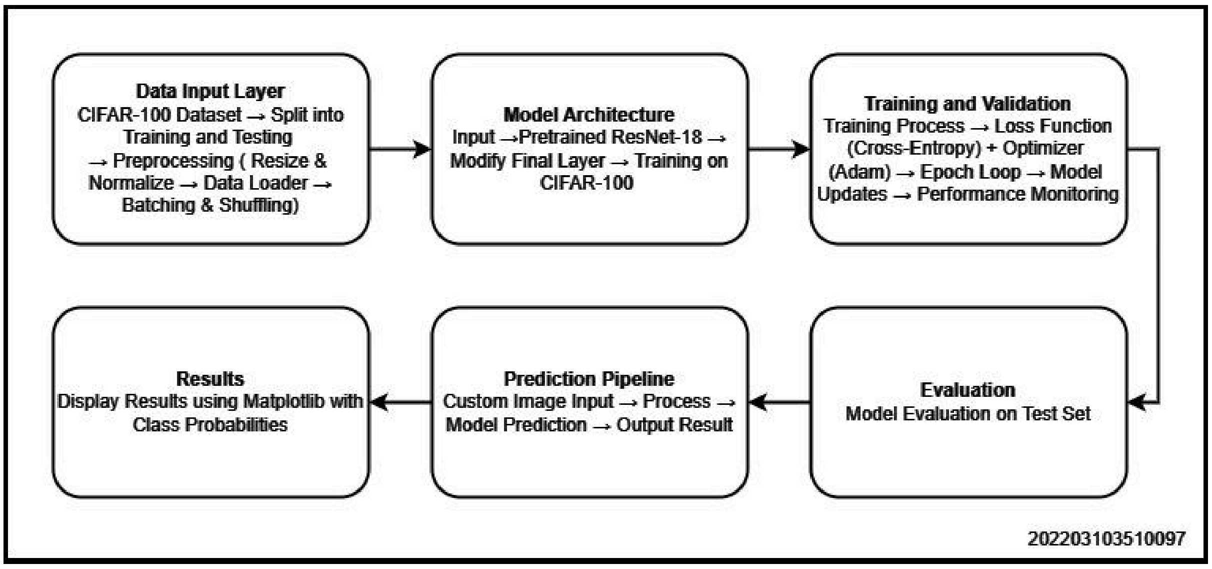
Practical Evaluation Task

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:


Data Input Layer
CIFAR-100 Dataset → Split into Training and Testing → Preprocessing (Resize & Normalize → Data Loader → Batching & Shuffling)
Model Architecture
Input →Pretrained ResNet-18 → Modify Final Layer → Training on CIFAR-100
Training and Validation
Training Process → Loss Function (Cross-Entropy) + Optimizer (Adam) → Epoch Loop → Model Updates → Performance Monitoring
Evaluation
Model Evaluation on Test Set
Prediction Pipeline
Custom Image Input → Process → Model Prediction → Output Result
Results
Display Results using Matplotlib with Class Probabilities
202203103510097

Code:

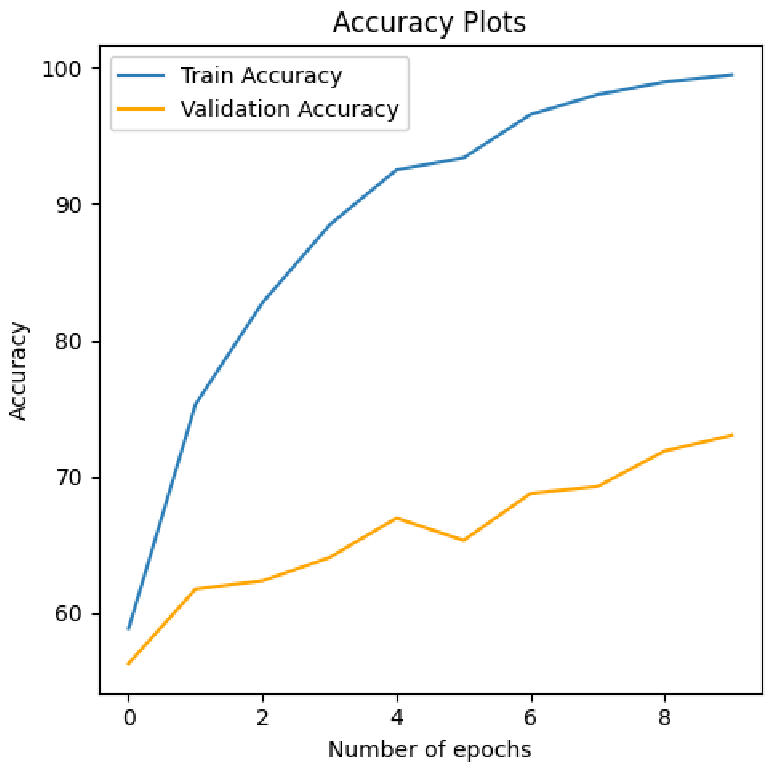
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models
import numpy as np
import matplotlib.pyplot as plt
import torch.nn.functional as F
from sklearn.metrics import accuracy_score
from torchvision 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
```

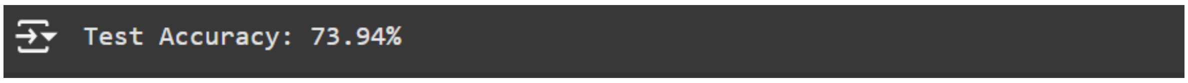
```
# Getting Predictions
def sort_prediction_test_image(test_img):
    prediction = predict_test_image(test_img)
    probs = F.softmax(prediction, dim=1)  # Convert logits to probabilities
    top_probs, top_indices = torch.topk(probs, 5)  # Get predictions
    top_probs = top_probs.squeeze().cpu().numpy()  # 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

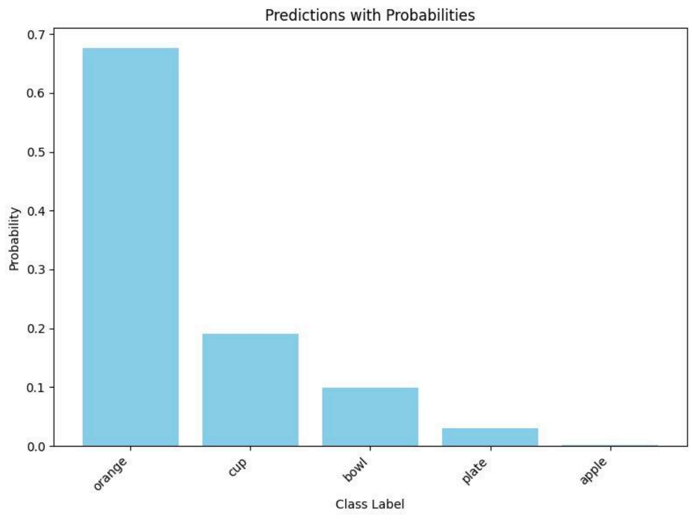


→ Test Accuracy : 73.94 %

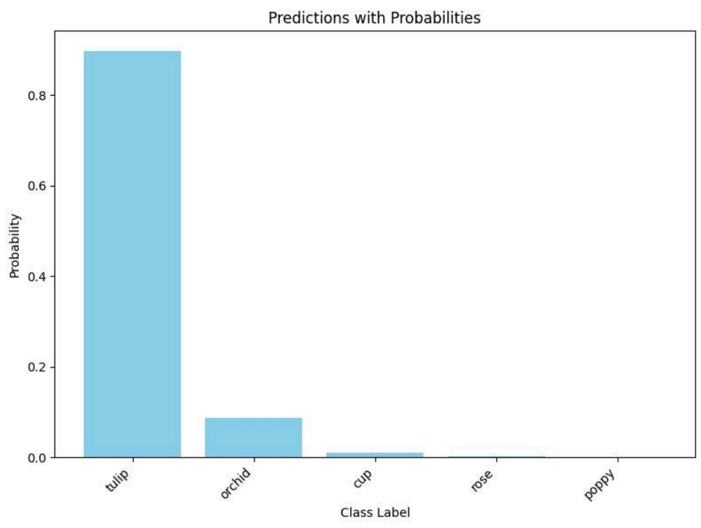


```
Test Accuracy: 73.94%
```

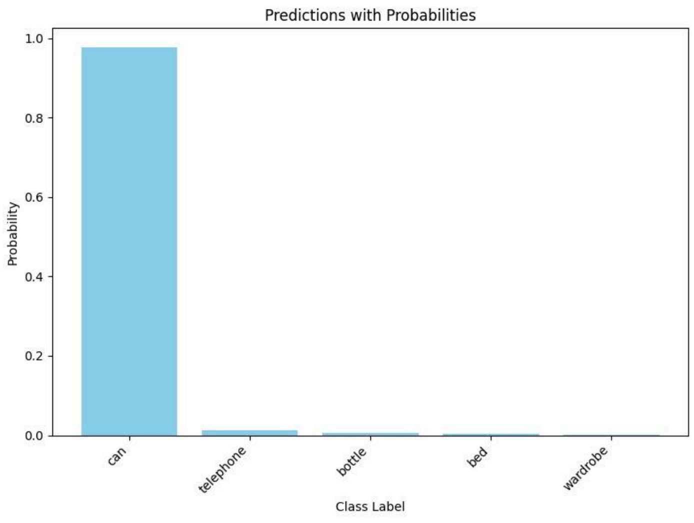
→ Predictions :

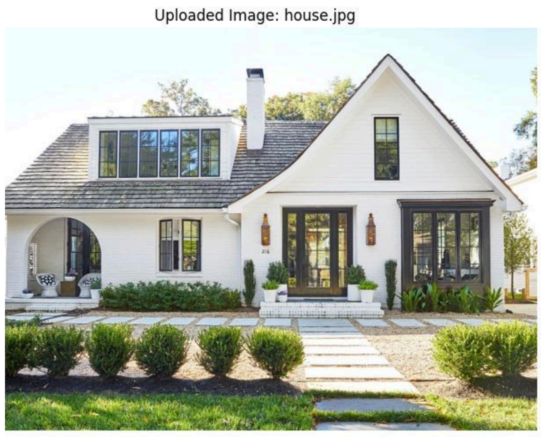


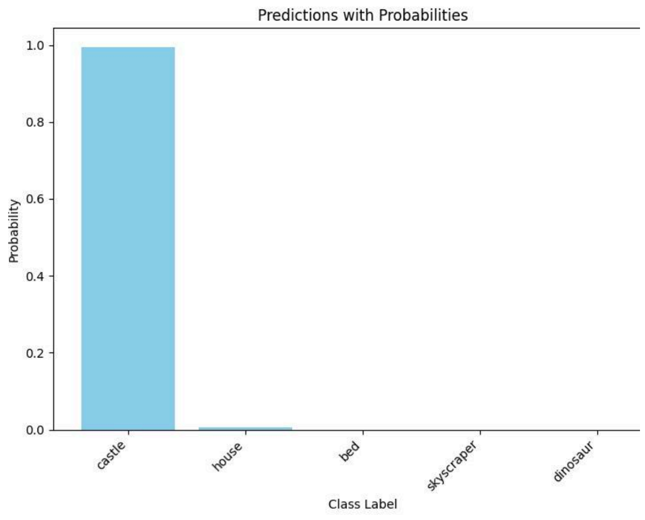
Uploaded Image: tulip.jpeg

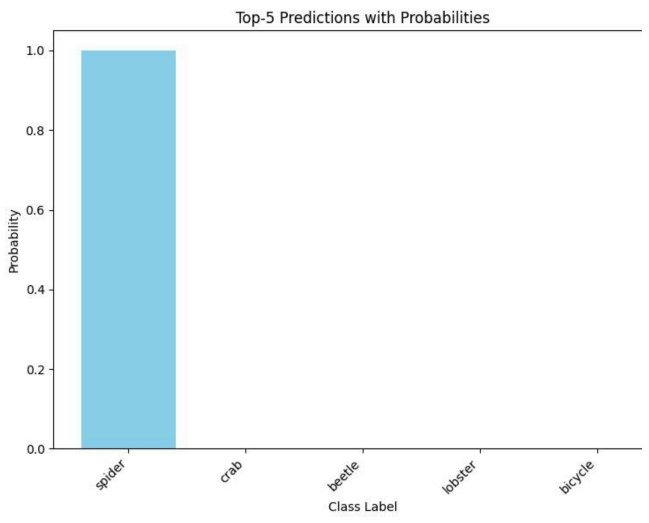

Predictions with Probabilities
Probability
0.0
0.2
0.4
0.6
0.8
tulip
orchid
cup
rose
poppy
Class Label

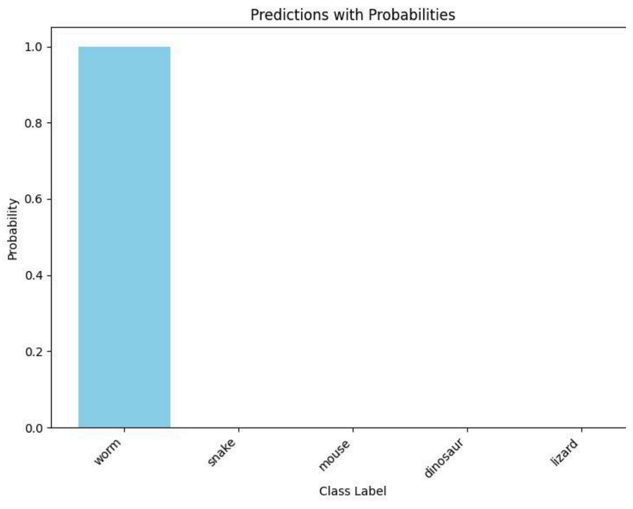
Uploaded Image: can.jpg
LIGHT TASTE NO SUGAR
Coca-Cola
CRISP TASTE


Predictions with Probabilities
Probability
0.0
0.2
0.4
0.6
0.8
1.0
can
telephone
bottle
bed
wardrobe
Class Label


Uploaded Image: house.jpg


Predictions with Probabilities
Probability
0.0
0.2
0.4
0.6
0.8
1.0
castle
house
bed
skyscraper
dinosaur
Class Label





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.