| **Practical Evaluation Task**  To study and implement a Convolutional Neural Network (CNN) for image classification on the CIFAR-100 dataset with real world examples. |
| --- |

| **Problem Description:** The objective is to design and implement a Convolutional Neural Network (CNN) for image classification using the CIFAR-100 dataset. After loading and normalizing the dataset, data augmentation techniques like rotations, shifts and flips are applied to enhance diversity, which helps the model generalize better. During training, the model’s accuracy is monitored and expected to improve with each epoch. By the end, the model should demonstrate robust accuracy when tested on unseen images, showing its effectiveness in real-world image classification tasks. |
| --- |

| **Study of CIFAR-100 Dataset:** The CIFAR-100 dataset consists of 60,000 color images, each 32x32 pixels in size, distributed across 100 fine-grained categories, with 600 images per category. It is divided into 50,000 training images and 10,000 test images. Each image is labeled with one of 100 specific classes, such as "cloud", "dolphin", "lizard", "mountain" or "pine\_tree". This dataset is commonly used for training and evaluating machine learning models, particularly in the field of image classification. Compared to the CIFAR-10 dataset, CIFAR-100 presents a more difficult challenge due to the larger number of categories. |
| --- |

| **Solution Architecture:** |
| --- |

| **Code:**  import torch  import torch.nn as nn  import torch.optim as optim  import torchvision  import torchvision.transforms as transforms  from torchvision import models  import numpy as np  import matplotlib.pyplot as plt  import torch.nn.functional as F  from sklearn.metrics import accuracy\_score  from torchvision import transforms  from torch.utils.data import DataLoader, Dataset  from PIL import Image  import pandas as pd  import cv2  # Check if GPU is available  device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  # Data preparation  transform = transforms.Compose([  transforms.Resize(256),  transforms.CenterCrop(224), # Resize to match ResNet input size  transforms.ToTensor(),  transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),  ])  # Load CIFAR-100 dataset  trainset = torchvision.datasets.CIFAR100(root='./data', train=True, download=True, transform=transform)  trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True, num\_workers=2)  testset = torchvision.datasets.CIFAR100(root='./data', train=False, download=True, transform=transform)  testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=False, num\_workers=2)  # Load pretrained ResNet model  model = models.resnet18(pretrained=True)  # Modify the final layer to match CIFAR-100 (100 classes)  model.fc = nn.Linear(model.fc.in\_features, 100)  model = model.to(device)  # Define loss function and optimizer  criterion = nn.CrossEntropyLoss()  optimizer = optim.Adam(model.parameters(), lr=0.001)  # Training loop  num\_epochs = 10  train\_losses = []  train\_accuracies = []  val\_losses = []  val\_accuracies = []  def evaluate(model, data\_loader):  model.eval() # Set the model to evaluation mode  correct = 0  total = 0  running\_loss = 0.0  criterion = nn.CrossEntropyLoss()  with torch.no\_grad():  for inputs, labels in data\_loader:  inputs, labels = inputs.to(device), labels.to(device)  outputs = model(inputs)  loss = criterion(outputs, labels)  running\_loss += loss.item()  \_, predicted = torch.max(outputs.data, 1)  total += labels.size(0)  correct += (predicted == labels).sum().item()  accuracy = 100 \* correct / total  loss = running\_loss / len(data\_loader)  return loss, accuracy  for epoch in range(num\_epochs):  model.train()  running\_loss = 0.0  correct = 0  total = 0  for inputs, labels in trainloader:  inputs, labels = inputs.to(device), labels.to(device)  optimizer.zero\_grad()  outputs = model(inputs)  loss = criterion(outputs, labels)  loss.backward()  optimizer.step()  running\_loss += loss.item()  epoch\_loss, epoch\_acc = evaluate(model, trainloader)  train\_losses.append(epoch\_loss)  train\_accuracies.append(epoch\_acc)  print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {running\_loss/len(trainloader):.4f}, Train Accuracy: {epoch\_acc:.2f}%')  # Optional: Evaluate on validation set if available (here using training loader as proxy)  val\_loss, val\_acc = evaluate(model, testloader)  val\_losses.append(val\_loss)  val\_accuracies.append(val\_acc)  print(f'Epoch [{epoch+1}/{num\_epochs}], Val Loss: {val\_loss:.4f}, Val Accuracy: {val\_acc:.2f}%')  # Plotting Training and Validation History (Loss and Accuracy)  plt.figure(figsize=(10,5))  plt.suptitle('Accuracy Plots', fontsize=18)  # Plot accuracy  plt.subplot(1,2,2)  plt.plot(train\_accuracies, label='Train Accuracy')  plt.plot(val\_accuracies, label='Validation Accuracy')  plt.legend()  plt.xlabel('Number of epochs', fontsize=14)  plt.ylabel('Accuracy', fontsize=14)  plt.show()  # Evaluation on Test Dataset  test\_loss, test\_acc = evaluate(model, testloader)  print(f"Test Accuracy: {test\_acc:.2f}%")  # Function to Resize Test Image for Prediction  def resize\_test\_image(test\_img):  img = cv2.imread(test\_img)  img\_RGB = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Convert from BGR to RGB  resized\_img = cv2.resize(img\_RGB, (224, 224)) # Resize to the input size of the model  resized\_img = resized\_img / 255.0 # Normalize the image  resized\_img = np.transpose(resized\_img, (2, 0, 1)) # Convert to CHW format for PyTorch  resized\_img = torch.tensor(resized\_img).float() # Convert to a PyTorch tensor  resized\_img = resized\_img.unsqueeze(0) # Add batch dimension (1, C, H, W)  return resized\_img  # Predicting on a Test Image  def predict\_test\_image(test\_img):  resized\_img = resize\_test\_image(test\_img)  resized\_img = resized\_img.to(device) # Move image to the same device as model  model.eval()  with torch.no\_grad():  prediction = model(resized\_img)  return prediction  # Getting Predictions  def sort\_prediction\_test\_image(test\_img):  prediction = predict\_test\_image(test\_img)  probs = F.softmax(prediction, dim=1) # Convert logits to probabilities  top\_probs, top\_indices = torch.topk(probs, 5) # Get predictions  top\_probs = top\_probs.squeeze().cpu().numpy() # Remove batch dimension and move to CPU  top\_indices = top\_indices.squeeze().cpu().numpy()  return top\_indices, top\_probs  # Get DataFrame for Predictions  def df\_prediction\_test\_image(test\_img):  top\_indices, top\_probs = sort\_prediction\_test\_image(test\_img)  fine\_labels = [  "apple", "aquarium\_fish", "baby", "bear", "beaver", "bed", "bee", "beetle", "bicycle", "bottle",  "bowl", "boy", "bridge", "bus", "butterfly", "camel", "can", "castle", "caterpillar", "cattle",  "chair", "chimpanzee", "clock", "cloud", "cockroach", "couch", "crab", "crocodile", "cup", "dinosaur",  "dolphin", "elephant", "flatfish", "forest", "fox", "girl", "hamster", "house", "kangaroo", "keyboard",  "lamp", "lawn\_mower", "leopard", "lion", "lizard", "lobster", "man", "maple\_tree", "motorcycle", "mountain",  "mouse", "mushroom", "oak\_tree", "orange", "orchid", "otter", "palm\_tree", "pear", "pickup\_truck", "pine\_tree",  "plain", "plate", "poppy", "porcupine", "possum", "rabbit", "raccoon", "ray", "road", "rocket", "rose",  "sea", "seal", "shark", "shrew", "skunk", "skyscraper", "snail", "snake", "spider", "squirrel", "streetcar",  "sunflower", "sweet\_pepper", "table", "tank", "telephone", "television", "tiger", "tractor", "train", "trout",  "tulip", "turtle", "wardrobe", "whale", "willow\_tree", "wolf", "woman", "worm"  ]  class\_name = [fine\_labels[idx] for idx in top\_indices]  df = pd.DataFrame(list(zip(class\_name, top\_probs)), columns=['Label', 'Probability'])  return df  # Function to display image and plot probability side by side  def plot\_prediction\_test\_image(test\_img):  # Display the image  img = Image.open(test\_img)  # Get predictions  df = df\_prediction\_test\_image(test\_img)  # Create a figure with two subplots: one for the image, one for the plot  fig, ax = plt.subplots(1, 2, figsize=(15, 6))  # Plot the image in the first subplot  ax[0].imshow(img)  ax[0].axis('off')  ax[0].set\_title(f"Uploaded Image: {test\_img.split('/')[-1]}")  # Plot the probabilities in the second subplot  ax[1].bar(df['Label'], df['Probability'], color='skyblue')  ax[1].set\_xlabel('Class Label')  ax[1].set\_ylabel('Probability')  ax[1].set\_title('Predictions with Probabilities')  ax[1].tick\_params(axis='x', rotation=45)  ax[1].set\_xticks(df['Label'])  ax[1].set\_xticklabels(df['Label'], rotation=45, ha='right')  # Adjust layout for better spacing  plt.tight\_layout()  plt.show()  plot\_prediction\_test\_image('/content/orange.jpeg')  plot\_prediction\_test\_image('/content/tulip.jpeg')  plot\_prediction\_test\_image('/content/can.jpg')  plot\_prediction\_test\_image('/content/house.jpeg')  plot\_prediction\_test\_image('/content/bee.wbep')  plot\_prediction\_test\_image('/content/worm.jpeg') |
| --- |

| **Results:**  **→ Plotted Accuracy**    **→ Test Accuracy : 73.94 %**  **→ Predictions :** |
| --- |

| **Summary :**  This code applies Convolutional Neural Networks (CNNs) on the CIFAR-100 dataset by fine-tuning a pretrained ResNet-18 model. The dataset consists of 60,000 32x32 color images across 100 classes. The images are preprocessed by resizing to 224x224 pixels, center cropping, and normalizing to match the input size and distribution expected by ResNet-18. The final fully connected layer of the pretrained model is replaced to output predictions for 100 CIFAR-100 classes.  The model is trained for 10 epochs using the Adam optimizer and cross-entropy loss. During training, both training and validation accuracy are tracked, with evaluations performed on the test set at each epoch. After training, the model’s test accuracy is printed.  Additionally, the code includes a mechanism for predicting custom test images. The images are resized and normalized before being passed through the model. The predicted classes and their probabilities are displayed alongside the image using matplotlib, providing clear visual feedback on the model's predictions. This approach leverages CNNs and transfer learning for effective image classification on CIFAR-100. |
| --- |