Custom CNN Model Training on FER-2013 Dataset And Deployment in Streamlit

1. Introduction

This project implements a Convolutional Neural Network (CNN) to classify facial emotions using the FER-2013 dataset. The model is trained on the dataset and then deployed using Streamlit to predict emotions from user-provided images. The purpose of the project is to develop an end-to-end solution for facial emotion recognition.

---

2. Dependencies

The following libraries are required for the project:

pip install torch torchvision scikit-learn

---

3. Dataset

- **FER-2013**: The dataset contains 7 categories of emotions: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise.
- The dataset is organized into train and test folders, with subfolders for each emotion class.

---

4. Data Augmentation and Preprocessing

To improve model generalization, the training dataset is augmented with transformations, including horizontal flips, random rotations, random crops, and brightness/contrast adjustments. These transformations help the model learn invariant features across various image distortions.

- **Transformations for Training Data:**

python

Copy code

```
transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((48, 48)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomCrop(48, padding=4),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

- **Transformations for Test Data:**

```
python
Copy code
transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((48, 48)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

---

5. Model Architecture

The model is a custom CNN consisting of 3 convolutional layers, each followed by batch normalization, ReLU activation, and max pooling. The network also includes fully connected layers with dropout to reduce overfitting.

- **Convolutional Layers**: Extract features from input images.
- **Batch Normalization**: Normalizes inputs to improve convergence.
- **ReLU Activation**: Introduces non-linearity to the model.
- **Max Pooling**: Reduces spatial dimensions while retaining essential features.
- **Fully Connected Layers**: Perform classification based on extracted features.
- **Dropout**: Regularizes the model by randomly dropping neurons during training.

```
python
Copy code
class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
```

```
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(256 * 6 * 6, 512)
        self.fc2 = nn.Linear(512, 7)
        self.dropout = nn.Dropout(p=0.3)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.bn1(x)
        x = self.pool(x)

        x = self.conv2(x)
        x = self.relu(x)
        x = self.bn2(x)
        x = self.pool(x)

        x = self.conv3(x)
        x = self.relu(x)
        x = self.bn3(x)
        x = self.pool(x)

        x = x.view(-1, 256 * 6 * 6)
        x = self.fc1(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

---

6. Training Configuration

- **Loss Function**: Cross-Entropy Loss with Label Smoothing (0.1), which helps prevent overconfidence in the predictions.
- **Optimizer**: Adam optimizer with a learning rate of 0.0005 and weight decay of 1e-4.
- **Learning Rate Scheduler**: StepLR, which reduces the learning rate by half every 5 epochs.
- **Early Stopping**: The model monitors validation accuracy, and training stops early if validation accuracy does not improve for 5 consecutive epochs.
- **Device**: The model runs on GPU if available; otherwise, it defaults to CPU.

```
# Loss Function
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)


# Optimizer
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-4)


# Learning Rate Scheduler
scheduler = StepLR(optimizer, step_size=5, gamma=0.5)
```

---

7. Accuracy Calculation

Accuracy is calculated by comparing the predicted labels with the true labels using the accuracy_score function from scikit-learn.

---

8. Early Stopping

The early_stopping function monitors validation accuracy for a certain patience period. If no improvement in validation accuracy is observed during this period, the training process halts.

---

9. Training and Evaluation

The model is trained for a maximum of 100 epochs, with evaluation occurring at the end of each epoch. Early stopping is applied if there is no improvement in validation accuracy.

---

10. Evaluation

The evaluate_model function evaluates the model on the test dataset by calculating the accuracy of the predictions.

11. Results

Training and test accuracies are displayed at the end of each epoch. Example output for a few epochs:

Epoch [1/100], Loss: 2.2981, Train Accuracy: 29.27%, Test Accuracy: 41.84%

Epoch [2/100], Loss: 1.5487, Train Accuracy: 44.44%, Test Accuracy: 50.20%

Epoch [3/100], Loss: 1.4683, Train Accuracy: 49.05%, Test Accuracy: 53.46%

Epoch [20/100], Loss: 1.1897, Train Accuracy: 64.55%, Test Accuracy: 63.15%

Epoch [50/100], Loss: 1.1414, Train Accuracy: 67.10%, Test Accuracy: 64.41%

---

12. Streamlit Model Deployment

The trained CNN model was deployed using Streamlit, providing an interactive user interface where users can upload an image and receive predictions on the facial emotion detected in the image.

13. How Streamlit Application Works

- The user uploads an image of a face in .jpg format.
- The uploaded image is preprocessed: resized to 48x48 pixels, converted to grayscale, and normalized.
- The processed image is passed to the CNN model, which predicts the emotion from one of the seven classes.
- The predicted emotion is displayed on the Streamlit interface.

14. Streamlit Application Interface

The interface allows users to:

- Upload an image
- View the uploaded image
- Receive the predicted emotion displayed alongside the image**.**

15. Conclusion

In this project, successfully built and deployed a facial emotion recognition system using a custom Convolutional Neural Network (CNN) trained on the FER-2013 dataset. The CNN model was able to achieve reasonable performance, with a test accuracy of 64%, by learning features from grayscale facial images and classifying them into seven distinct emotions: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise.

The deployment of this model through Streamlit allows for real-time user interaction, where users can upload images to receive predictions about the facial emotion detected. This integration

demonstrates the practical applicability of deep learning models in user-facing applications and highlights how machine learning models can be made accessible through modern web frameworks.