ML LAB 14
CNN

NAME: DELISHA RIYONA DSOUZA
SRN: PES2UG23CS166
DATE: 18 NOV

INTRODUCTION
The objective of this lab was to design, implement, train, and evaluate a Convolutional Neural Network (CNN) using PyTorch to classify hand gesture images into three categories: rock, paper, and scissors. The dataset contains 2,188 images organized into class folders and was loaded using torchvision's ImageFolder utility. The goal was to build a model that generalizes well to unseen images and to report the architecture, training hyperparameters, and final performance.

Model Architecture

The CNN model used in this lab consists of three convolutional layers followed by a fully connected classifier. The convolutional block begins with a Conv2d layer that takes 3 input channels and produces 16 feature maps using a 3×3 kernel with padding 1, followed by a ReLU activation and a 2×2 MaxPooling layer. This is followed by a second Conv2d layer with 32 output channels (3×3 kernel, padding 1), again followed by ReLU and MaxPooling. The third convolutional layer increases the depth to 64 channels, using the same 3×3 kernel and padding strategy, after which another MaxPooling operation reduces the spatial dimensions. Starting from a 128×128 input image, these three MaxPooling layers progressively downsample the feature map to 16×16, resulting in a flattened size of 64 × 16 × 16 = 16,384 features. The fully connected classifier then processes this flattened vector through a Linear layer with 256 hidden units, applies ReLU activation and dropout (p=0.3) for regularization, and finally outputs predictions through a final Linear layer with 3 units corresponding to the classes rock, paper, and scissors.

Mention key parameters like kernel size, number of channels, and the use of Max Pooling.

The CNN uses three convolutional layers, each with a 3×3 kernel size and padding = 1. The number of channels increases progressively across the layers: the first convolution maps the input from 3 to 16 channels, the second from 16 to 32 channels, and the third from 32 to 64 channels. After each convolution and ReLU activation, the model applies a 2×2 MaxPooling layer, which reduces the spatial resolution by half and helps extract the most important features while reducing computation.

Describe the fully-connected classifier.

The fully-connected classifier begins by flattening the output of the final convolutional layer into a single feature vector of size 16,384. This is passed into a Linear layer with 256 neurons, followed by a ReLU activation to introduce non-linearity and a Dropout layer (p = 0.3) to reduce overfitting. Finally, the classifier ends with a Linear layer with 3 output units, corresponding to the three classes: rock, paper, and scissors.

Training and Performance

State the key hyperparameters used for training: optimizer, loss function, learning rate, and number of epochs.

The model was trained using the Adam optimizer, with a learning rate of 0.001. The CrossEntropyLoss function was used as the loss criterion, and the network was trained for a total of 10 epochs

Report the final Test Accuracy your model achieved.

The final test accuracy achieved by the model was 98.63%.

Conclusion and Analysis:

Model Performance:
The model performed very well, achieving a high test accuracy of **98.63%**, indicating that the CNN was able to learn and generalize the visual patterns in rock, paper, and scissors images effectively.

Challenges Faced:

- The dataset is relatively small, so there was a risk of overfitting during training.
- Ensuring correct preprocessing and maintaining consistent transforms for both training and testing were important to avoid mismatch errors.

Possible Improvements:

- **Add Data Augmentation:** Incorporating random flips, rotations, and color jitter could help the model become more robust and prevent overfitting.
- **Use Transfer Learning:** Using a pretrained model like ResNet18 or MobileNet and fine-tuning it on this dataset may further improve accuracy, especially for small datasets.

```
Epoch 1/10, Loss = 0.6753
Epoch 2/10, Loss = 0.1605
Epoch 3/10, Loss = 0.0760
Epoch 4/10, Loss = 0.0674
Epoch 5/10, Loss = 0.0311
Epoch 6/10, Loss = 0.0269
Epoch 7/10, Loss = 0.0136
Epoch 8/10, Loss = 0.0081
Epoch 9/10, Loss = 0.0173
Epoch 10/10, Loss = 0.0030
Training complete!
```

```python
model.eval() # Set the model to evaluation mode
correct = 0
total = 0

# TODO: Use torch.no_grad()
# We don't need to calculate gradients during evaluation
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # TODO: Get model predictions
        # 1. Get the raw model outputs (logits)
        outputs = model(images)

        # 2. Get the predicted class (the one with the highest score
        #    Hint: use torch.max(outputs, 1)
        _, predicted = torch.max(outputs, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

```
Test Accuracy: 98.63%
```