Report: Gesture Recognition System Experiment

Introduction

This report outlines the procedure followed to design and evaluate a gesture recognition system using a Convolutional Neural Network (CNN). The system was tasked with recognizing three hand gestures: open palm, fist, and thumbs up. The dataset was split into training and testing sets, and the model was evaluated using k-fold cross-validation. Performance metrics, including accuracy and F1 score, were used to assess the model's performance.

Dataset Preparation

1. **Data Collection:**

   1. Video Recording: Three videos were recorded, each representing one of the gestures: **open palm, fist, and thumbs up**.
   2. These videos were recorded from various angles and orientations to ensure diversity in the dataset.
   3. Frame Extraction: From each video, 500 frames were extracted, resulting in 500 images per gesture.

2. **Data Split:**

   The 500 images in each subfolder were randomly divided into training and test sets:

- **Training Set: 300** images per gesture
- **Test Set: 200** images per gesture

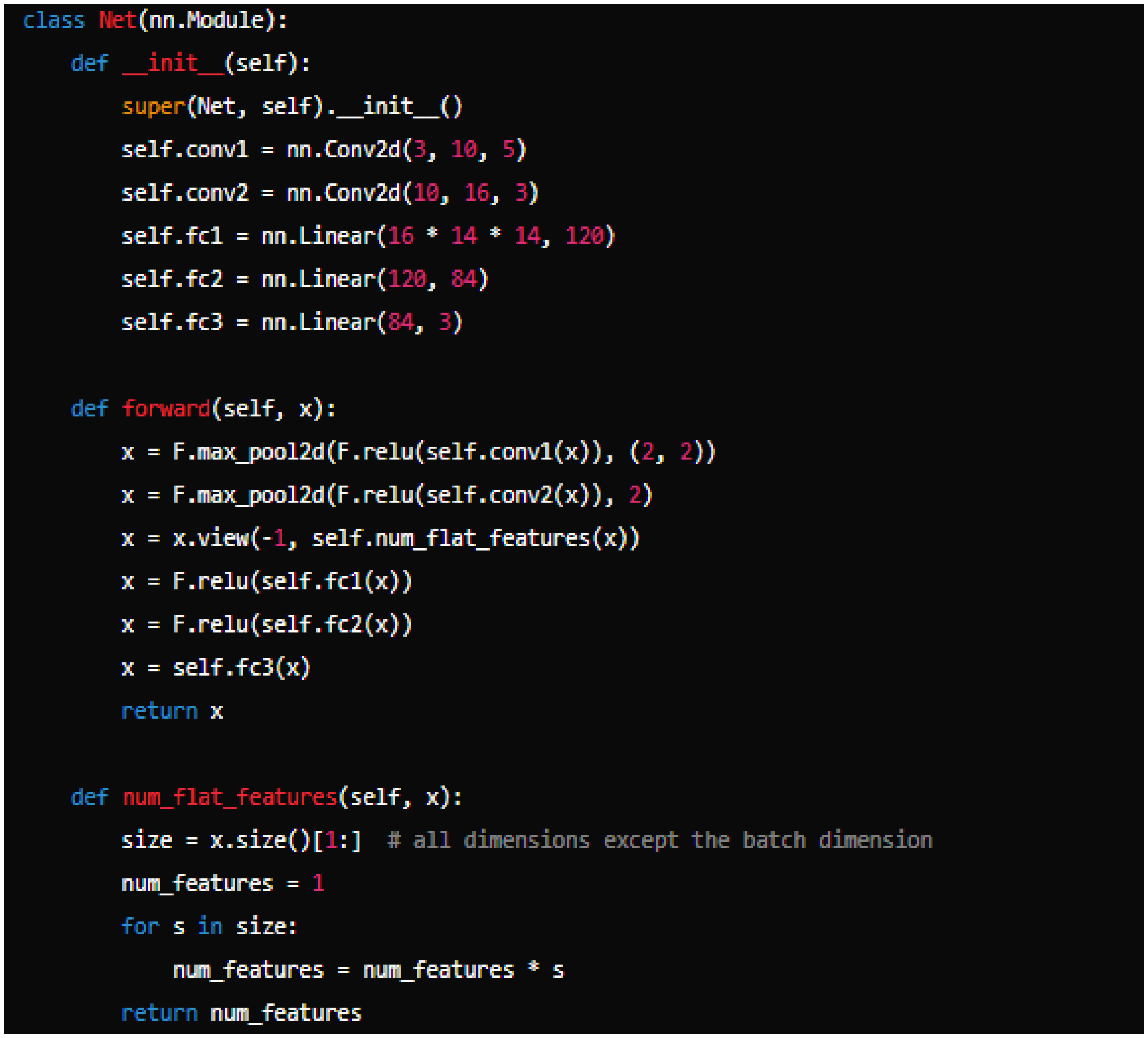
The dataset was organized into the following structure:

```
  datasets/
train/
  open_palm/
  fist/
  thumbs_up/
test/
  open_palm/
  fist/
  thumbs_up/
```

Model Architecture

The CNN model used in this experiment consists of the following layers:

- Two convolutional layers with **ReLU** activation and **max-pooling**.
- Three fully connected layers with **ReLU** activation.
- An output layer with 3 nodes, corresponding to the three gestures.



```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 10, 5)
        self.conv2 = nn.Conv2d(10, 16, 3)
        self.fc1 = nn.Linear(16 * 14 * 14, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 3)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features = num_features * s
        return num_features
```

Training Procedure

1. **Transformations**:

    - Images were resized to **64x64 pixels and normalized**.
    - Data augmentation techniques, such as random horizontal flips and rotations, were applied during training to improve generalization.

2. **Training**:

    - The model was trained using **stochastic gradient descent (SGD)** with a **learning rate of 0.001** and **momentum of 0.9**.
    - Training was performed for **20 epochs**.
    - Loss values were plotted to monitor the training process.
    - The Total time taken for training was **277.2662773132324 sec**

Initial Testing and Overfitting Check

Initially, the model was tested on the entire dataset for **20 epochs**.

The results were:

- **Accuracy**: 99.67%
- **F1 Score**: 1

The extremely high accuracy and F1 score indicated the possibility of overfitting, prompting the need for a more rigorous evaluation method.

Below is the loss vs iteration curve when we test the model on entire dataset
The output of this was stored in the **output.xlsx** file.

Contents of `output.xlsx`

The Excel file `output.xlsx` contains two columns:

- **Ground truth**: The actual labels of the test images. **(0(fists), 1(Palm), 2(Thumbs up) in order)**
- **Predicted**: The labels predicted by the CNN model for the test images.(Predicted by model)

Calculation of Accuracy

Accuracy is the ratio of the number of correct predictions to the total number of predictions. Mathematically, it is given by:

$$ACCURACY = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \times 100\%$$

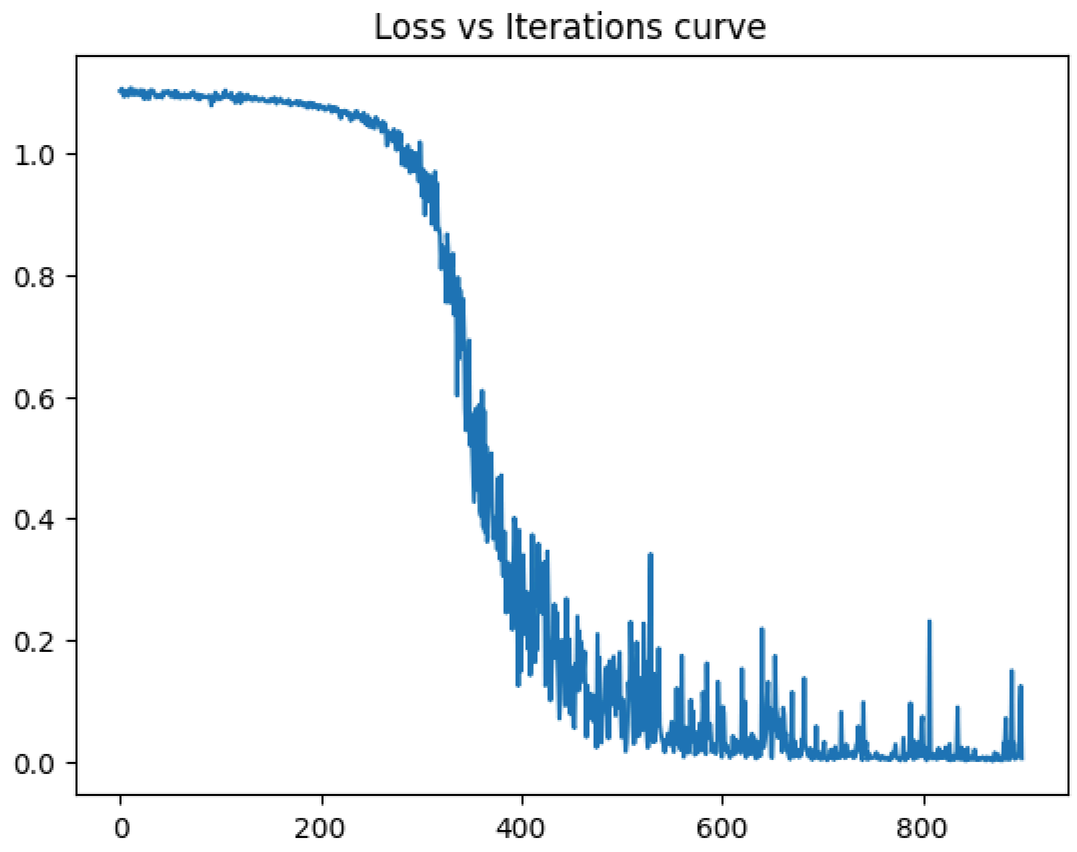
Calculation of F1 Score

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances both concerns. It is particularly useful when the dataset is imbalanced. Mathematically, it is given by:

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Where:

- **Precision** is the ratio of true positive predictions to the total predicted positives.
- **Recall** is the ratio of true positive predictions to the total actual positives.



Cross-Validation

To ensure reliable performance estimates, **k-fold cross-validation** was used:

- **K-Fold Cross-Validation**:
  - The dataset was split into **5 folds**.
  - For each fold, the model was trained on 4 folds and tested on the remaining fold.
  - This process was repeated 5 times, with each fold serving as the test set once.
  - Predictions for each fold were saved in separate Excel files.
  - In this case the time taken was **495.7946033477783 sec**

Evaluation Metrics

- Accuracy and F1 Score were calculated for each fold using the ground truth and predicted labels.
- Mean and standard deviation of these metrics were computed to provide an overall performance estimate.

Results

1. **Per-Fold Metrics**:
   - Fold 0: **Accuracy** = 97.22%, **F1 Score** = 0.97
   - Fold 1: **Accuracy** = 96.11%, **F1 Score** = 0.96
   - Fold 2: **Accuracy** = 98.33%, **F1 Score** = 0.98
   - Fold 3: **Accuracy** = 100.00%, **F1 Score** = 1.00
   - Fold 4: **Accuracy** = 100.00%, **F1 Score** = 1.00

Overall Metrics:

- Mean Accuracy: **98.33%**
- Mean F1 Score: **0.98**

Code Modifications and Justifications

The provided code was modified to include k-fold cross-validation and the saving of predictions for each fold. These changes are essential for ensuring the model's robustness and reliability. Here are the specific changes made:

1. **Added K-Fold Cross-Validation**:

   - Implemented k-fold cross-validation to split the dataset into 5 folds. This helps in evaluating the model's performance on different splits of the data, providing a more reliable performance estimate.

2. **Saving Predictions for Each Fold**:

   - Modified the code to save predictions for each fold in separate Excel files (`output_fold_0.xlsx`, `output_fold_1.xlsx`, etc.). This allows for a detailed analysis of the model's performance on each fold.

3. **Calculation of Overall Metrics**:

   - Added code to calculate and print the mean accuracy and F1 score across all folds. This provides a summary of the model's performance.

4. **Transformation Adjustments**:
    - Ensured that image transformations (resizing, normalization, augmentation) were correctly applied to maintain the expected image size and improve generalization.

Conclusion

The gesture recognition system achieved high accuracy and F1 scores across all folds, indicating excellent performance. The low variability between folds suggests that the model's performance is consistent and not overfitting. The use of k-fold cross-validation helped in obtaining reliable performance estimates by ensuring that the model was evaluated on multiple splits of the data.

Key Insights

- **High Performance**: The model consistently achieved high accuracy and F1 scores across all folds.
- **Model Robustness**: The low standard deviation in performance metrics indicates robustness and consistency.
- **No Overfitting**: The performance on the test sets was comparable to the training sets, suggesting the model generalizes well.