

# ML Lab Week 14: CNN Image Classification

Name	Diya D Bhat
SRN	PES2UG23CS183
Section	C
Date	20/11/25
Course name	Machine Learning

## 1. Introduction

In this lab, we built and trained a Convolutional Neural Network (CNN) to classify images of rock, paper, and scissors. The goal was to understand how image preprocessing, convolutional layers, pooling, and fully connected layers work together to perform image recognition. We also learned how to train the model, evaluate it, and test it on new images.

## 2. Model Architecture

The CNN consists of three convolutional blocks, each using a 3x3 kernel, ReLU activation, and MaxPool2d(2) to reduce spatial size.

- Block 1: 3 → 16 channels
- Block 2: 16 → 32 channels
- Block 3: 32 → 64 channels

After pooling three times, the image reduces from  $128 \times 128 \rightarrow 16 \times 16$ , which is flattened and passed into the fully connected classifier which has the following features:

- Linear layer from  $64 \times 16 \times 16 \rightarrow 256$
- Activation = ReLU activation
- Dropout = 0.3
- Final linear layer  $256 \rightarrow 3$  for class prediction.

**Screenshot of model structure:**

The screenshot shows a Jupyter Notebook interface with the following code:

```
# 2. Define the loss function (Criterion). Use CrossEntropyLoss for classification.  
criterion = nn.CrossEntropyLoss() # <-- Replace this  
  
# 3. Define the optimizer. Use Adam with a learning rate of 0.001  
optimizer = optim.Adam(model.parameters(), lr=0.001) # <-- Replace this  
  
print(model)  
  
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)  
    )  
)
```

### 3. Training and Performance

The model was trained using:

- Optimizer: Adam
- Learning rate: 0.001
- Loss function: CrossEntropyLoss
- Epochs: 10

After training, the model achieved a final test accuracy of 99.09% (replace with your actual value).

#### Screenshot of model training:

A screenshot of a Jupyter Notebook interface. The title bar shows "PES2UG23CS183\_Week14\_Lab.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with search, command, code, text, and run all buttons. The main area contains Python code for training a model:

```
# 3. Calculate the loss (using criterion)
loss = criterion(outputs, labels)

# 4. Perform a backward pass (loss.backward())
loss.backward()

# 5. Update the weights (optimizer.step())
optimizer.step()

total_loss += loss.item()

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

print("Training complete!")

...
Epoch 1/10, Loss = 0.6532
Epoch 2/10, Loss = 0.1969
Epoch 3/10, Loss = 0.0890
Epoch 4/10, Loss = 0.0600
Epoch 5/10, Loss = 0.0289
Epoch 6/10, Loss = 0.0327
Epoch 7/10, Loss = 0.0331
Epoch 8/10, Loss = 0.0230
Epoch 9/10, Loss = 0.0075
Epoch 10/10, Loss = 0.0051
Training complete!
```

## Screenshot of model evaluation and test accuracy:

A screenshot of a Jupyter Notebook interface. The title bar shows "PES2UG23CS183\_Week14\_Lab.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with search, command, code, text, and run all buttons. The main area contains Python code for evaluating a model:

```
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(): # <-- Replace this
    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) # <-- Replace this

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

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

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

The output below the code cell shows the result: "Test Accuracy: 99.09%".

## Screenshot of the final output:

```
Randomly selected images:  
Image 1: /content/dataset/scissors/RTCZxs3LhwklhSKk.png  
Image 2: /content/dataset/paper/Gv71lr2m5b9rBwPy.png  
  
Player 1 shows: scissors  
Player 2 shows: paper  
  
RESULT: Player 1 wins! scissors beats paper
```

## 4. Conclusion and Analysis

The model performed well for a simple CNN, showing that the architecture can learn meaningful features from the RPS images. A challenge was ensuring the dataset was correctly loaded and transformed. To further improve accuracy, we could add data augmentation (rotation, flips) or use a deeper network such as a pre-trained model. These changes would help the model generalize better.