Facial Expression Recognition on FER-2013 Using CNN in PyTorch

1 Introduction

Facial expression recognition (FER) plays a vital role in enhancing human-computer interaction, security systems, and emotion-based content analysis. This project aims to build a deep learning model using convolutional neural networks (CNNs) to automatically recognize seven facial expressions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.

We use the FER-2013 dataset, a benchmark dataset consisting of grayscale images of faces labeled with one of the seven emotions. Our approach includes preprocessing, model design, training with data augmentation, evaluation on test data, and result visualization using accuracy/loss curves and a confusion matrix.

2 Dataset Overview

2.1 Dataset Source

The FER-2013 dataset is available on Kaggle. It contains facial images in CSV format, where each image is a 48×48 grayscale array stored as a space-separated string. The dataset is split into three subsets using the `Usage` column:

- **Training** (28,709 samples)
- **PublicTest** (Validation, 3,589 samples)
- **PrivateTest** (Test, 3,589 samples)

2.2 Emotion Classes

Label	Emotion
0	Angry
1	Disgust
2	Fear
3	Happy
4	Sad
5	Surprise
6	Neutral

Table 1: Emotion Classes in FER-2013 Dataset

3 Data Preprocessing and Augmentation

3.1 Data Loader Implementation

We created a custom `FerDataset` class to parse the CSV and return image-label pairs. Images are reshaped into 48×48 grayscale arrays and converted to PIL images for transformation.

```python
class FerDataset(torch.utils.data.Dataset):
    def __init__(self, ds_path, split='Training', transform=None):
        self.ds = pd.read_csv(ds_path)
        self.ds = self.ds[self.ds['Usage'] == split]
        self.transform = transform

    def __len__(self):
        return len(self.ds)

    def __getitem__(self, index):
        label = int(self.ds.iloc[index]['emotion'])
        pixels = np.array(self.ds.iloc[index]['pixels'].split(), dtype='uint8')
        image = pixels.reshape(48, 48)
        image = Image.fromarray(image)

        if self.transform:
            image = self.transform(image)

        return image, label
```

Python

3.2 Transformations

- **Training set**: Random horizontal flips and rotation (±10°), normalization to range [-1, 1]
- **Validation/Test set**: Normalization to range [-1, 1] only

```python
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

val_test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])
```

Python

3.3 DataLoaders

Used a batch size of 64. Data loaders were wrapped with a `DeviceDataLoader` class to enable GPU compatibility.

```python
csv_path = r"C:\Users\sahil\OneDrive\Documents\Facial\fer2013.csv"

train_loader = DataLoader(FerDataset(csv_path, split='Training', transform=train_transform), batch_size=64, shuffle=True)
val_loader = DataLoader(FerDataset(csv_path, split='PublicTest', transform=val_test_transform), batch_size=64)
test_loader = DataLoader(FerDataset(csv_path, split='PrivateTest', transform=val_test_transform), batch_size=64)
```

Python

4 Model Architecture

4.1 CNN Summary

- 4 convolutional layers with batch normalization, ReLU, and max pooling

- Final FC layers: 256 neurons → 7-class output
- Dropout: 0.3 after the first FC layer
- Total parameters ≈ 1.7M

Layer	Output Shape	Details
Input	[1, 48, 48]	Grayscale image
Conv2D + BN + ReLU	[32, 48, 48]	Kernel: 3x3, padding=1
MaxPool2D	[32, 24, 24]	Pool: 2x2
Conv2D + BN + ReLU	[64, 24, 24]	
MaxPool2D	[64, 12, 12]	
Conv2D + BN + ReLU	[128, 12, 12]	
MaxPool2D	[128, 6, 6]	
Conv2D + BN + ReLU	[256, 6, 6]	
MaxPool2D	[256, 3, 3]	
Flatten	2304	
FC1 + ReLU + Dropout	256	Dropout: 0.3
FC2	7	Final logits

Table 2: Model Architecture

```python
# Model Creation
class EmotionAnalyser(nn.Module):
    def __init__(self, num_classes=7):
        super().__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1) # ==> Output Size [32, 48, 48]
        self.bn1 = nn.BatchNorm2d(32)

        self.pool = nn.MaxPool2d(2,2) # Output Size ==> [32, 24, 24]

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1) # Output Size [64, 24, 24] --> [64, 12, 12]
        self.bn2 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1) # Output Size [128, 12, 12] --> [128, 6, 6]
        self.bn3 = nn.BatchNorm2d(128)

        self.fc1 = nn.Linear(128*6*6, 256)
        self.fc2 = nn.Linear(256, num_classes)

        self.relu = nn.ReLU()
        self.droupout = nn.Dropout(0.5)

    def forward(self, xb):
        out = self.poo (module) nn elf.bn1(self.conv1(xb))))
        out = self.pool(self.relu(self.bn2(self.conv2(out))))
        out = self.pool(self.relu(self.bn3(self.conv3(out))))
        out = out.view(-1, 128 * 6 * 6)
        out = self.relu(self.fc1(out))
        out = self.droupout(out)
        out = self.fc2(out)

        return out
```

Python

5 Training Setup

- **Loss Function**: CrossEntropyLoss (multi-class classification)
- **Optimizer**: Adam (`lr=0.001`)

- **Scheduler**: ReduceLROnPlateau (mode='min', factor=0.5, patience=2)
- **Epochs**: 50
- **Model Checkpoint**: Saved when validation accuracy improved

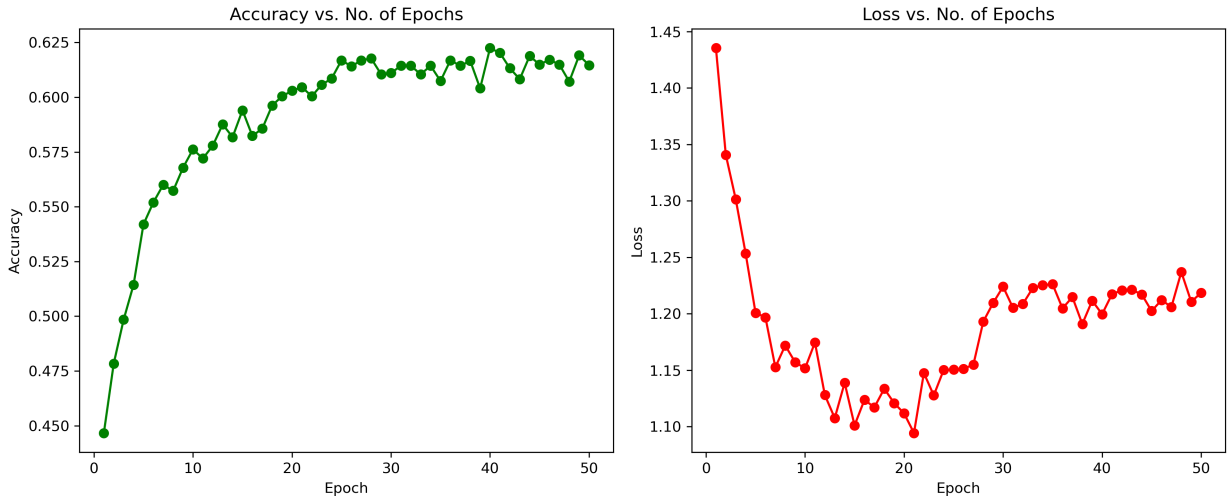
```python
learning_rate = 1e-3
epochs = 50
model = EmotionAnalyser()
to_device(model, device)


optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=2)
```

Python

6 Results and Evaluation

6.1 Accuracy and Loss Curves

Accuracy and loss curves were generated across 50 epochs and saved as:

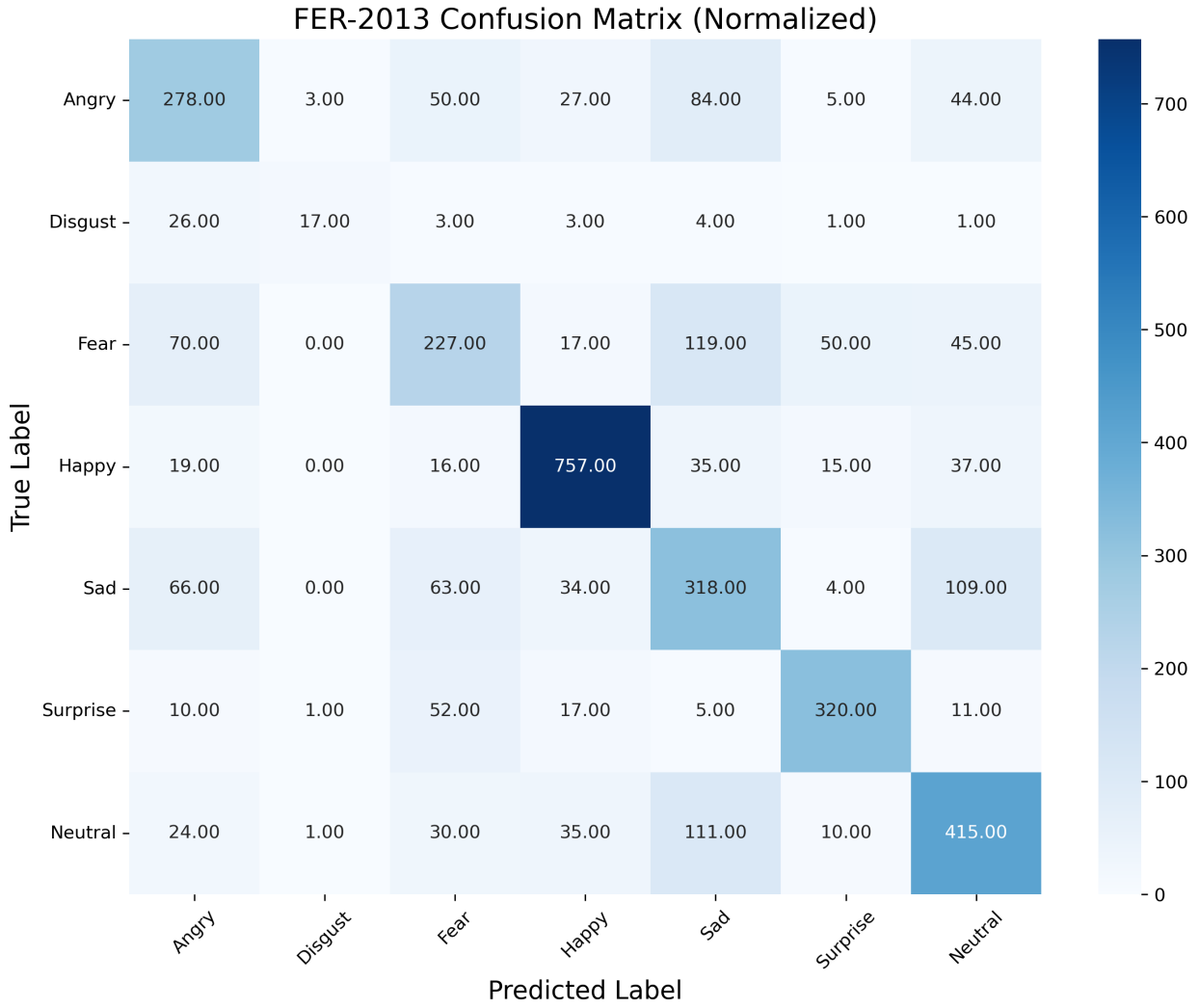


6.2 Test Set Performance

- **Test Accuracy**: 63.33%
- **Test Loss**: 1.1434

6.3 Confusion Matrix

A confusion matrix was plotted and saved to:



Key observations:

- **Happy**, **Neutral**, and **Sad** had the highest classification accuracy.
- **Disgust** was often confused with Angry or Fear due to low sample count.

7 Conclusion

This project demonstrates a CNN-based approach to facial expression recognition using the FER-2013 dataset. With proper preprocessing, data augmentation, and model tuning, the system achieved competitive accuracy. Future work will focus on increasing accuracy further through advanced architectures and training techniques.