Name:- MOHAMMED SHAZI UL ISLAM

SRN:- PES2UG23CS348

1 Week 14: CNN Lab - Rock, Paper, Scissors

Objective: Build, train, and test a Convolutional Neural Network (CNN) to classify images of hands playing Rock, Paper, or Scissors.

1.0.1 Step 1: Setup and Data Download

This first cell downloads the dataset from Kaggle.

```
import shutil import
os

src_root = "/kaggle/input/rockpaperscissors"

dst_root = "/content/dataset" os.makedirs(dst_root,

exist_ok=True) folders_to_copy = ["rock", "paper",

"scissors"]

for folder in folders_to_copy:
    src_path = os.path.join(src_root, folder) dst_path
    = os.path.join(dst_root, folder)
```

[10]:
```
        if os.path.exists(src_path):                1
            shutil.copytree(src_path, dst_path, dirs_exist_ok=True)
            print("Copied:", folder)
    else
```

```
import kagglehub

path = kagglehub.dataset_download("drgfreeman/rockpaperscissors")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/rockpaperscissors
```

[11]:

:

```
Copied: scissors
```

1.0.2 Step 2: Imports and Device Setup

Import the necessary libraries and check if a GPU is available.

[12]:
```
import os import torch
import torch.nn as nn import
torch.optim as optim
from torchvision import datasets, transforms from
torch.utils.data import DataLoader, random_split from PIL
import Image import numpy as np

# Set the 'device' variable
# Check if CUDA (GPU) is available, otherwise use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

Using device: cuda

1.0.3 Step 3: Data Loading and Preprocessing

Here we will define our image transformations, load the dataset, split it, and create DataLoaders.

[13]:
```
DATA_DIR = "/content/dataset"

# Define the image transforms
# 1. Resize all images to 128x128
# 2. Convert them to Tensors
# 3. Normalize them (mean=0.5, std=0.5)
transform = transforms.Compose([
transforms.Resize((128, 128)),
transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), ])

# Load dataset using ImageFolder
full_dataset = datasets.ImageFolder(DATA_DIR, transform=transform)

class_names = full_dataset.classes print("Classes:",
class_names)

# Split the dataset: 80% train, 20% test total =
len(full_dataset)
```

```
train_size = int(0.8 * total) test_size =
total - train_size

# Use random_split to create train_dataset and test_dataset train_dataset,
test_dataset = random_split(full_dataset, [train_size, test_size])


# Create the DataLoaders
# Use a batch_size of 32; shuffle training loader but not test loader train_loader =
DataLoader(train_dataset, batch_size=32, shuffle=True) test_loader =
DataLoader(test_dataset, batch_size=32, shuffle=False)

print(f"Total images: {len(full_dataset)}") print(f"Training
images: {len(train_dataset)}") print(f"Test images:
{len(test_dataset)}")
```

```
Classes: ['paper', 'rock', 'scissors']
Total images: 2188
Training images: 1750
Test images: 438
```

1.0.4 Step 4: Define the CNN Model

Fill in the conv_block and fc_block with the correct layers.

[14]:
```
class RPS_CNN(nn.Module):
    def _init_(self):
        super(RPS_CNN, self)._init_()

        # Define the convolutional block
        # 3 conv blocks with increasing channels and max-pooling
        self.conv_block = nn.Sequential( nn.Conv2d(3, 16,
        kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2),

            nn.Conv2d(16, 32, kernel_size=3,
            padding=1), nn.ReLU(), nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3,
            padding=1), nn.ReLU(), nn.MaxPool2d(2),
        )

        # After 3 MaxPool(2) layers, our 128x128 image becomes:
        # 128 -> 64 -> 32 -> 16
```

```
        # So the flattened size is 64 * 16 * 16

        # Define the fully-connected (classifier) block self.fc =
        nn.Sequential( nn.Flatten(),
            nn.Linear(64 * 16 * 16, 256),
            nn.ReLU(), nn.Dropout(0.3),
            nn.Linear(256, 3), )

    def forward(self, x): x =
        self.conv_block(x) x =
        self.fc(x) return x

# Initialize the model, criterion, and optimizer
# 1. Create an instance of RPS_CNN and move it to the 'device' model =
RPS_CNN().to(device)

# 2. Define the loss function (Criterion). Use CrossEntropyLoss for␣ sclassification.
criterion = nn.CrossEntropyLoss()

# 3. Define the optimizer. Use Adam with a learning rate of 0.001 optimizer =
optim.Adam(model.parameters(), lr=0.001) 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)
  )
)
```

1.0.5 Step 5: Train the Model

Fill in the core training steps inside the loop.

[15]:
```
EPOCHS = 10

for epoch in range(EPOCHS):
    model.train() # Set the model to training mode total_loss = 0

    for images, labels in train_loader: # Move data to the correct
        device images, labels = images.to(device),
        labels.to(device)

        # Training steps
        optimizer.zero_grad() outputs =
        model(images) loss =
        criterion(outputs, labels)
        loss.backward() optimizer.step()
        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{EPOCHS}, Loss = {total_loss/len(train_loader):.
  s4f}") print("Training
complete!")
```

```
Epoch 1/10, Loss = 0.6277
Epoch 2/10, Loss = 0.1329
Epoch 3/10, Loss = 0.0746
Epoch 4/10, Loss = 0.0597
Epoch 5/10, Loss = 0.0248
```

Epoch 6/10, Loss = 0.0097
Epoch 7/10, Loss = 0.0062
Epoch 8/10, Loss = 0.0126
Epoch 9/10, Loss = 0.0109
Epoch 10/10, Loss = 0.0081 Training complete!

1.0.6 Step 6: Evaluate the Model

Test the model's accuracy on the unseen test set.

[16]:

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

# 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)

        # 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%

1.0.7 Step 7: Test on a Single Image

Let's see how the model performs on one image.

[17]:
```
def predict_image(model, img_path):
    model.eval()

    img = Image.open(img_path).convert("RGB")
    # Apply the same transforms as training, and add a batch dimension␣
 ↪(unsqueeze) img =
    transform(img).unsqueeze(0).to(device)

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

        # Get the predicted class index _, pred
    =  torch.max(output,  1)  return
    class_names[pred.item()]

# Test the function (this path should exist)
test_img_path = "/content/dataset/paper/0Uomd0HvOB33m47I.png"
prediction = predict_image(model, test_img_path)
print(f"Model prediction for {test_img_path}: {prediction}")
```

Model prediction for /content/dataset/paper/0Uomd0HvOB33m47I.png: paper

1.0.8 Step 8: Play the Game!

This code is complete. If your model is trained, you can run this cell to have the model play against itself.

[18]:
```python
import random
import os

def pick_random_image(class_name):
    folder = f"/content/dataset/{class_name}"
    files = os.listdir(folder)
    img = random.choice(files)
    return os.path.join(folder, img)

def rps_winner(move1, move2):
    if move1 == move2:
        return "Draw"

    rules = {
        "rock": "scissors",
        "paper": "rock",
        "scissors": "paper"
    }

    if rules[move1] == move2:
        return f"Player 1 wins! {move1} beats {move2}"
    else:
        return f"Player 2 wins! {move2} beats {move1}"


# ---------------------------------------------------------------
# 1. Choose any two random classes
# ---------------------------------------------------------------

choices = ["rock", "paper", "scissors"]
c1 = random.choice(choices)
c2 = random.choice(choices)

img1_path = pick_random_image(c1)
img2_path = pick_random_image(c2)

print("Randomly selected images:")
print("Image 1:", img1_path)
print("Image 2:", img2_path)
```

```
# ------------------------------------------------------------
# 2. Predict their labels using the model
# ------------------------------------------------------------

p1 = predict_image(model, img1_path)
p2 = predict_image(model, img2_path)

print("\nPlayer 1 shows:", p1)
print("Player 2 shows:", p2)

# ------------------------------------------------------------
# 3. Decide the winner
# ------------------------------------------------------------

print("\nRESULT:", rps_winner(p1, p2))
```

Randomly selected images:

Image 1: /content/dataset/rock/EvmNpXcSU8y41PXX.png

Image 2: /content/dataset/paper/eaOtD5yLQHTuFTz3.png

Player 1 shows: rock

Player 2 shows: paper

RESULT: Player 2 wins! paper beats rock

REPORT-

1. Introduction

The objective of this lab was to design, build, and train a Convolutional Neural Network (CNN) using the **PyTorch** framework [3]to classify images of hand gestures[4]. The model was trained on the "Rock Paper Scissors" dataset [5], which contains over 2,000 images categorized into three classes: **'rock'**, **'paper'**, and **'scissors'**[6666]. The final goal was to evaluate the model's classification accuracy on a held-out test set[7].

---

2. Model Architecture

The CNN model, defined as `RPS_CNN`, follows a standard architecture consisting of a **convolutional block** for feature extraction and a **fully-connected block** for classification[9].

Convolutional Block (Feature Extractor)

The feature extractor comprises three sequential blocks, each using a convolution, ReLU activation, and max-pooling[10]. The input images are resized to 128×128[11].

Layer Type	Input Channe	Output Channel	Kernel Size	Stride/Poo	Output Feature Size
Conv + Re MaxPool	3	16	$3 \tim 3$ [14]	MaxPool $ 2$ [15]	$16 \times 64 \times 64$
Conv + Re MaxPool	16	32	$3 \tim 3$ [16]	MaxPool $ 2$ [17]	$32 \times 32 \times 32$
Conv + Re MaxPool	32	64	$3 \tim 3$ [18]	MaxPool $ 2$ [19]	$64 \times 16 \times 16$

Fully-Connected Block (Classifier) [20]

The final feature map ($64 \times 16 \times 16$) is flattened and passed through a classifier consisting of two linear layers[21]:

- **Input Size:** The `Flatten` layer converts the $64 \times 16 \times 16$ feature map into a vector of **16,384** features[222222].
- **Hidden Layer:** `Linear` layer (16,384 inputs, **256 outputs**) [23]$\rightarrow$ `ReLU` activation[24]. A **Dropout** layer with $p=0.3$ is applied for regularization[25252525].
- **Output Layer:** The final `Linear` layer maps the 256 outputs to **3** classes[26].

---

3. Training and Performance

Key Hyperparameters

Hyperparameter	Value
Optimizer	`optim.Adam`
Loss Function	`nn.CrossEntropyLoss()`
Learning Rate	0.001

Hyperparameter	Value
Number of Epochs	10 [32]
Batch Size	32
Data Split	80% Train (1750 images), 20% Test (438 images)
Device Used	`cuda` (GPU) 35353535
Final Test Accuracy	

The model was evaluated on the unseen test dataset after 10 epochs.

Final Test Accuracy: 98.63%

---

4. Conclusion and Analysis

Discussion of Results

The model performed **exceptionally well**, achieving a high Test Accuracy of **98.63%**[40]. This demonstrates that the deep CNN architecture effectively extracted the salient features needed to distinguish between the three hand gestures. The training process showed rapid convergence, with the loss dropping significantly after the first few epochs [41], indicating that the chosen hyperparameters and model design were suitable for this image classification task.

Challenges Faced

The primary "challenge" was conceptual: ensuring the correct calculation for the number of input features to the first fully-connected layer after three rounds of max-pooling on the 128×128 input images. This was successfully resolved by determining the flattened size as $64 \times 16 \times 16 = 16,384$[43].

Potential Improvements

1. **Introduce Advanced Data Augmentation:** To make the model more robust to new, real-world images, the current transformation pipeline could be expanded to include more aggressive techniques like **Random Affine Transformations** (e.g., small rotations, shearing, and horizontal flips) [45].
2. **Experiment with Transfer Learning:** Instead of training a model from scratch, a state-of-the-art pre-trained model like **ResNet** or **VGG** could be used as the feature extractor and fine-tuned on the Rock, Paper, Scissors dataset. This approach often leads to higher accuracy and faster training convergence for smaller datasets.

1
1