REPORT SUBMISSION

NAME:RIYA JAIN

PROJECT CATEGORY:VISION

PROJECT TOPIC: Gesture Recognition System

UG COURSE:B.Tech in Computer Science and Engineering

YEAR OF COURSE:3rd Year

COLLEGE:JSS Academy of Technical Education,Noida

GESTURE RECOGNITION SYSTEM REPORT

Project Overview

The goal of this project is to build a gesture recognition system using a Convolutional Neural Network (CNN). The system classifies images into three categories (open palm, fist or thumbs-up) based on the gestures they depict. The project involves training the CNN on a dataset of gesture images and evaluating its performance on a test dataset.

Given

We are given a code file named cnn.py which contains the basic python code for building the model. We are supposed to add the path of the datasets folder in the code for training of data and run it while removing all the errors.

Dataset

We captured 500 images of each gesture (open palm, fist or thumbs-up) in different angles and orientations and put it in folder named datasets. The dataset consists of gesture images divided into two folders: train and test. Each folder contains subfolders for each class of gestures (open palm, fist or thumbs-up). Each of these 3 sub-folders contain 300 images of the corresponding gesture for the "train" sub-folder and 200 images of the corresponding gesture for the "test" sub-folder. All the images are of size 64x64.
By analyzing the given code, we understood that the images are preprocessed using the following transformations:

- Conversion to tensor
- Normalization with mean 0.5 and standard deviation 0.5

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

Platform Used

We used Jupyter Notebook to build our model because it is more efficient for working in Machine Learning that any IDLEs or Code Editor.
It helped to detect the error parts in the provided code files which enabled us to solve them more efficiently.

Libraries Used

- Pandas
- Torch
- Torchvision
- Openpyxl
- Matplotlib.pyplot
- Time
- os

Model Architecture

The CNN model used in this project as per analysis of given code is defined as follows:

1. Convolutional Layer 1: 3 input channels, 10 output channels, kernel size of 5
2. Max Pooling Layer 1: Kernel size of 2x2
3. Convolutional Layer 2: 10 input channels, 16 output channels, kernel size of 3
4. Max Pooling Layer 2: Kernel size of 2x2
5. Fully Connected Layer 1: Input size 16*14*14, output size 120
6. Fully Connected Layer 2: Input size 120, output size 84
7. Output Layer: Input size 84, output size 3

Training

The training process involves the following:

- Criterion: Cross Entropy Loss
- Optimizer: Stochastic Gradient Descent (SGD) with learning rate 0.001 and momentum 0.9
- Batch Size: 20
- Number of Epochs: 20

Training Results

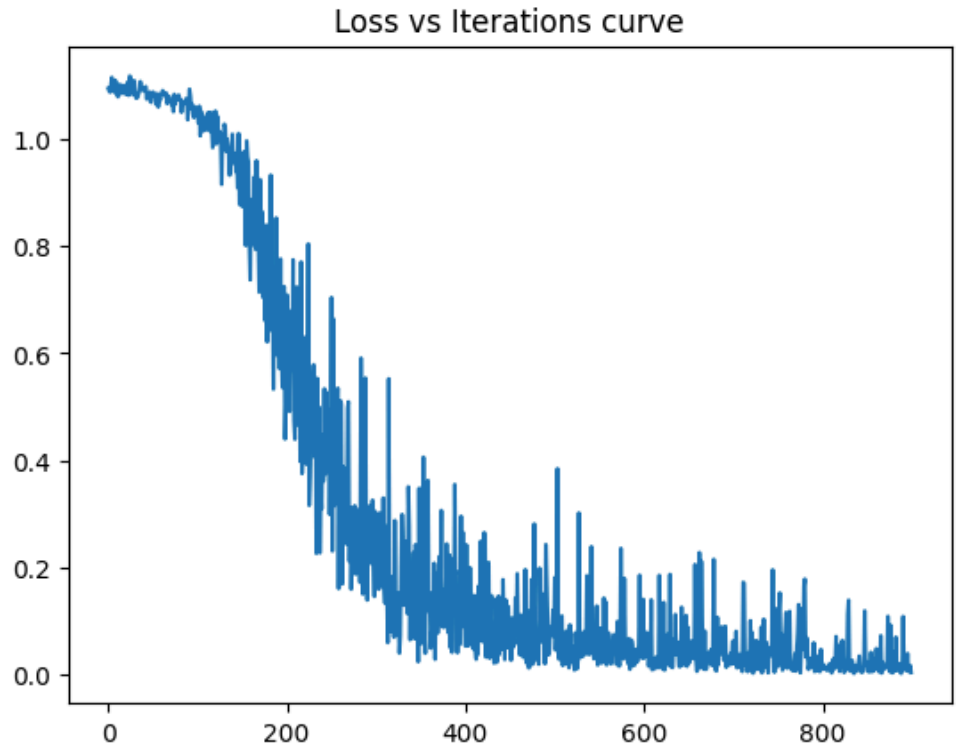
The code below is training a neural network by iterating over multiple epochs and batches of data, computing the loss, and updating the model's parameters using backpropagation. It also prints the loss for each batch and stores the loss values in a list.

```
[11]: for epoch in range(num_epochs):
          for i, data in enumerate(folderL, 0):
              inputs, labels = data
              optimizer.zero_grad()
              outputs = net(inputs)
              loss = criterion(outputs, labels)
              loss.backward()
              optimizer.step()
              print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, loss.item()))
              loss_values.append(loss.item())
```

```
[1,     1] loss: 1.094
[1,     2] loss: 1.108
[1,     3] loss: 1.078
[1,     4] loss: 1.079
[1,     5] loss: 1.103
[1,     6] loss: 1.095
[1,     7] loss: 1.082
[1,     8] loss: 1.107
[1,     9] loss: 1.104
[1,    10] loss: 1.088
[1,    11] loss: 1.078
[1,    12] loss: 1.080
[1,    13] loss: 1.111
[1,    14] loss: 1.095
[1,    15] loss: 1.103
[1,    16] loss: 1.075
[1,    17] loss: 1.109
[1,    18] loss: 1.098
```

The loss during the training process is plotted in the graph below. The loss values are recorded at each iteration for visualization. We used the matplotlib library to do so.

```
: plt.plot(loss_values)
  plt.title("Loss vs Iterations curve")
  plt.show()
```



Evaluation

Training Accuracy

After training, the model's accuracy on the training dataset is evaluated. We used sklearn.metrics library to get the accuracy_score. The accuracy achieved on the training dataset is:

Accuracy of the network on the train images: 99.78 %

```
training_accuracy = accuracy_score(all_labels, all_predictions, normalize=True, sample_weight=None)
training_f1_score = f1_score(all_labels, all_predictions, average='weighted')
print('Accuracy of the network on the train images: %.2f %%' % (training_accuracy * 100))
print('F1 Score of the network on the train images: %.2f' % training_f1_score)
```

```
Accuracy of the network on the train images: 99.78 %
F1 Score of the network on the train images: 1.00
```

Test Accuracy

The model's performance is also evaluated on the test dataset. We used sklearn.metrics library to get the accuracy_score. The accuracy achieved on the test dataset is:

Accuracy of the network on the test images: 96.17 %

```
test_accuracy = accuracy_score(gt_arr, pred_arr, normalize=True, sample_weight=None)
test_f1_score = f1_score(gt_arr, pred_arr, average='weighted')
print(f'Accuracy of the network on the test images: %.2f %%' % (test_accuracy * 100))
print(f'F1 Score of the network on the test images: %.2f' % test_f1_score)
```

```
Accuracy of the network on the test images: 96.17 %
F1 Score of the network on the test images: 0.96
```

F1-Score

The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. Precision measures the proportion of true positive predictions out of all positive predictions, while recall measures the proportion of true positive predictions out of all actual positives. The relative contribution of precision and recall to the F1 score are equal.F1 score is also known as balanced F-score or F-measure.

F1 Score of the network on the train images: 1.00

F1 Score of the network on the test images: 0.96

The above F1 score shows the overall high level of performance in terms of both precision and recall for the classifier.

But to ensure that each class has high performance for classifier, we find F1 score for each class:

```
# Calculate F1 score for each class
f1_scores = f1_score(gt_arr, pred_arr, average=None)
num_classes = len(f1_scores)

# Print F1 score for each class
for i in range(num_classes):
    print(f"F1 score for class {i}: {f1_scores[i]}")
```

```
F1 score for class 0: 0.9523809523809523
F1 score for class 1: 0.95
F1 score for class 2: 0.9825436408977556
```

Above score shows that each class has high performance.

Accuracy Score

The accuracy score is a metric commonly used to evaluate the overall performance of a classifier in a multi-class classification problem. It measures the proportion of correctly classified samples out of the total number of samples in the dataset.
The accuracy score of our model is:

```
# Calculate accuracy
accuracy = accuracy_score(gt_arr, pred_arr)

# Print accuracy
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9616666666666667
```

Predictions

The predictions made by the model on the test dataset are saved in an Excel file named output.xlsx. The file contains the ground truth labels (GT) and the predicted labels (Pred) for each test image. We added the openpyxl module to create the excel file by python.

Report Generation

The entire process, including model training, evaluation, and prediction recording, was completed in:

```
print(f"Total time taken = {time() - time_start} sec")
```

```
Total time taken = 78.68026614189148 sec
```

Excel File Verification

The predictions were successfully saved in the Excel file, and the file was verified to exist at the end of the process.

```
# Check if the file was actually created
if os.path.exists(excel_filename):
    print(f"Excel file '{excel_filename}' was created successfully.")
else:
    print(f"Failed to create Excel file '{excel_filename}'.")
```

```
Excel file 'output.xlsx' was created successfully.
```

This concludes that we successfully created the gesture recognition system project.

Error Handling

All errors in the cnn.py file were detected by adding multiple print statements in the code that helped to understand which process is having issues with code. Addition of multiple try-except statements were done to handle the errors in a much better way.

Conclusion

The CNN model successfully classified gesture images with high accuracy on both training and test datasets. The results indicate that the model is effective in recognizing gestures. Future improvements could include hyperparameter tuning, data augmentation, and experimenting with deeper networks to further enhance performance.