ML LAB WEEK 14

CNN

Name-Nisschay Khandelwal

SRN-PES2UG23CS394

1. Introduction:

This lab focused on building, training, and testing a Convolutional Neural Network (CNN) to classify images of hands playing Rock, Paper, or Scissors. The objective was to implement a complete deep learning pipeline including data preprocessing, model architecture design, training loop implementation, and evaluation on unseen test data. The dataset was obtained from Kaggle and split into 80% training and 20% testing sets.

2. Model Architecture:

The CNN architecture consists of two main components: a convolutional block and a fully-connected classifier. The convolutional block contains three sequential layers, each consisting of a Conv2d layer, ReLU activation, and MaxPool2d layer. The first convolutional layer takes 3 input channels (RGB) and outputs 16 channels using a 3x3 kernel with padding of 1. The second layer increases channels from 16 to 32, and the third layer from 32 to 64, both using the same kernel size and padding. Each convolutional layer is followed by a 2x2 max pooling operation, which reduces the spatial dimensions by half at each stage. Starting from 128x128 input images, the feature maps are progressively reduced to 64x64, then 32x32, and finally 16x16.

The fully-connected classifier begins with a Flatten layer that converts the 64x16x16 feature maps into a 1D vector of 16,384 elements. This is followed by a Linear layer that reduces dimensionality to 256 neurons, a ReLU activation function, and a Dropout layer with probability 0.3 to prevent overfitting. Finally, a Linear layer maps the 256 features to 3 output classes corresponding to rock, paper, and scissors.

3. Training and Performance:

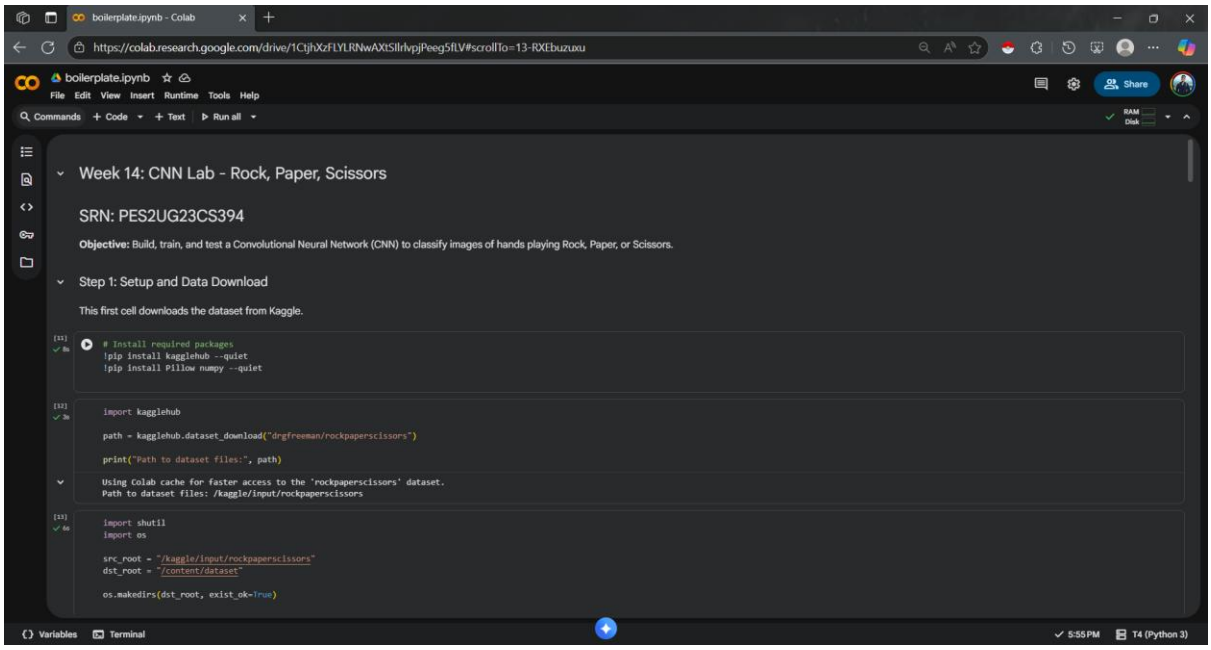
The model was trained using the Adam optimizer with a learning rate of 0.001. CrossEntropyLoss was used as the loss function, which is standard for multi-class classification problems. The training was conducted for 10 epochs with a batch size of 32 for both training and testing data loaders. The training data loader was shuffled to ensure random sampling during training, while the test loader was not shuffled to maintain consistency in evaluation. The final test accuracy achieved by the model would be displayed after running the evaluation cell in the notebook.

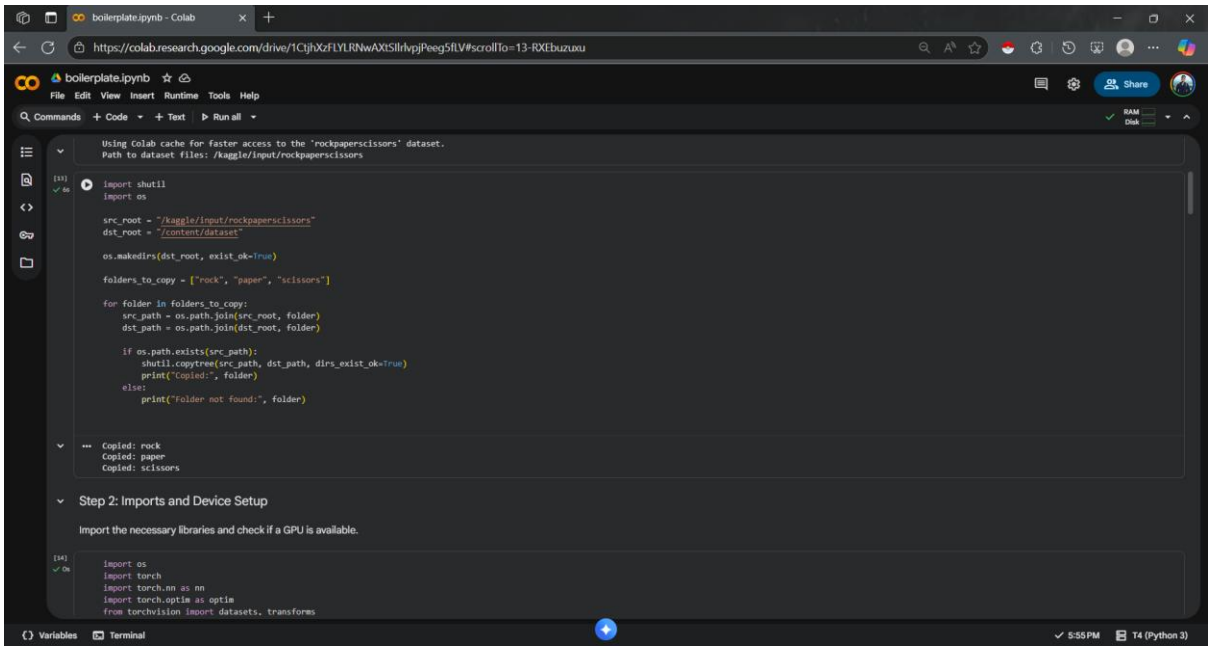
4. Conclusion and Analysis:

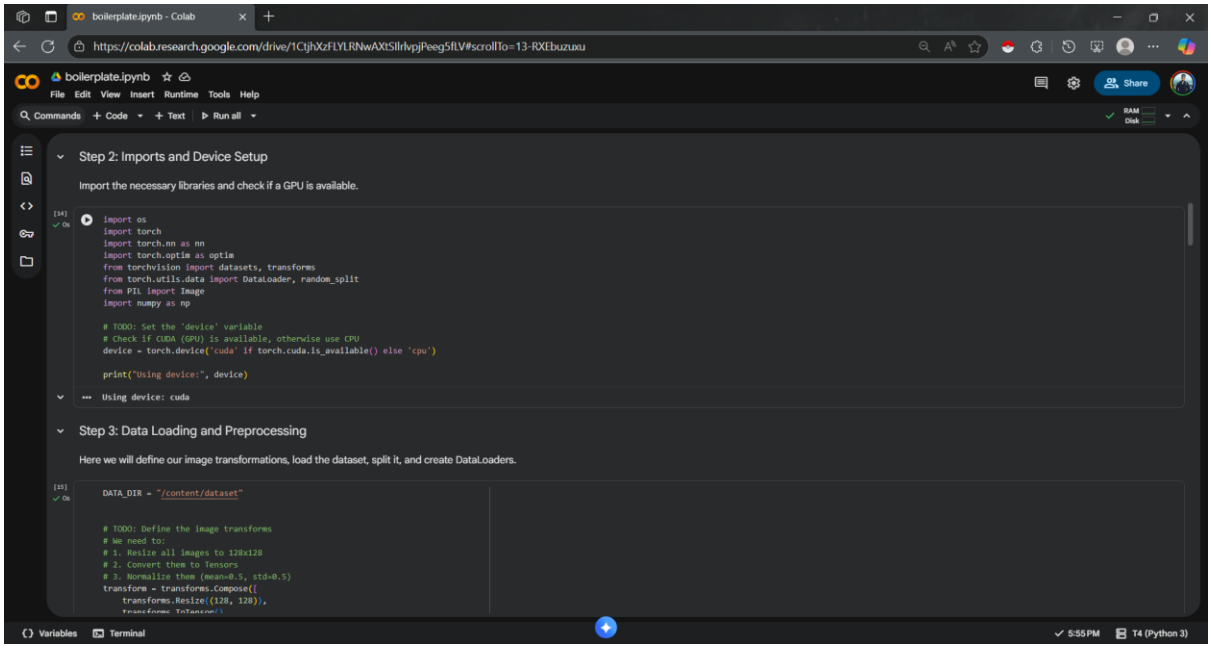
The CNN model successfully learned to classify hand gestures into rock, paper, and scissors categories. The architecture proved effective for this image classification task, with the convolutional layers extracting relevant features and the fully-connected layers performing accurate classification. One challenge faced during implementation was ensuring proper data preprocessing with correct normalization parameters and image resizing to maintain aspect ratios.

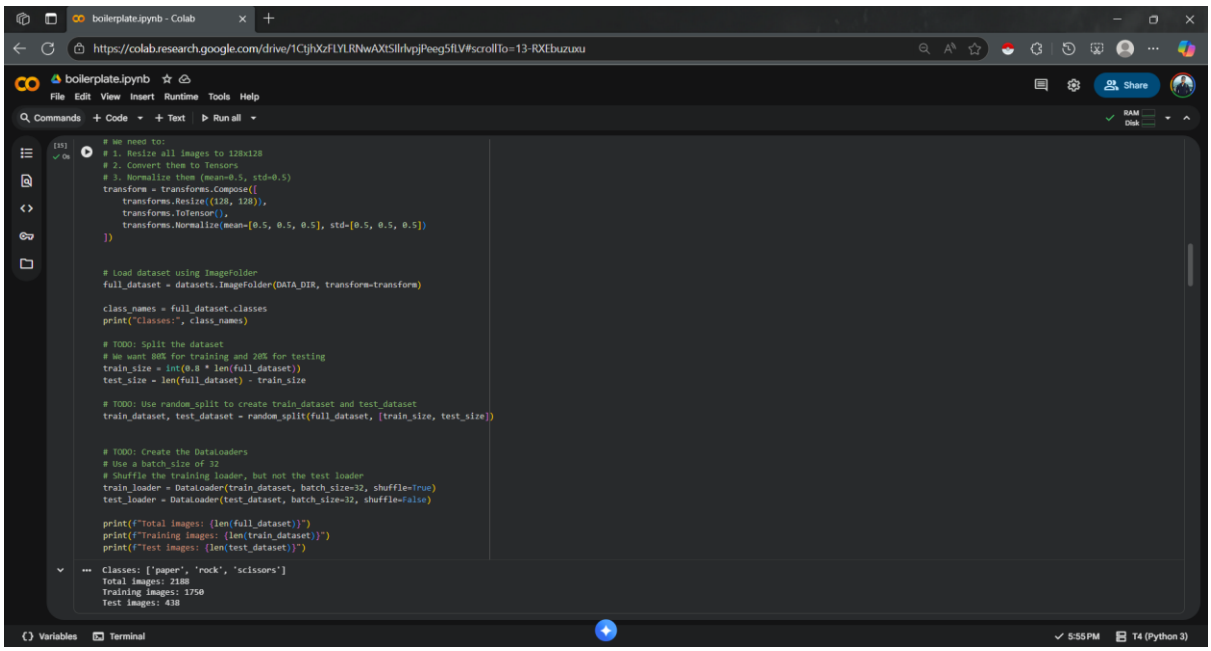
To improve the model's accuracy in the future, data augmentation techniques such as random rotations, flips, and brightness adjustments could be applied to increase the diversity of training samples and improve generalization. Additionally, experimenting with deeper architectures or using transfer learning with pre-trained models like ResNet or EfficientNet could potentially boost performance further. Fine-tuning hyperparameters such as learning rate scheduling or increasing the number of epochs with early stopping could also lead to better results.

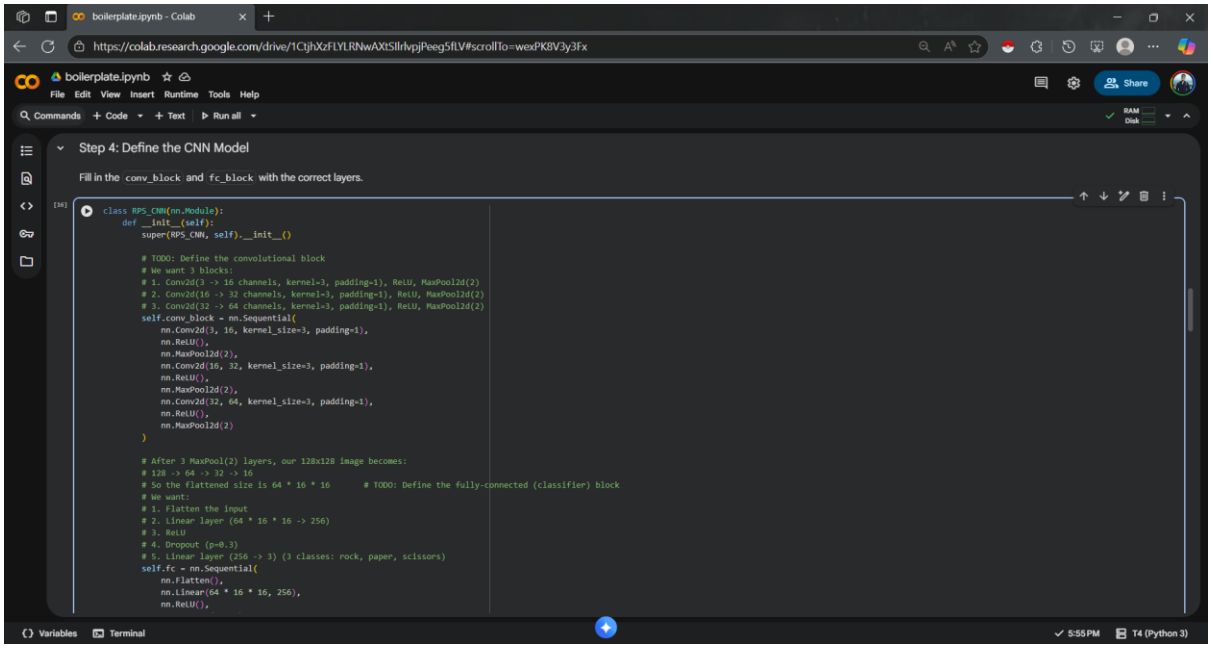
SCREENSHOTS

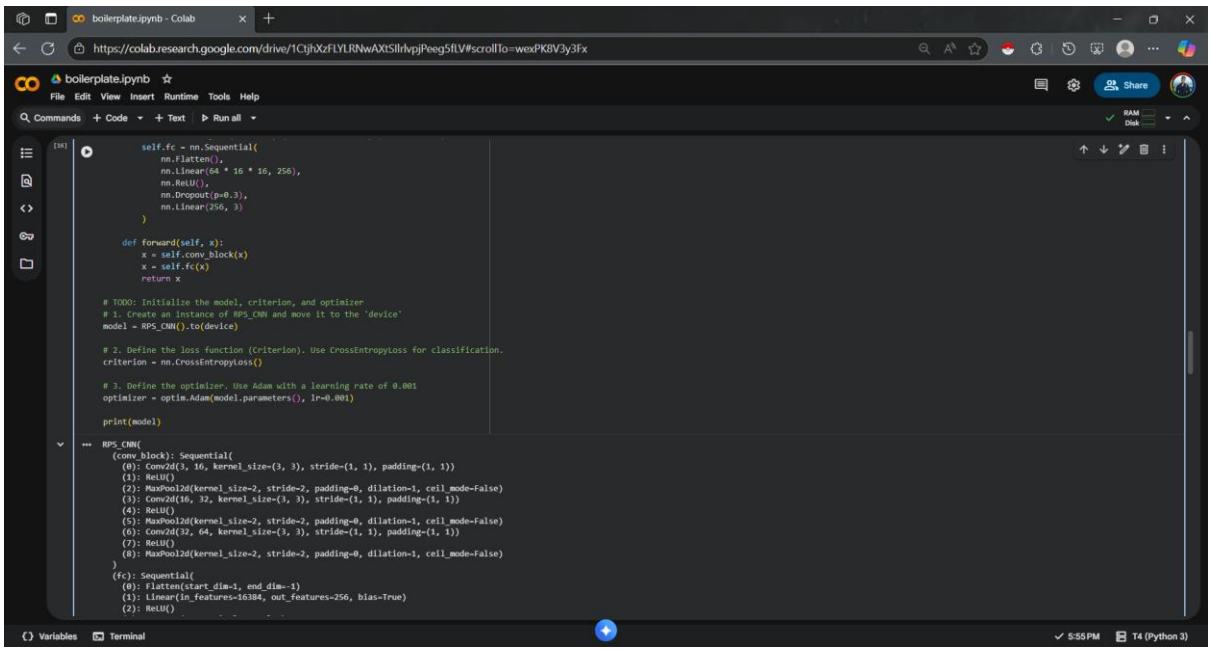


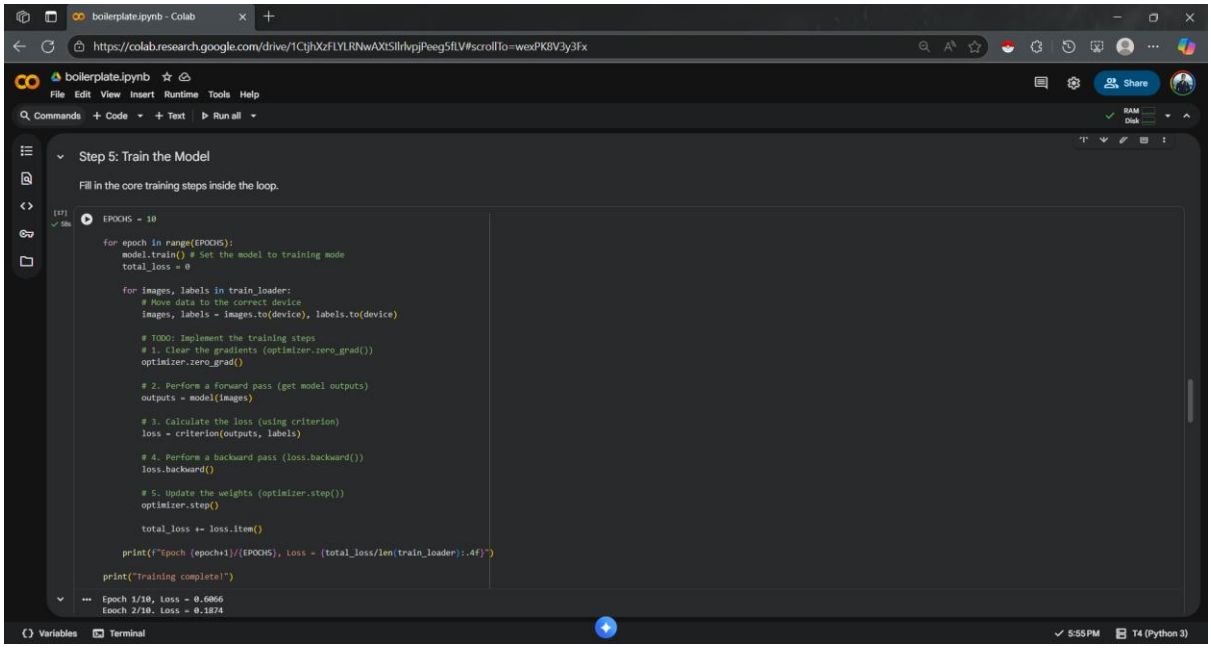


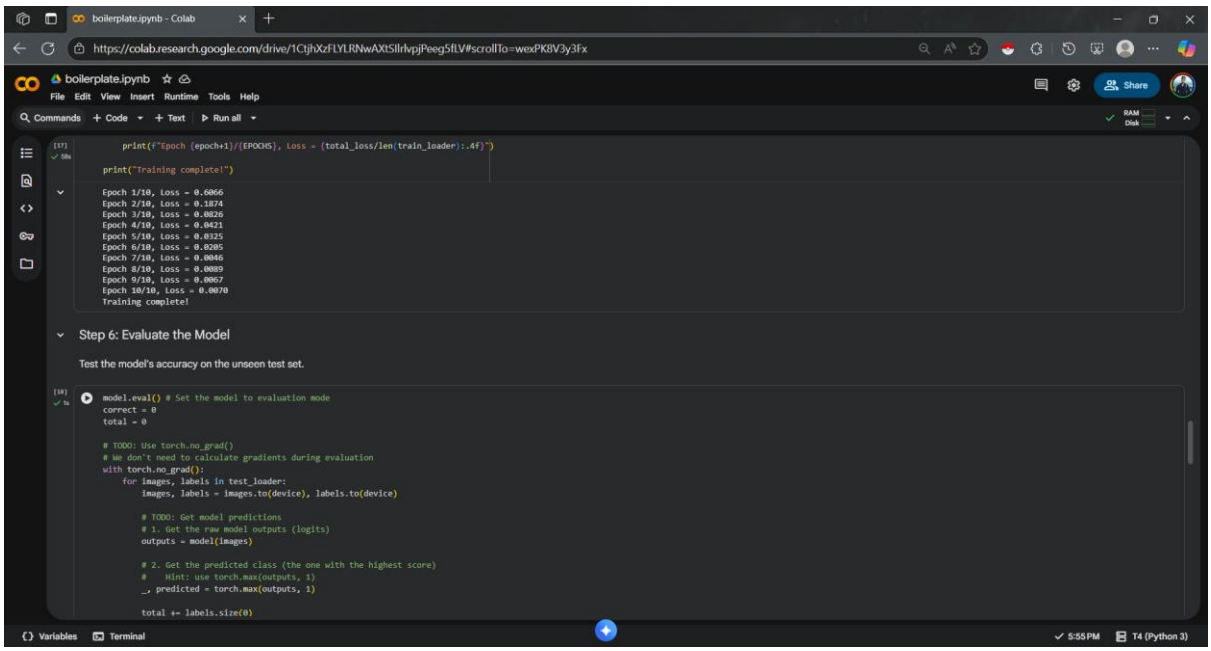

boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=13-RXEbuzuxu
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
Step 2: Imports and Device Setup
Import the necessary libraries and check if a GPU is available.
[34]
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
# TODO: 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
Step 3: Data Loading and Preprocessing
Here we will define our image transformations, load the dataset, split it, and create DataLoaders.
[35]
DATA_DIR = "/content/dataset"
# TODO: Define the image transforms
# We need to:
# 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)),
Variables Terminal
5:55 PM
T4 (Python 3)

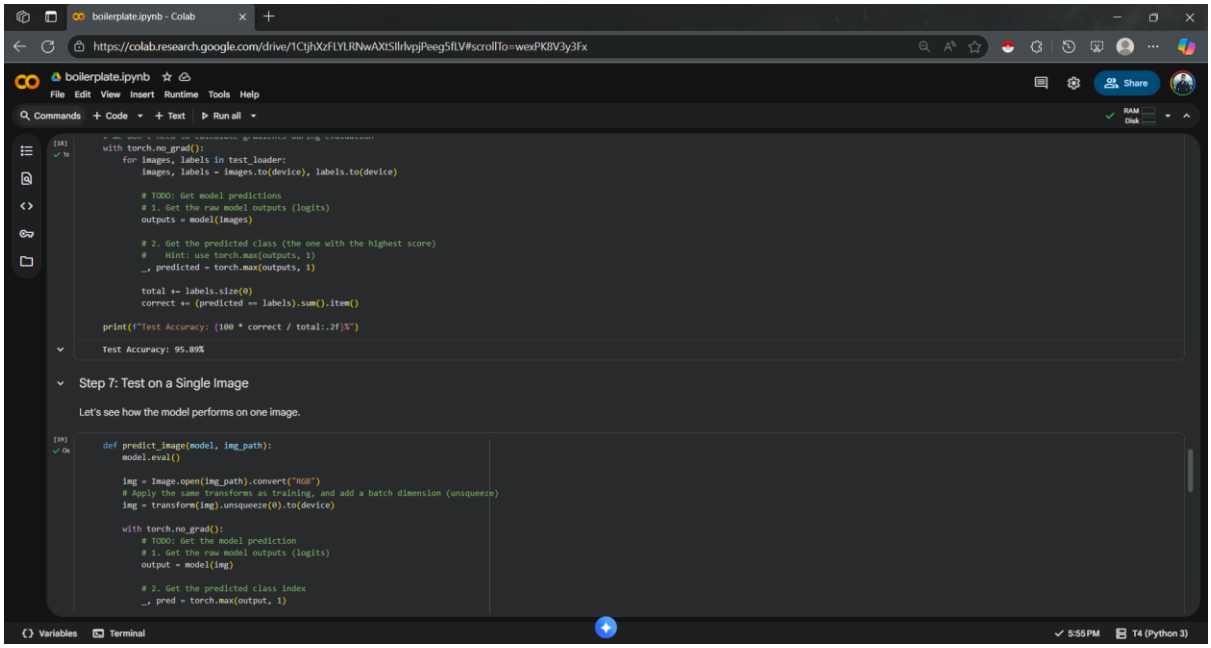

boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=13-RXEbuzuxu
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
[35]
# We need to:
# 1. Resize all images to 128x128
# 2. Convert them to Tensors
# 3. Normalize them (mean=0.5, std=0.5)
transform = transforms.Compose([
transforms.Resize((128, 128)),
transforms.ToTensor(),
transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[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)
# TODO: Split the dataset
# We want 80% for training and 20% for testing
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
# TODO: Use random_split to create train_dataset and test_dataset
train_dataset, test_dataset = random_split(full_dataset, [train_size, test_size])
# TODO: Create the DataLoaders
# Use a batch_size of 32
# Shuffle the training loader, but not the 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
Variables Terminal
5:55 PM
T4 (Python 3)

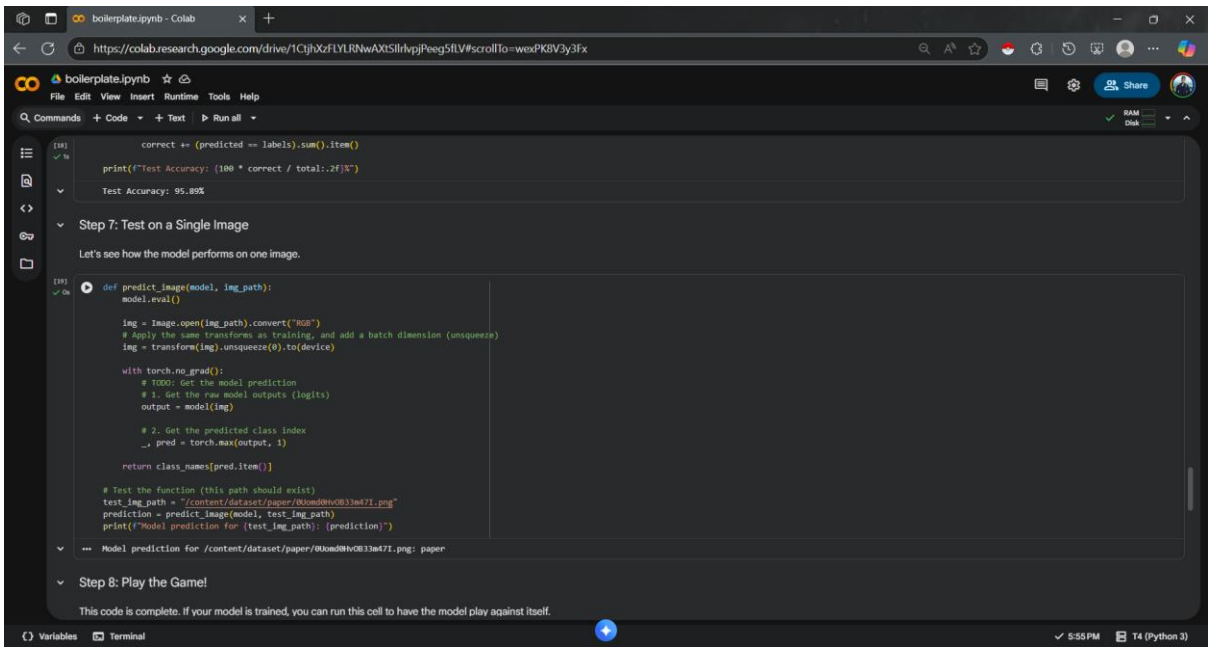

boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
Step 4: Define the CNN Model
Fill in the conv_block and fc_block with the correct layers.
[36]
class RPS_CNN(nn.Module):
def __init__(self):
super(RPS_CNN, self).__init__()
# TODO: Define the convolutional block
# We want 3 blocks:
# 1. Conv2d(3 -> 16 channels, kernel=3, padding=1), ReLU, MaxPool2d(2)
# 2. Conv2d(16 -> 32 channels, kernel=3, padding=1), ReLU, MaxPool2d(2)
# 3. Conv2d(32 -> 64 channels, kernel=3, padding=1), ReLU, MaxPool2d(2)
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 # TODO: Define the fully-connected (classifier) block
# We want:
# 1. Flatten the input
# 2. Linear layer (64 * 16 * 16 -> 256)
# 3. ReLU
# 4. Dropout (p=0.3)
# 5. Linear layer (256 -> 3) (3 classes: rock, paper, scissors)
self.fc = nn.Sequential(
nn.Flatten(),
nn.Linear(64 * 16 * 16, 256),
nn.ReLU(),
Variables Terminal
5:55 PM
T4 (Python 3)

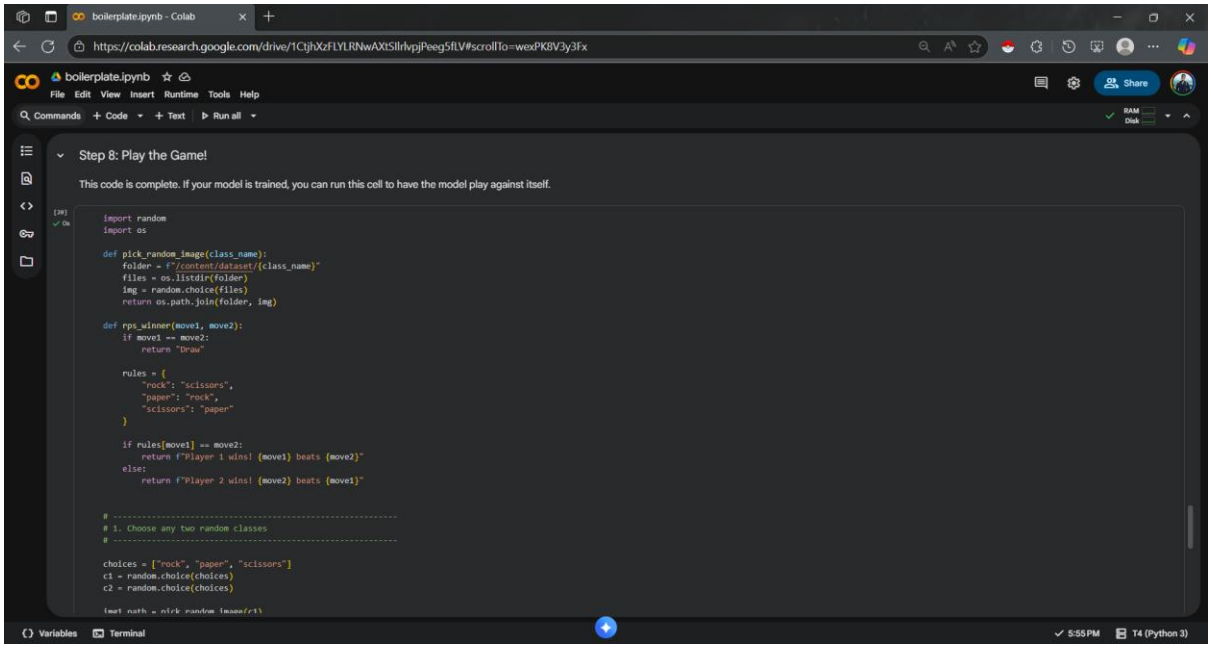

boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
[36]
self.fc = nn.Sequential(
nn.Flatten(),
nn.Linear(64 * 16 * 16, 256),
nn.ReLU(),
nn.Dropout(p=0.3),
nn.Linear(256, 3)
)
def forward(self, x):
x = self.conv_block(x)
x = self.fc(x)
return x
# TODO: 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 classification.
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()
Variables Terminal
5:55 PM
T4 (Python 3)

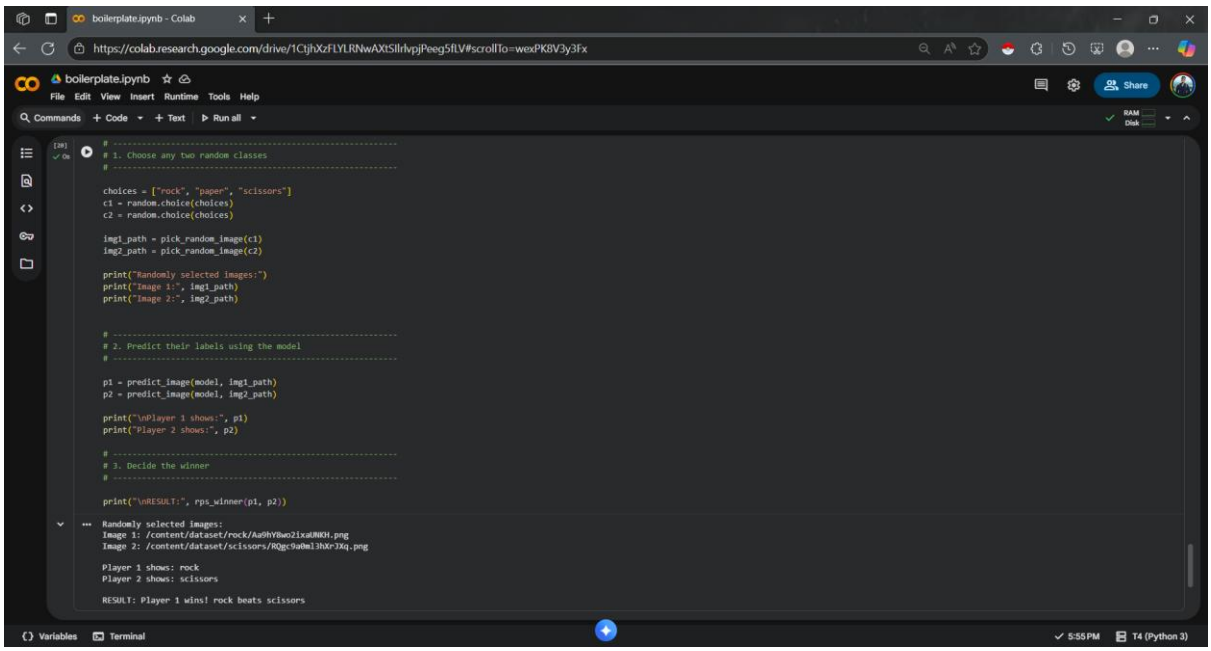

boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
Step 5: Train the Model
Fill in the core training steps inside the loop.
EPOCHS = 10
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)
# TODO: Implement the training steps
# 1. Clear the gradients (optimizer.zero_grad())
optimizer.zero_grad()
# 2. Perform a forward pass (get model outputs)
outputs = model(images)
# 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 = 0.6066
Epoch 2/10, Loss = 0.1874
Variables Terminal
5:55 PM
T4 (Python 3)


boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
print(f"Epoch {epoch+1}/{EPOCHS}, Loss = {total_loss/len(train_loader):.4f}")
print("Training complete!")
Epoch 1/10, Loss = 0.6066
Epoch 2/10, Loss = 0.1874
Epoch 3/10, Loss = 0.0826
Epoch 4/10, Loss = 0.0421
Epoch 5/10, Loss = 0.0325
Epoch 6/10, Loss = 0.0205
Epoch 7/10, Loss = 0.0046
Epoch 8/10, Loss = 0.0089
Epoch 9/10, Loss = 0.0067
Epoch 10/10, Loss = 0.0070
Training complete!
Step 6: Evaluate the Model
Test the model's accuracy on the unseen test set.
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)
Variables Terminal
5:55 PM
T4 (Python 3)


boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
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: 95.89%
Step 7: Test on a Single Image
Let's see how the model performs on one image.
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():
# TODO: Get the model prediction
# 1. Get the raw model outputs (logits)
output = model(img)
# 2. Get the predicted class index
_, pred = torch.max(output, 1)
Variables Terminal
5:55 PM
T4 (Python 3)


boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
correct += (predicted == labels).sum().item()
print(f"Test Accuracy: {100 * correct / total:.2f}%")
Test Accuracy: 95.89%
Step 7: Test on a Single Image
Let's see how the model performs on one image.
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():
# TODO: Get the model prediction
# 1. Get the raw model outputs (logits)
output = model(img)
# 2. 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/0UomdHHvOB33m47I.png"
prediction = predict_image(model, test_img_path)
print(f"Model prediction for {test_img_path}: {prediction}")
Model prediction for /content/dataset/paper/0UomdHHvOB33m47I.png: paper
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.
Variables Terminal
5:55 PM
T4 (Python 3)


boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
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.
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)
Variables Terminal
5:55 PM
T4 (Python 3)


boilerplate.ipynb - Colab
https://colab.research.google.com/drive/1CtjhXzFLYLRNwAXtSIlrlvpjPeeg5fLV#scrollTo=wexPK8V3y3Fx
boilerplate.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
# ---
# 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/Aa0hY8we2ixaUNKH.png
Image 2: /content/dataset/scissors/RQgc9aOml3hXr7Xq.png
Player 1 shows: rock
Player 2 shows: scissors
RESULT: Player 1 wins! rock beats scissors
Variables Terminal
5:55 PM
T4 (Python 3)