ML LAB – WEEK 14

NAME: NINAD CHAVAN
SRN: PES2UG23CS392

I. INTRODUCTION

The objective of this laboratory experiment was to design, implement, and evaluate a Convolutional Neural Network (CNN) using the PyTorch framework to classify images into three distinct categories: rock, paper, and scissors. Starting with a boilerplate notebook, the task involved constructing a complete deep learning pipeline. This included preprocessing the dataset (resizing and normalization), defining a custom CNN architecture with multiple convolutional and pooling layers, and implementing the training and evaluation loops. The ultimate goal was to observe how the CNN extracts spatial features from raw image data to achieve high classification accuracy on unseen test data.

II. MODEL ARCHITECTURE

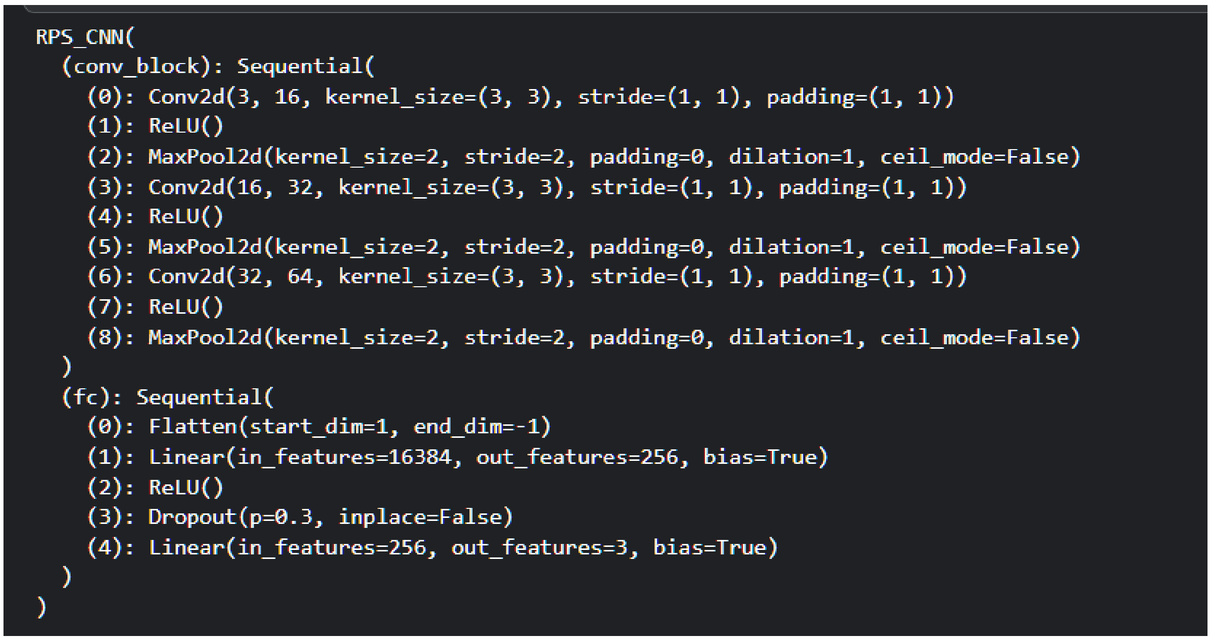
The architecture implemented is a customized Convolutional Neural Network (RPS_CNN) designed to process 128×128 RGB images. The network is divided into two main components: the Feature Extractor (Convolutional Blocks) and the Classifier (Fully Connected Layers).

1. Convolutional Blocks

The feature extractor consists of three sequential blocks. Each block applies a convolution operation to extract features, followed by a ReLU activation for non-linearity, and a MaxPool layer to reduce spatial dimensions.

- Block 1: Conv2d(3 → 16, kernel=3, padding=1) → ReLU → MaxPool2d(2)
- Block 2: Conv2d(16 → 32, kernel=3, padding=1) → ReLU → MaxPool2d(2)

- Block 3: Conv2d(32 → 64, kernel=3, padding=1) → ReLU → MaxPool2d(2)



```
RPS_CNN(
  (conv_block): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=16384, out_features=256, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=256, out_features=3, bias=True)
  )
)
```

2. Dimensionality Reduction

The input image size is 128×128. After passing through three MaxPool layers (each with a stride of 2), the spatial resolution is halved three times:

128 → 64 → 32 → 16

The final feature map has a depth of 64 channels and a spatial size of 16×16. This is flattened into a vector of size 64 × 16 × 16 = 16,384 neurons.

3. Fully Connected Classifier

The flattened vector is passed through a Multilayer Perceptron (MLP) for final classification:

- **Input: Flattened vector (16,384)**
- **Hidden Layer: Linear(16384 → 256) → ReLU**
- **Regularization: Dropout(p=0.3) (Randomly zeroes 30% of neurons to prevent overfitting)**
- **Output Layer: Linear(256 → 3) (Outputs logits for rock, paper, scissors)**

III. TRAINING AND PERFORMANCE

1. Hyperparameters

The following hyperparameters were selected for the training process:

- Optimizer: Adam (Adaptive Moment Estimation)
- Loss Function: CrossEntropyLoss (Suitable for multi-class classification)
- Learning Rate: 0.001
- Batch Size: 32
- Epochs: 10

2. Training Results

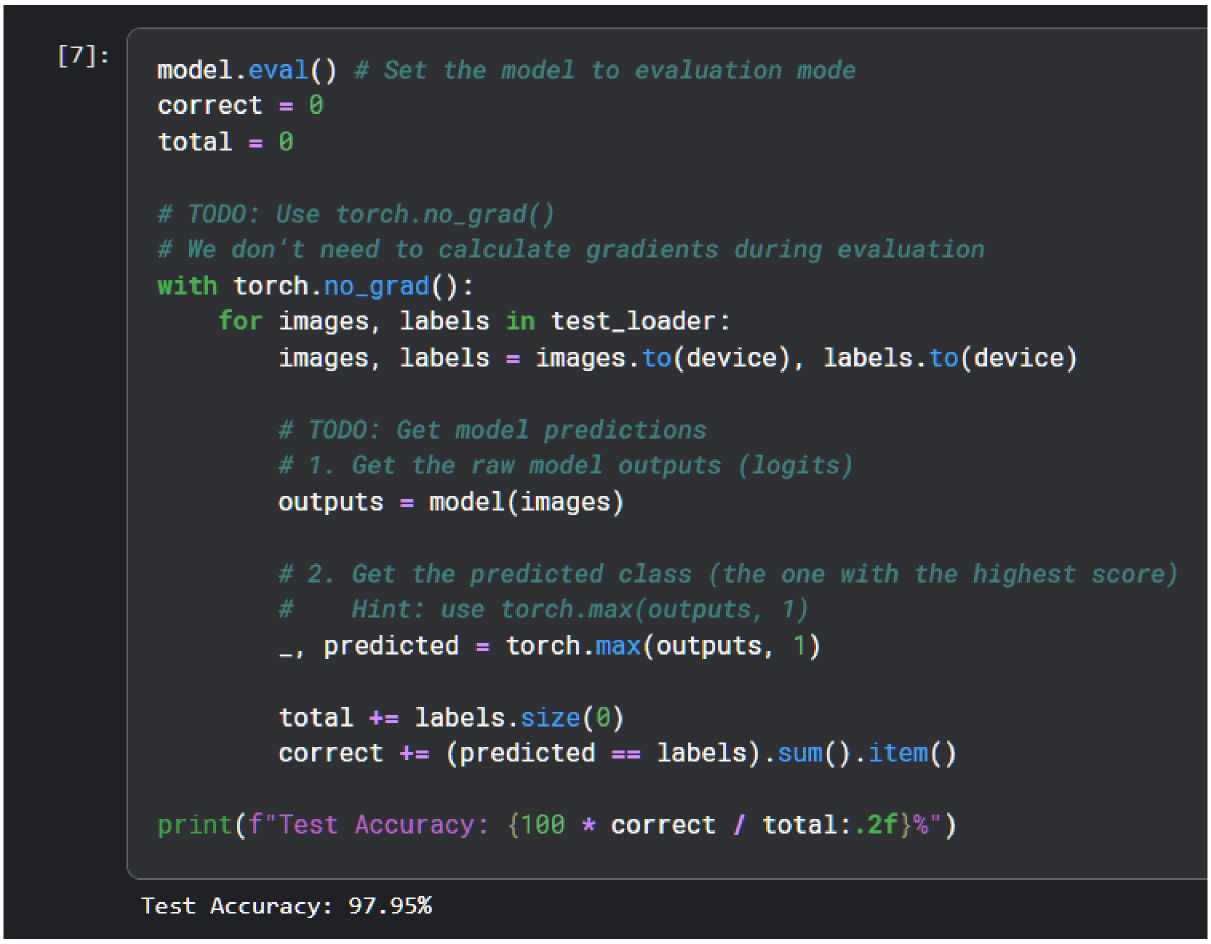
The model showed consistent convergence over the 10 epochs.

- Initial Loss (Epoch 1): 0.6905
- Final Loss (Epoch 10): 0.0070

3. Evaluation

After training, the model was evaluated on a held-out test set comprising 20% of the total data.

- **Final Test Accuracy: 97.95%**



```
[7]: 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: 97.95%
```

The model was also tested on individual random samples and a simulated game loop, where it correctly predicted input images (e.g., correctly identifying "paper" in the single image test).

IV. CONCLUSION AND ANALYSIS

Conclusion

The implemented CNN achieved a high accuracy of 97.95% on the test dataset. The rapid decrease in loss from 0.6905 to 0.0070 indicates that the model successfully learned the distinguishing features of the hand gestures without significant issues. The architecture's depth (3 convolutional layers) was sufficient to capture the complexity of the dataset.

Challenges Faced

- Dimension Mismatch: Calculating the correct input size for the first Linear layer required tracking the image size reduction through the pooling layers (128 → 16).

- Overfitting Risks: With a high number of parameters in the fully connected layer (>4 million weights), there was a risk of memorizing the data. This was mitigated using the Dropout layer (p=0.3).

Potential Improvements

1. Data Augmentation: Currently, the model trains on static images. Adding random rotations, horizontal flips, or brightness adjustments would make the model more robust to real-world variations.
2. Batch Normalization: Adding BatchNorm2d layers after each convolution could speed up convergence and allow for higher learning rates.
3. Scheduler: Implementing a Learning Rate Scheduler to decrease the learning rate as the loss plateaus could help fine-tune the weights in later epochs.