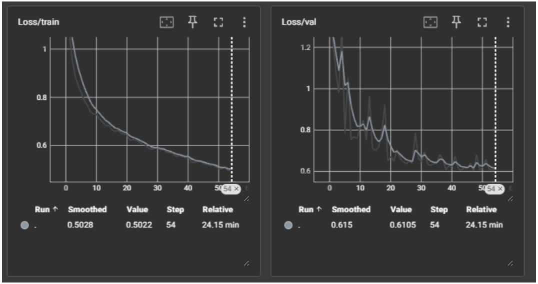
New Tools and Skills

Deep Learning Debugging:

- **Sanity Checks:** Developed scripts to visualize data transforms and verify the data quality being fed to the model.
- **Methodical Tuning:** Learned to isolate variables and test hypotheses systematically (e.g., proving the issue was data quality, not the scheduler).

Experiment Visualization:

- **TensorBoard:** Used extensively to monitor training in real-time.
  - Plotted loss and accuracy curves to diagnose overfitting vs. underfitting.
  - Compared different runs to see the impact of hyperparameter changes.



Thank You

Questions?