**CNN-Based Image Classification Using CIFAR-10 Dataset**

**Project Report**

**Introduction:**

This report outlines the complete approach followed to build an image classification system using a Convolution Neural Network(CNN) by PyTorch. The objective was to classify images from CIFAR-10 dataset into one of ten predefined categories The project was implemented using python in the Pycharm IDE. The final model was trained, evaluated, visualized and saved successfully.

**Project objective:**

To develop CNN nased image classification model using PyTorch that can accurately identify the class of objects in images from the CIFAR-10 dataset, consisting of 10 categories, airplane, automobile, bird, cat, deer, dog,frog,horse, ship,and truck.

**Dataset Description**

Dataset – CIFAR-10

Total Images: 60000 32\*32 color images

Training Set: 50000 images

Test Set: 10,000 images

Categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck

The Dataset was downloaded directly using torchvision.datasets.CIFAR10()

**Tools & Libraries Used:**

Programming language: Python 3.x

IDE: PyCharm

Libraries: PyTorch, torchvision, matplotlob, numpy

**Step By Step Approach**

**Step1: Data Loading and Preprocessing**

Used torchvision.transforms to convert images to tensors and normalize ixel values

Loaded the training test data using DataLoader to create iterable datasets

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])

**STEP2: Define CNN Architecture**

Created a class SimpleCNN using nn.Module

The architecture includes 3 convolutional layers, max pooling, ReLU activators, and 2 fully connected layers.

self.conv1 = nn.Conv2d(3, 32, 3, padding=1)

self.conv2 = nn.Conv2d(32, 64, 3, padding=1)

self.conv3 = nn.Conv2d(64, 64, 3, padding=1)

self.fc1 = nn.Linear(64 \* 4 \* 4, 64)

self.fc2 = nn.Linear(64, 10)

**STEP3: Define loss function and Optimizer**

Used CrossEntropyLoss for multi-class classification.

Used SGD optimizer with learning rate of 0.001 and momentum of 0.9.

**STEP 4: Training the Model**

Trained the model for 10 epoch

For each batch, calculated loss,performed backpropagation, and updated model weights.

Printed running loss for every 100 mini batches.

**STEP 5: Model Evaluation**

Evaluated the model on the test set

Calculated the accuracy using correct predictions vs total images.

**STEP 6: Visualizing Predictions**

Displayed 8 random test images using matplotlib.

Showed both actual and predicted labels.

**STEP 7: Saving the Model**

Saved the trained model to a file named cnn\_model.pth

**Results**

Final Test accuracy achieved: 72.5% (depending on training output)

Screenshots of the output results were captured to demonstrate predictions.

**Challenges Faced**

**Challenge**: Low initial accuracy.

**Solution:** Added a third convolutional layer and normalized the images.

**Challenge**: Overfitting

**Solution:** shuffled data and considered adding dropout

**Future Improvements**

Add dropout to improve generalization

Apply data augmentation techniques

Explore pretrained models using transfer learning.

Use learning rate scheduling and train for more epoch.

**Conclusion**

This Project helped deepen understanding of convolutional neural network, data preprocessing, and model evaluation in PyTorch.The ability to visualize and interpret predictions provides insight into the model’s learning process.

Further improvements can be made by extending the architecture or using advanced deep learning techniques.

**Output and Screenshots**

/Users/devashreepk/Library/Caches/JetBrains/PyCharm2024.3/demo/PyCharmLearningProject/.venv/bin/python /Users/devashreepk/PycharmProjects/Machine-learning/Image Classification CNN with Pytorch/ImageclassificationCNN.py

['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

[1, 100] loss: 2.302

[1, 200] loss: 2.301

[1, 300] loss: 2.296

[1, 400] loss: 2.291

[1, 500] loss: 2.284

[1, 600] loss: 2.262

[1, 700] loss: 2.225

[1, 800] loss: 2.154

[1, 900] loss: 2.097

[1, 1000] loss: 2.053

[1, 1100] loss: 2.004

[1, 1200] loss: 1.982

[1, 1300] loss: 1.965

[1, 1400] loss: 1.940

[1, 1500] loss: 1.892

[2, 100] loss: 1.882

[2, 200] loss: 1.839

[2, 300] loss: 1.790

[2, 400] loss: 1.767

[2, 500] loss: 1.721

[2, 600] loss: 1.722

[2, 700] loss: 1.676

[2, 800] loss: 1.671

[2, 900] loss: 1.643

[2, 1000] loss: 1.620

[2, 1100] loss: 1.534

[2, 1200] loss: 1.587

[2, 1300] loss: 1.568

[2, 1400] loss: 1.569

[2, 1500] loss: 1.538

[3, 100] loss: 1.493

[3, 200] loss: 1.539

[3, 300] loss: 1.524

[3, 400] loss: 1.479

[3, 500] loss: 1.484

[3, 600] loss: 1.461

[3, 700] loss: 1.476

[3, 800] loss: 1.434

[3, 900] loss: 1.442

[3, 1000] loss: 1.424

[3, 1100] loss: 1.426

[3, 1200] loss: 1.420

[3, 1300] loss: 1.448

[3, 1400] loss: 1.392

[3, 1500] loss: 1.378

[4, 100] loss: 1.397

[4, 200] loss: 1.359

[4, 300] loss: 1.385

[4, 400] loss: 1.339

[4, 500] loss: 1.325

[4, 600] loss: 1.315

[4, 700] loss: 1.330

[4, 800] loss: 1.319

[4, 900] loss: 1.300

[4, 1000] loss: 1.329

[4, 1100] loss: 1.339

[4, 1200] loss: 1.293

[4, 1300] loss: 1.291

[4, 1400] loss: 1.276

[4, 1500] loss: 1.295

[5, 100] loss: 1.276

[5, 200] loss: 1.264

[5, 300] loss: 1.238

[5, 400] loss: 1.280

[5, 500] loss: 1.225

[5, 600] loss: 1.219

[5, 700] loss: 1.202

[5, 800] loss: 1.232

[5, 900] loss: 1.235

[5, 1000] loss: 1.194

[5, 1100] loss: 1.177

[5, 1200] loss: 1.206

[5, 1300] loss: 1.204

[5, 1400] loss: 1.180

[5, 1500] loss: 1.176

[6, 100] loss: 1.122

[6, 200] loss: 1.146

[6, 300] loss: 1.147

[6, 400] loss: 1.147

[6, 500] loss: 1.125

[6, 600] loss: 1.103

[6, 700] loss: 1.132

[6, 800] loss: 1.121

[6, 900] loss: 1.114

[6, 1000] loss: 1.119

[6, 1100] loss: 1.132

[6, 1200] loss: 1.101

[6, 1300] loss: 1.121

[6, 1400] loss: 1.118

[6, 1500] loss: 1.066

[7, 100] loss: 1.084

[7, 200] loss: 1.080

[7, 300] loss: 1.059

[7, 400] loss: 1.031

[7, 500] loss: 1.061

[7, 600] loss: 1.022

[7, 700] loss: 1.055

[7, 800] loss: 1.036

[7, 900] loss: 1.040

[7, 1000] loss: 1.025

[7, 1100] loss: 1.019

[7, 1200] loss: 1.030

[7, 1300] loss: 1.032

[7, 1400] loss: 0.994

[7, 1500] loss: 1.006

[8, 100] loss: 0.973

[8, 200] loss: 0.993

[8, 300] loss: 0.975

[8, 400] loss: 0.974

[8, 500] loss: 0.996

[8, 600] loss: 0.955

[8, 700] loss: 0.982

[8, 800] loss: 0.969

[8, 900] loss: 0.929

[8, 1000] loss: 0.945

[8, 1100] loss: 0.947

[8, 1200] loss: 0.946

[8, 1300] loss: 0.941

[8, 1400] loss: 0.971

[8, 1500] loss: 0.962

[9, 100] loss: 0.904

[9, 200] loss: 0.893

[9, 300] loss: 0.900

[9, 400] loss: 0.909

[9, 500] loss: 0.913

[9, 600] loss: 0.884

[9, 700] loss: 0.915

[9, 800] loss: 0.912

[9, 900] loss: 0.909

[9, 1000] loss: 0.874

[9, 1100] loss: 0.884

[9, 1200] loss: 0.931

[9, 1300] loss: 0.918

[9, 1400] loss: 0.915

[9, 1500] loss: 0.889

[10, 100] loss: 0.858

[10, 200] loss: 0.843

[10, 300] loss: 0.843

[10, 400] loss: 0.853

[10, 500] loss: 0.850

[10, 600] loss: 0.855

[10, 700] loss: 0.857

[10, 800] loss: 0.837

[10, 900] loss: 0.823

[10, 1000] loss: 0.887

[10, 1100] loss: 0.858

[10, 1200] loss: 0.797

[10, 1300] loss: 0.864

[10, 1400] loss: 0.846

[10, 1500] loss: 0.844

Finished Training

Accuracy: 68.70%

Model saved successfully

Process finished with exit code 0





