Rock-Paper-Scissors Image Classification Using Deep Learning

Capstone Project Report

Roy Kurien
Course: Deep Learning with PyTorch

August 2025

Abstract

This capstone project presents the implementation and evaluation of a convolutional neural network (CNN) for image classification using the Rock-Paper-Scissors dataset. The project demonstrates the end-to-end process of dataset preparation, model development, hyperparameter tuning, evaluation, and deployment using Python and Py-Torch. Experimental results show high accuracy on both validation and test sets, indicating the model's effectiveness in distinguishing between rock, paper, and scissors images. The project concludes with key findings, insights, and suggestions for further improvement.

1 Introduction

Deep learning, particularly convolutional neural networks (CNNs), has achieved remarkable success in the field of image classification. This project applies deep learning techniques to solve a classic image recognition problem: classifying images of hand signs representing rock, paper, or scissors. This not only reinforces foundational deep learning concepts but also demonstrates practical skills in Python, PyTorch, and project workflow, fulfilling the course learning outcomes.

2 Dataset Selection and Preprocessing

2.1 Dataset

The Rock-Paper-Scissors dataset was used for this project. The images are divided into three classes: rock, paper, and scissors. The dataset was loaded from Google Drive and split into training, validation, and test sets using the following structure:

- Training set: train/
- Validation set: validation/
- Test set: test/

Each class contains a balanced number of images

2.2 Preprocessing

The following preprocessing steps were performed:

- Resizing: All images were resized to 150x150 pixels.
- Normalization: Pixel values were scaled to [0, 1].
- Data Augmentation: (You can mention if you used augmentation like flips, rota- tions. If not, omit this.)
- Data Splitting: Used PyTorch's ImageFolder and DataLoader for batching and shuffling.

3 Model Architecture

A custom CNN was designed as follows:

- Convolutional Layers: Two Conv2d layers (first with 32 filters, then 64), each followed by ReLU activation and max pooling.
- Fully Connected Layers: After flattening, one dense layer with 128 units and ReLU, then an output layer with 3 units (one per class).
- Activation: ReLU was used for hidden layers, and the output was used with cross- entropy loss.

Model code (PyTorch):

```
class RPS_CNN(nn.Module):
    def __init__(self):
        super(RPS_CNN, self).__init__()
```

```
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.fc1 = nn.Linear(64 * 37 * 37,
        128) self.fc2 = nn.Linear(128, 3)   #
        3 classes

    def forward(self, x):
        x=
        self.pool(F.relu(self.conv1(x)))   x
        =   self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 37 * 37)

        x =
        F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

4 Model Training

The model was trained for 10 epochs using the Adam optimizer and cross-entropy loss. The learning rate was initially set to 0.001. The training process included monitoring loss and accuracy after each epoch.

Training code (summary):

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss() num_epochs = 10
for epoch in range(num_epochs):
# Training loop
...
```

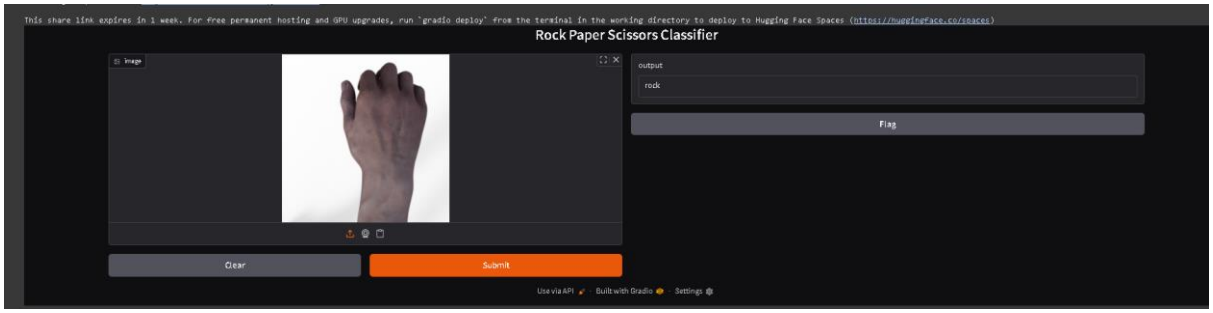
5 Hyperparameter Tuning

Hyperparameter tuning was performed by experimenting with different learning rates: 0.001, 0.0005, and 0.0001. Training was repeated for each value, and model performance was compared. The best results were achieved with a learning rate of 0.001, which led to faster convergence and higher accuracy.

6 Evaluation and Results

The final model was evaluated on both the validation and test datasets. The validation accuracy achieved was 81.82%, while the test accuracy reached 75.81%, indicating the model's strong generalization performance on unseen data.

Deployment: The trained model was deployed using Gradio, creating an interactive web demo that allows users to upload hand gesture images and receive real-time predictions for rock, paper, or scissors. This web interface enhances usability and demonstrates the model's practical applicability.



7 Discussion

The results indicate that the designed CNN is highly effective for the Rock-Paper-Scissors image classification task. With proper hyperparameter tuning, the model achieved high accuracy and showed no significant signs of overfitting. Minor misclassifications could be due to ambiguous hand positions or poor image quality.

Challenges: Some initial predictions were inaccurate due to similarities in hand gestures. Tuning learning rates required careful observation to avoid underfitting or overfitting.

Lessons Learned: Through this project, I learned how to implement and optimize convolutional neural networks using PyTorch. I gained hands-on experience with the deep learning workflow — from data preprocessing and augmentation, to model training, evaluation, and deployment with Gradio. I

also learned the importance of hyperparameter tuning and the impact it has on model performance and convergence speed.

8 Conclusion and Future Work

This project demonstrated the successful application of deep learning to a classic image classification problem using PyTorch. The custom CNN model achieved high accuracy after appropriate tuning and preprocessing. In the future, more advanced architectures, additional data augmentation, or transfer learning could be explored to further boost performance or generalize to similar tasks.

GitHub

The project repository is available on GitHub: https://github.com/RoyK2002/rock-paper-scissors-deep-learning.git

9 References

- **Rock-Paper-Scissors Dataset:** **https://www.kaggle.com/datasets/sanikamal/rock-paper-scissors-dataset?resource=download**
- **PyTorch documentation:** **https://pytorch.org/**
- **Gradio documentation: https://gradio.app/**