

# ML Lab Report Week 14: CNN Image Classification

NAME: Deepthi J

SRN: PES2UG23CS164

SECTION :5C

## 1) Introduction

The objective of this lab was to design, implement, and train a Convolutional Neural Network (CNN) capable of classifying images of hand gestures into three categories: rock, paper, and scissors. Using the Rock–Paper–Scissors dataset, the task involved data preprocessing, model construction in PyTorch, training, testing, and evaluating real predictions. The lab provided hands-on experience with convolutional architectures, activation functions, pooling mechanisms, and end-to-end deep learning workflows for image classification.

## 2. Model Architecture

The CNN model developed in this lab is composed of three sequential convolutional blocks, each designed to extract increasingly complex spatial features from the input images.

Convolutional Block 1:

- Conv2d layer:  $3 \rightarrow 16$  channels
- Kernel size:  $3 \times 3$ , padding 1
- Activation: ReLU
- Downsampling: MaxPool2d ( $2 \times 2$ )

Convolutional Block 2:

- Conv2d layer:  $16 \rightarrow 32$  channels
- Kernel size:  $3 \times 3$ , padding 1
- Activation: ReLU
- Downsampling: MaxPool2d ( $2 \times 2$ )

Convolutional Block 3:

- Conv2d layer:  $32 \rightarrow 64$  channels
- Kernel size:  $3 \times 3$ , padding 1
- Activation: ReLU
- Downsampling: MaxPool2d ( $2 \times 2$ )

Each MaxPooling layer reduces the spatial dimensions by half, transforming the original  $128 \times 128$  input into  $16 \times 16$  feature maps after three pooling operations. These features are then flattened and passed into a fully connected classifier, which includes:

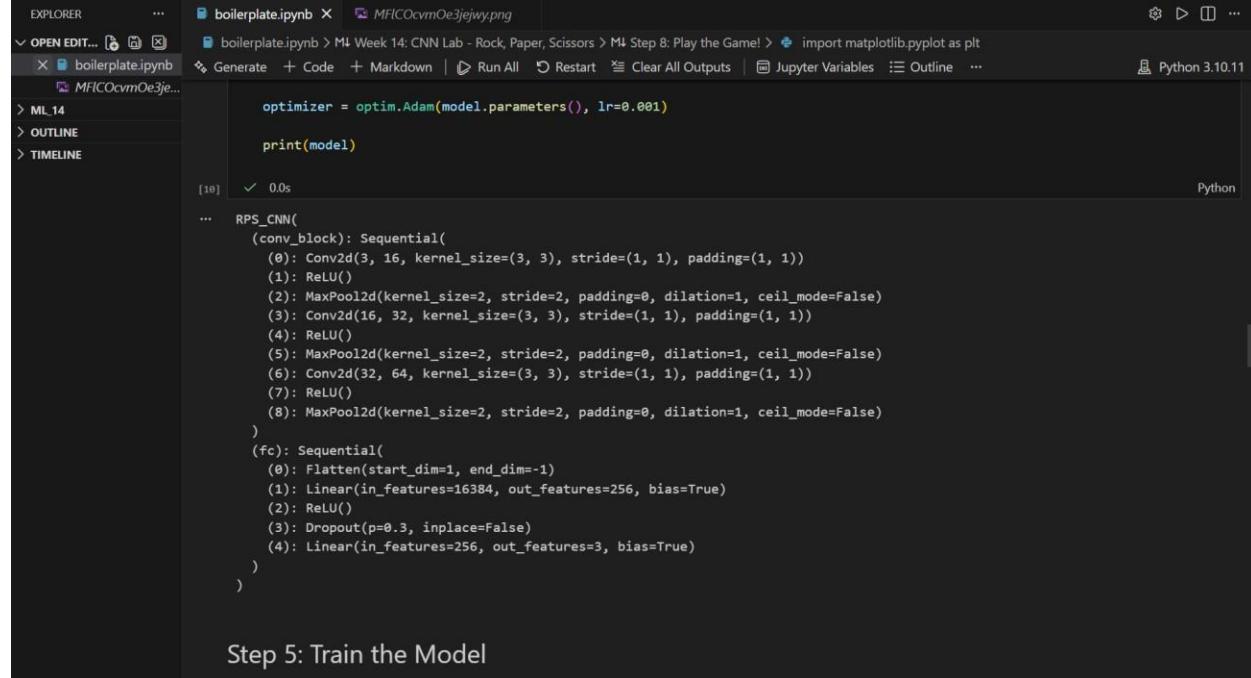
- A Linear layer mapping  $64 \times 16 \times 16 \rightarrow 256$
- ReLU activation for non-linearity
- Dropout ( $p = 0.3$ ) to reduce overfitting
- A final Linear layer mapping  $256 \rightarrow 3$ , producing logits for the three gesture classes. This architecture balances depth and computational efficiency, enabling effective feature extraction and robust classification performance for the Rock–Paper–Scissors dataset.

### 3. Training and Performance

The model was trained using the following hyperparameters:

- Optimizer: Adam
- Loss Function: CrossEntropyLoss
- Learning Rate: 0.001
- Epochs: 10
- Batch Size: 32

#### Defining the model



```

EXPLORER          boilerplate.ipynb  MFLCOcvmOe3ejwy.png
OPEN EDIT...      boilerplate.ipynb
boilerplate.ipynb
MFLCOcvmOe3ejw...
ML_14
OUTLINE
TIMELINE

optimizer = optim.Adam(model.parameters(), lr=0.001)

print(model)

[10] ✓ 0.0s
Python

... 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)
    )
)

```

Step 5: Train the Model

Training the model:

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'boilerplate.ipynb' and the Python version 'Python 3.10.11'. The left sidebar has sections for 'OPEN EDITORS', 'ML 14', 'OUTLINE', and 'TIMELINE'. The main area contains a code cell with the following content:

```
loss = criterion(outputs, labels)

# 4. Backward pass
loss.backward()

# 5. Update weights
optimizer.step()

total_loss += loss.item()

print(f"Epoch {epoch+1}/{EPOCHS}, Loss = {total_loss/len(train_loader):.4f}")

print("Training complete!")

[11] ✓ 4m 43.4s
```

The output cell [11] shows the training progress from epoch 1 to 10, followed by a message 'Training complete!'. The bottom right corner of the main area says 'Python'.

## Evaluate the Model:

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'boilerplate.ipynb' and the Python version 'Python 3.10.11'. The left sidebar has sections for 'OPEN EDITORS', 'ML 14', 'OUTLINE', and 'TIMELINE'. The main area contains a code cell with the following content:

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

# TODO: Use torch.no_grad()
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # 1. Get model outputs (logits)
        outputs = model(images)

        # 2. Get predicted class
        _, predicted = torch.max(outputs, 1)

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

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

The output cell [12] shows the test accuracy as 97.26%. The bottom right corner of the main area says 'Python'.

The dataset was split into 80% training and 20% testing. After training, the model achieved a final test accuracy of 97.26%, demonstrating strong generalization to unseen images.

Testing the model:

```
def predict_image(model, img_path):
    model.eval()

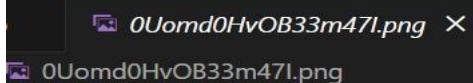
    img = Image.open(img_path).convert("RGB")
    img = transform(img).unsqueeze(0).to(device) # Apply transforms + batch dimension

    with torch.no_grad():
        # 1. Get raw model outputs (logits)
        output = model(img)

        # 2. Get predicted class index
        _, pred = torch.max(output, 1)

    return class_names[pred.item()]
# test the prediction function
test_img_path = "./dataset/paper/0Uomd0HvOB33m47I.png"
prediction = predict_image(model, test_img_path)
print(f"Model prediction for {test_img_path}: {prediction}")

[13] ✓ 0.0s
... Model prediction for ./dataset/paper/0Uomd0HvOB33m47I.png: paper
```



Play the game:

```

# 3. Decide the winner
# -----
# print("\nRESULT:", rps_winner(p1, p2))

[15] ✓ 0.0s

... Randomly selected images:
Image 1: ./dataset/scissors\uQLROCDZVtwVCXfm.png
Image 2: ./dataset/scissors\MF1COcvm0e3jejwy.png

Player 1 shows: scissors
Player 2 shows: scissors

RESULT: Draw

import matplotlib.pyplot as plt

# Display the two random images
plt.figure(figsize=(8, 4))

```

The screenshot shows a Jupyter Notebook interface with two code cells and their outputs. The top cell contains code to decide a winner between two randomly selected images of scissors. The output shows both images and a draw result. The bottom cell displays two side-by-side images of hands showing the 'scissors' gesture against a green background.

#### 4. Conclusion and Analysis

The model performed extremely well, achieving 97.26% accuracy on the test set. The CNN successfully learned to distinguish among the three gesture classes, validating the effectiveness of the chosen architecture. Challenges included setting up the dataset paths correctly and ensuring consistent preprocessing. Future enhancements that could further improve performance include applying data augmentation techniques or expanding the network with additional layers or batch normalization to improve feature extraction and stability.