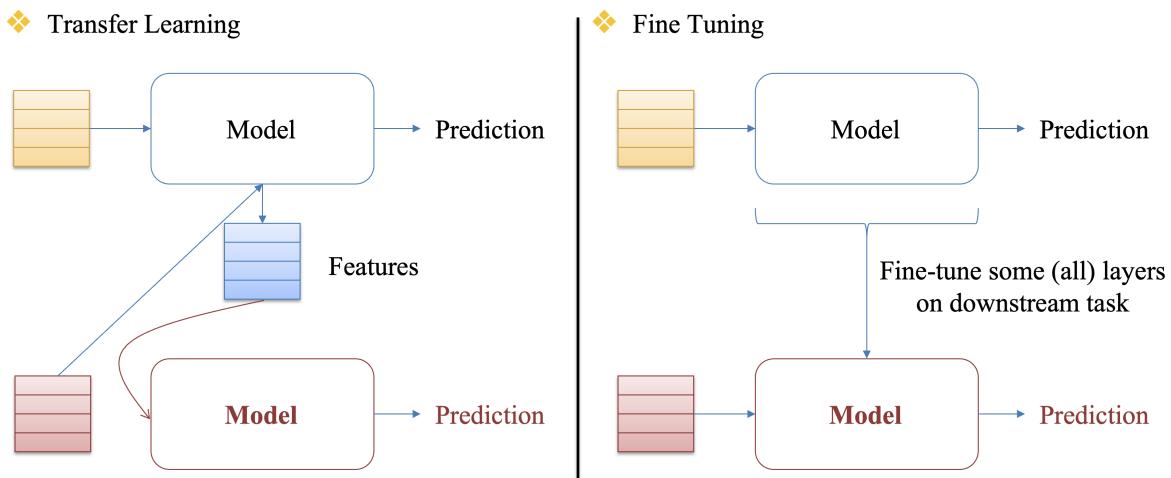


Exercise: Pretrained Models for Image

Ngày 20 tháng 11 năm 2023

Phần 1. Pretrained Models



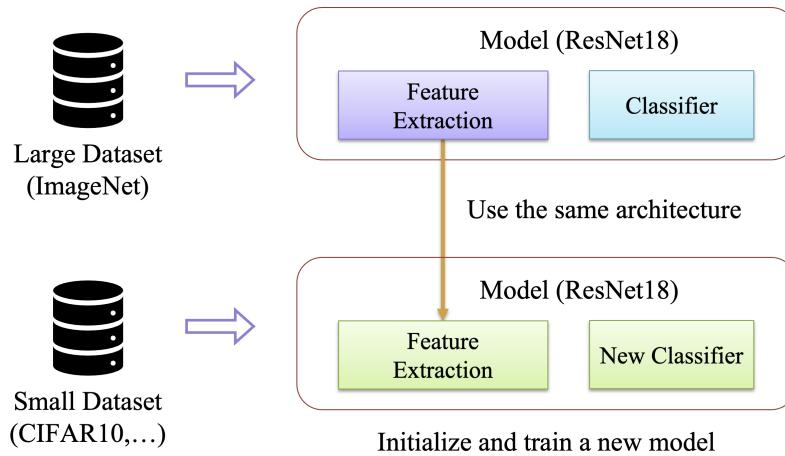
Hình 1: Kỹ thuật Transfer Learning và Fine Tuning sử dụng Pretrained Model.

Với sự phát triển nhanh chóng của dữ liệu và các kiến trúc mô hình áp dụng cho các loại dữ liệu khác nhau ngày càng lớn. Diễn hình những mô hình lớn đầu tiên thành công cho bài toán phân loại như ResNet, VGG,... Bên cạnh đó là các bộ dữ liệu như ImageNet với hơn 1 triệu hình ảnh được gán nhãn cho 1,000 nhãn lớp. Các bộ dữ liệu lớn sẽ giúp mô hình lớn học được nhiều thông tin (đặc trưng) hơn, tuy nhiên cần nhiều tài nguyên tính toán. Vì vậy, tận dụng những mô hình đã được huấn luyện trước trên những bộ dữ liệu lớn như ImageNet để huấn luyện với bộ dữ liệu nhỏ hơn gọi là kỹ thuật học chuyển giao (transfer learning) và tinh chỉnh mô hình (fine tuning).

Trong phần này chúng ta sẽ tìm hiểu các kỹ thuật transfer learning, fine tuning và so sánh với việc huấn luyện các mô hình từ đầu trên bộ dữ liệu Flower cho bài toán phân loại hình ảnh. Bộ dữ liệu Flower với hơn 3 000 ảnh; thuộc 5 nhãn khác nhau. Huấn luyện mô hình ResNet18 với các kỹ thuật transfer learning và fine tuning, Phân tích so sánh ưu nhược điểm của các phương pháp để chọn kỹ thuật phù hợp với những trường hợp khác nhau với mục đích tăng hiệu quả của mô hình.

Nội dung phần tiếp theo gồm: - Huấn luyện mô hình ResNet18 từ đầu - Transfer Learning với mô hình ResNet18 được huấn luyện trước trên bộ dữ liệu ImageNet - Fine Tuning với mô hình ResNet18 được huấn luyện trước trên bộ dữ liệu ImageNet

2.1. Train from scratch



Hình 2: Huấn luyện mô hình ResNet18 từ đầu trên bộ dữ liệu Flower.

Ở phần này, chúng ta sử dụng kiến trúc mô hình với các trọng số khởi tạo để huấn luyện với bộ dữ liệu Flower

1. Tải về bộ dữ liệu

```

1 # import libs
2 import os
3 import time
4 import random
5 import numpy as np
6
7 import torch
8 import torch.nn as nn
9 import torch.optim as optim
10 import torch.nn.functional as F
11 import torch.utils.data as data
12
13 from torchvision import datasets, models, transforms
14
15 from torchsummary import summary
16
17 import matplotlib.pyplot as plt
18 from PIL import Image
19
20 # download dataset
21 !gdown 11Buzytn4vIh4x_0qz8MY29JMMdIqSzj -
22 !unzip ./flower_photos.zip
23
24 # load dataset from path
25 data_path = "./flower_photos"
26 dataset = datasets.ImageFolder(root= data_path)
27
28 num_samples = len(dataset)
29 num_classes = len(dataset.classes)
30 num_samples, num_classes

```

2. Tiền xử lý dữ liệu Gồm 2 bước: - Chia tập: training - validation - testing với tỉ lệ 0.8 : 0.1 : 0.1 - Tăng cường với tập training

```
1 TRAIN_RATIO, VALID_RATIO = 0.8, 0.1
2
3 n_train_examples = int(num_samples * TRAIN_RATIO)
4 n_valid_examples = int(num_samples * VALID_RATIO)
5 n_test_examples = num_samples - n_train_examples - n_valid_examples
6
7 train_dataset, valid_dataset, test_dataset = data.random_split(
8     dataset,
9     [n_train_examples, n_valid_examples, n_test_examples]
10)
11
12 # resize + convert to tensor
13 IMG_SIZE = 224
14
15 train_transforms = transforms.Compose([
16     transforms.Resize((IMG_SIZE, IMG_SIZE)),
17     transforms.RandomHorizontalFlip(),
18     transforms.RandomRotation(0.2),
19     transforms.ToTensor(),
20     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
21])
22
23 test_transforms = transforms.Compose([
24     transforms.Resize((IMG_SIZE, IMG_SIZE)),
25     transforms.ToTensor(),
26     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
27])
28 train_dataset.dataset.transform = train_transforms
29 valid_dataset.dataset.transform = test_transforms
30 test_dataset.dataset.transform = test_transforms
31
32 # build dataloader
33 BATCH_SIZE = 256
34
35 train_dataloader = data.DataLoader(
36     train_dataset,
37     shuffle=True,
38     batch_size=BATCH_SIZE
39)
40
41 valid_dataloader = data.DataLoader(
42     valid_dataset,
43     batch_size=BATCH_SIZE
44)
45
46 test_dataloader = data.DataLoader(
47     test_dataset,
48     batch_size=BATCH_SIZE
49)
```

3. Định nghĩa hàm train

```
1 # train epoch
2 def train_epoch(model, optimizer, criterion, train_dataloader, device, epoch=0,
3     log_interval=5):
4     model.train()
5     accs, losses = [], []
6     start_time = time.time()
7
8     for idx, (inputs, labels) in enumerate(train_dataloader):
9         inputs = inputs.to(device)
```

```
9     labels = labels.to(device)
10
11     optimizer.zero_grad()
12
13     predictions = model(inputs)
14
15     # compute loss
16     loss = criterion(predictions, labels)
17     losses.append(loss.item())
18
19     # backward
20     loss.backward()
21     optimizer.step()
22
23     total_acc = (predictions.argmax(1) == labels).sum().item()
24     acc = total_acc / labels.size(0)
25     accs.append(acc)
26     if idx % log_interval == 0 and idx > 0:
27         elapsed = time.time() - start_time
28         print(
29             "| epoch {:3d} | {:5d}/{:5d} batches "
30             "| accuracy {:.3f}".format(
31                 epoch, idx, len(train_dataloader), sum(accs) / len(accs)
32             )
33         )
34         start_time = time.time()
35
36     epoch_acc = sum(accs) / len(accs)
37     epoch_loss = sum(losses) / len(losses)
38     return epoch_acc, epoch_loss
39
40 # evaluate
41 def evaluate_epoch(model, criterion, valid_dataloader):
42     model.eval()
43     accs, losses = [], []
44
45     with torch.no_grad():
46         for idx, (inputs, labels) in enumerate(valid_dataloader):
47             inputs = inputs.to(device)
48             labels = labels.to(device)
49
50             predictions = model(inputs)
51
52             loss = criterion(predictions, labels)
53             losses.append(loss.item())
54
55             total_acc = (predictions.argmax(1) == labels).sum().item()
56             acc = total_acc / labels.size(0)
57             accs.append(acc)
58
59     epoch_acc = sum(accs) / len(accs)
60     epoch_loss = sum(losses) / len(losses)
61     return epoch_acc, epoch_loss
62
63 # train model
64 def train(model, model_name, save_model, optimizer, criterion, train_dataloader,
65           valid_dataloader, num_epochs, device):
66     train_accs, train_losses = [], []
67     eval_accs, eval_losses = [], []
68     best_loss_eval = 100
```

```

68     times = []
69     for epoch in range(1, num_epochs+1):
70         epoch_start_time = time.time()
71         # Training
72         train_acc, train_loss = train_epoch(model, optimizer, criterion,
73         train_dataloader, device, epoch)
74         train_accs.append(train_acc)
75         train_losses.append(train_loss)
76
77         # Evaluation
78         eval_acc, eval_loss = evaluate_epoch(model, criterion, valid_dataloader)
79         eval_accs.append(eval_acc)
80         eval_losses.append(eval_loss)
81
82         # Save best model
83         if eval_loss < best_loss_eval:
84             torch.save(model.state_dict(), save_model + f'/{model_name}.pt')
85
86         times.append(time.time() - epoch_start_time)
87         # Print loss, acc end epoch
88         print("-" * 59)
89         print(
90             "| End of epoch {:3d} | Time: {:5.2f}s | Train Accuracy {:8.3f} | Train
91             Loss {:8.3f}"
92             "| Valid Accuracy {:8.3f} | Valid Loss {:8.3f} ".format(
93                 epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc,
94                 eval_loss
95             )
96         )
97         print("-" * 59)
98
99         # Load best model
100        model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
101        model.eval()
102        metrics = {
103            'train_accuracy': train_accs,
104            'train_loss': train_losses,
105            'valid_accuracy': eval_accs,
106            'valid_loss': eval_losses,
107            'time': times
108        }
109        return model, metrics
110
111    # plot result
112    def plot_result(num_epochs, train_accs, eval_accs, train_losses, eval_losses):
113        epochs = list(range(num_epochs))
114        fig, axs = plt.subplots(nrows = 1, ncols = 2 , figsize = (12,6))
115        axs[0].plot(epochs, train_accs, label = "Training")
116        axs[0].plot(epochs, eval_accs, label = "Evaluation")
117        axs[1].plot(epochs, train_losses, label = "Training")
118        axs[1].plot(epochs, eval_losses, label = "Evaluation")
119        axs[0].set_xlabel("Epochs")
120        axs[1].set_xlabel("Epochs")
121        axs[0].set_ylabel("Accuracy")
122        axs[1].set_ylabel("Loss")
123        plt.legend()

```

4. Load model ResNet18 với trọng số khởi tạo từ đầu

```

1 # load model with weights=None
2 base_model = models.resnet18(weights=None)

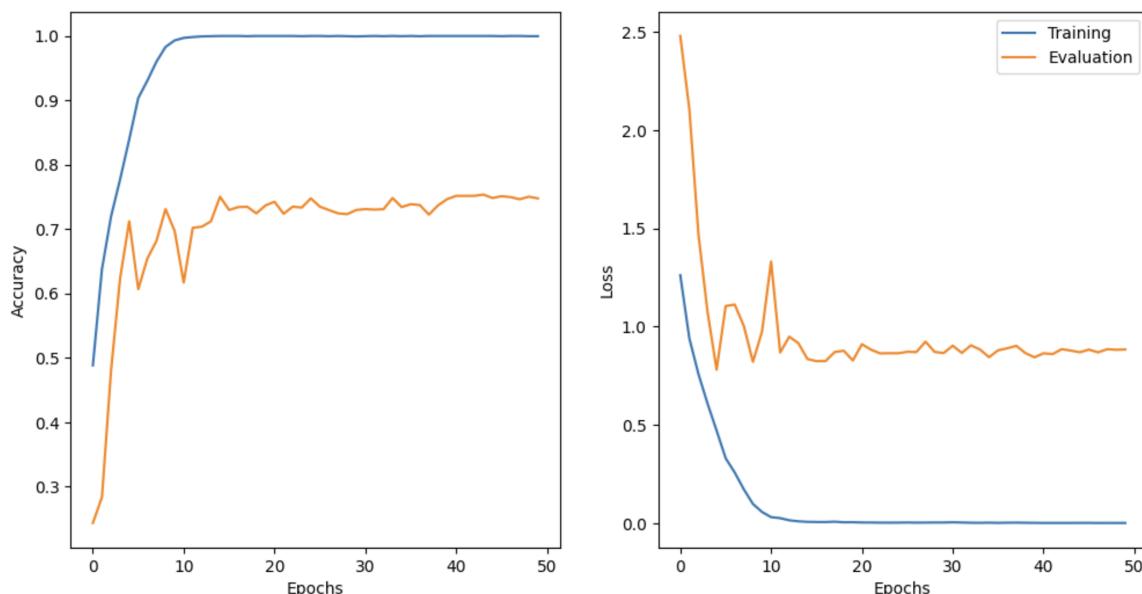
```

```

3
4 # custom linear layer from 1000 classes (ImageNet) to 5 class (Flower)
5 in_features = base_model.fc.in_features
6 base_model.fc = nn.Linear(in_features, num_classes)
7
8 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
9 base_model.to(device)
10
11 criterion = torch.nn.CrossEntropyLoss()
12 optimizer = optim.Adam(base_model.parameters(), lr=0.0001)
13
14 num_epochs = 50
15 save_model = './model'
16 os.makedirs(save_model, exist_ok = True)
17 model_name = 'base_model'
18
19 base_model, base_metrics = train(
20     base_model, model_name, save_model, optimizer, criterion, train_dataloader,
21     valid_dataloader, num_epochs, device
21 )

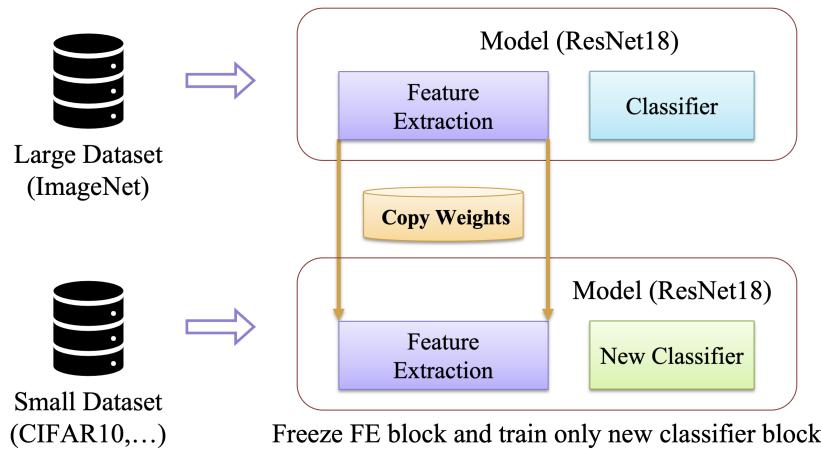
```

5. Quá trình huấn luyện



Hình 3: Huấn luyện mô hình ResNet18 từ đầu trên bộ dữ liệu Flower.

2.2. Transfer Learning



Hình 4: Kỹ thuật transfer learning.

Ở phần này, chúng ta sử dụng kiến trúc mô hình ResNet18 đã được huấn luyện trên bộ dữ liệu ImageNet như là phần trích xuất đặc trưng. Vì vậy chúng sẽ đóng băng (freeze) tất cả các layer thuộc phần feature extraction trong mạng ResNet và chỉ huấn luyện phần phân loại mới cho bộ dữ liệu Flower.

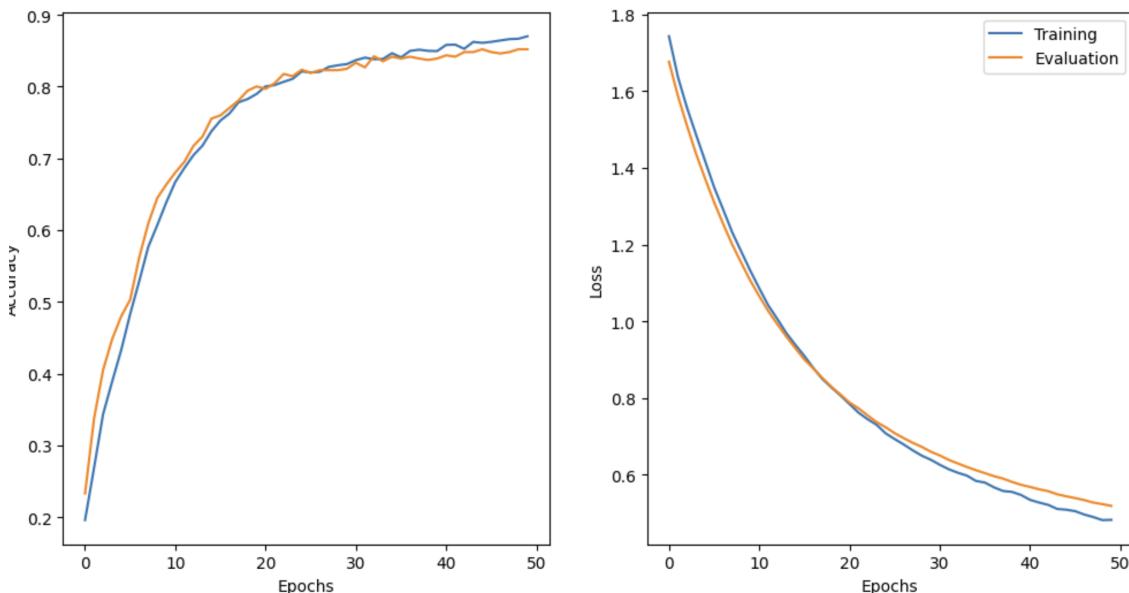
Quá trình chuẩn bị dữ liệu, xây dựng các hàm để huấn luyện mô hình tương tự với phần 1. Phần này chúng ta tải về mô hình ResNet18 cùng với bộ trọng số và đóng băng các layer cho phần feature extraction.

```

1 # load model and its weight
2 transfer_model = models.resnet18(
3     weights=models.ResNet18_Weights.IMAGENET1K_V1
4 )
5 # freeze FE layer
6 for param in transfer_model.parameters():
7     param.requires_grad = False
8
9 # custom linear layer
10 in_features = transfer_model.fc.in_features
11 transfer_model.fc = nn.Linear(in_features, num_classes)
12
13 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
14 transfer_model.to(device)
15
16 criterion = torch.nn.CrossEntropyLoss()
17 optimizer = optim.Adam(transfer_model.parameters(), lr=0.0001)
18
19 num_epochs = 50
20 save_model = './model'
21 os.makedirs(save_model, exist_ok = True)
22 model_name = 'transfer_model'
23
24 transfer_model, transfer_metrics = train(
25     transfer_model, model_name, save_model, optimizer, criterion, train_dataloader,
26     valid_dataloader, num_epochs, device
26 )

```

Kết quả huấn luyện mô hình:

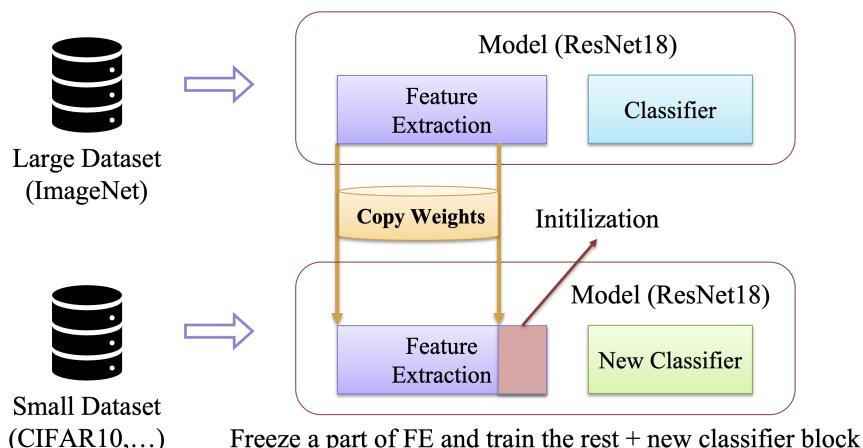


Hình 5: Quá trình huấn luyện bộ dữ liệu Flower với kỹ thuật Transfer Learning.

2.3. Fine Tuning

Ở phần này, chúng ta sử dụng kiến trúc mô hình ResNet18 đã được huấn luyện trên bộ dữ liệu ImageNet, thay vì đóng băng tất cả layer trong mô hình, với fine tuning chúng ta chỉ đóng băng 1 số hoặc không đóng băng bất kỳ layer nào (lấy bộ trọng số ResNet18 làm trọng số khởi tạo của mô hình).

- Đóng băng một số layer.



Hình 6: Kỹ thuật finetuning (Freeze một số layers).

```

1 # load model and its weight
2 fine_tuning_model = models.resnet18(
3     weights=models.ResNet18_Weights.IMGNET1K_V1
4 )
5
6 # freeze the first 50 layers
7 num_layers = 50
8 for index, param in enumerate(fine_tuning_model.parameters()):

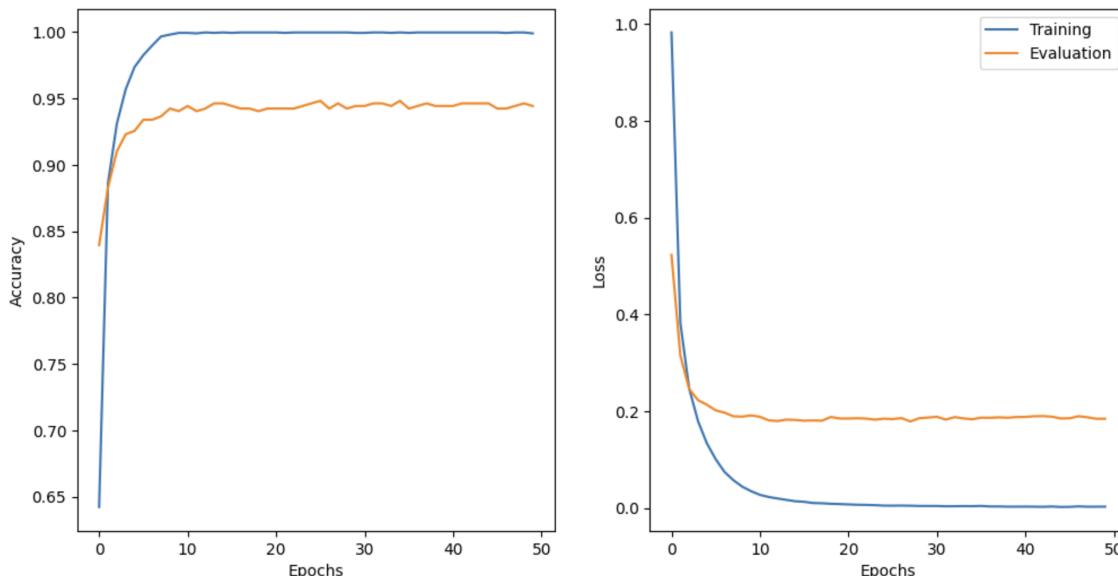
```

```

9     if index < num_layers:
10        param.requires_grad = False
11
12 # custom linear layer
13 in_features = fine_tuning_model.fc.in_features
14 fine_tuning_model.fc = nn.Linear(in_features, num_classes)
15
16 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
17 fine_tuning_model.to(device)
18
19 criterion = torch.nn.CrossEntropyLoss()
20 optimizer = optim.Adam(fine_tuning_model.parameters(), lr=0.0001)
21
22 num_epochs = 50
23 save_model = './model'
24 os.makedirs(save_model, exist_ok = True)
25 model_name = 'fine_tuning_model'
26
27 fine_tuning_model, fine_tuning_metrics = train(
28     fine_tuning_model, model_name, save_model, optimizer, criterion, train_dataloader,
29     valid_dataloader, num_epochs, device
)

```

Kết quả huấn luyện mô hình:



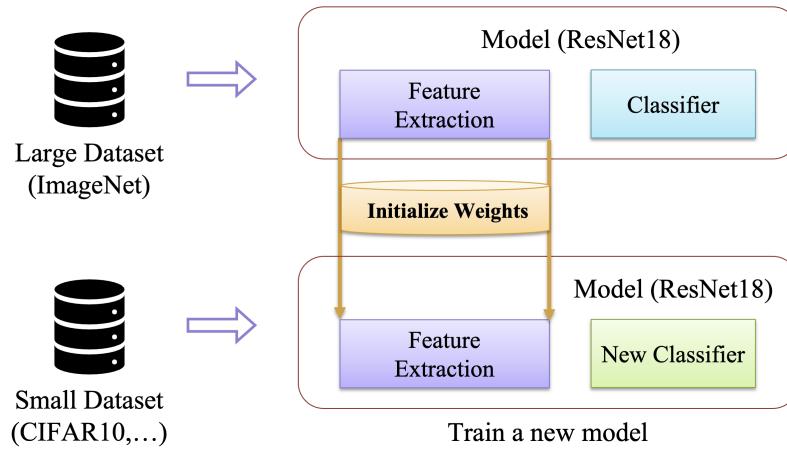
Hình 7: Quá trình huấn luyện bộ dữ liệu Flower với kỹ thuật Fine Tuning.

- Cập nhật tất cả layer (Trọng số của pretrained ResNet18 là trọng số khởi tạo cho mô hình phân loại trên bộ dữ liệu Flower)

```

1 # load model and its weight
2 initialization_model = models.resnet18(
3     weights=models.ResNet18_Weights.IMGNET1K_V1
4 )
5
6 in_features = initialization_model.fc.in_features
7 initialization_model.fc = nn.Linear(in_features, num_classes)
8
9 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```



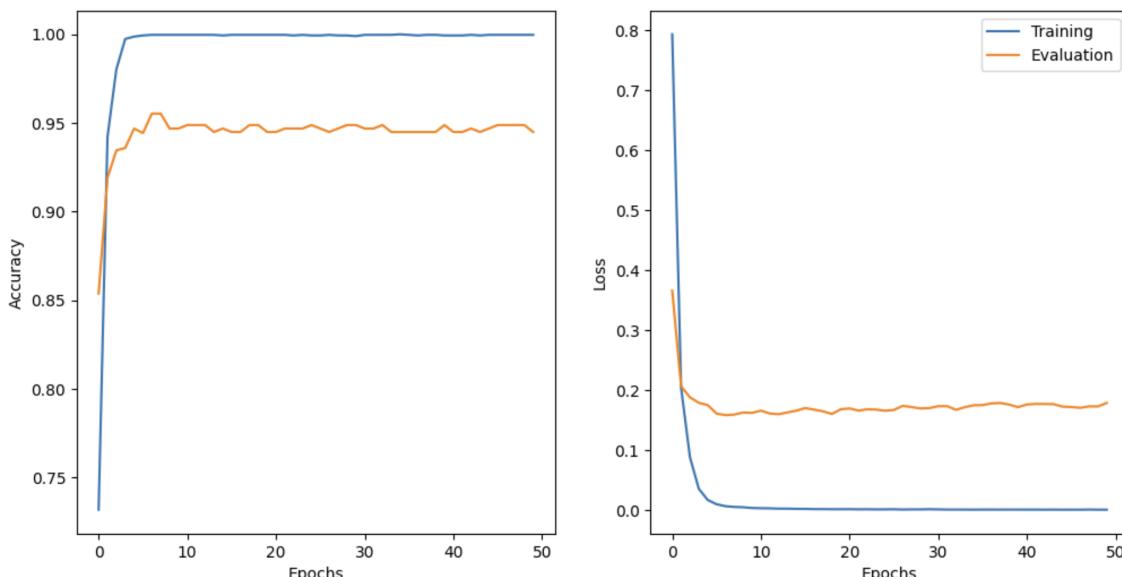
Hình 8: Kỹ thuật finetuning (Sử dụng ResNet18 là trọng số khởi tạo).

```

10 initialization_model.to(device)
11
12 criterion = torch.nn.CrossEntropyLoss()
13 optimizer = optim.Adam(initialization_model.parameters(), lr=0.0001)
14
15 num_epochs = 50
16 save_model = './model'
17 os.makedirs(save_model, exist_ok = True)
18 model_name = 'initialization_model'
19
20 initialization_model, initilization_metrics = train(
21     initialization_model, model_name, save_model, optimizer, criterion,
22     train_dataloader, valid_dataloader, num_epochs, device
22 )

```

Kết quả huấn luyện mô hình:

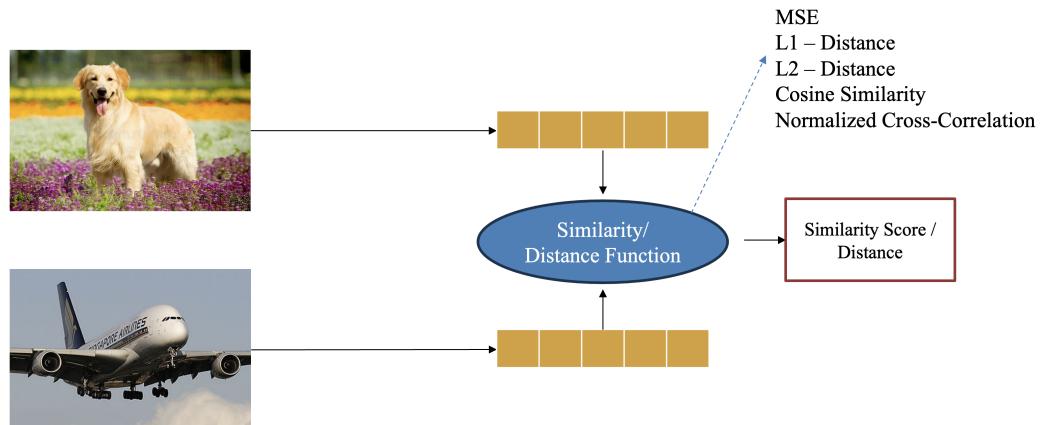


Hình 9: Quá trình huấn luyện bộ dữ liệu Flower với kỹ thuật Fine Tuning (Khởi tạo trọng số từ ResNet18).

Phần 2. Image Retrieval

Mục tiêu phần này, xây dựng mô hình tìm kiếm hình ảnh với đầu vào ảnh truy vấn và tập dữ liệu các ảnh trong bộ cơ sở dữ liệu. Chúng ta cần tìm những ảnh trong tập dữ liệu này giống nhất với ảnh truy vấn.

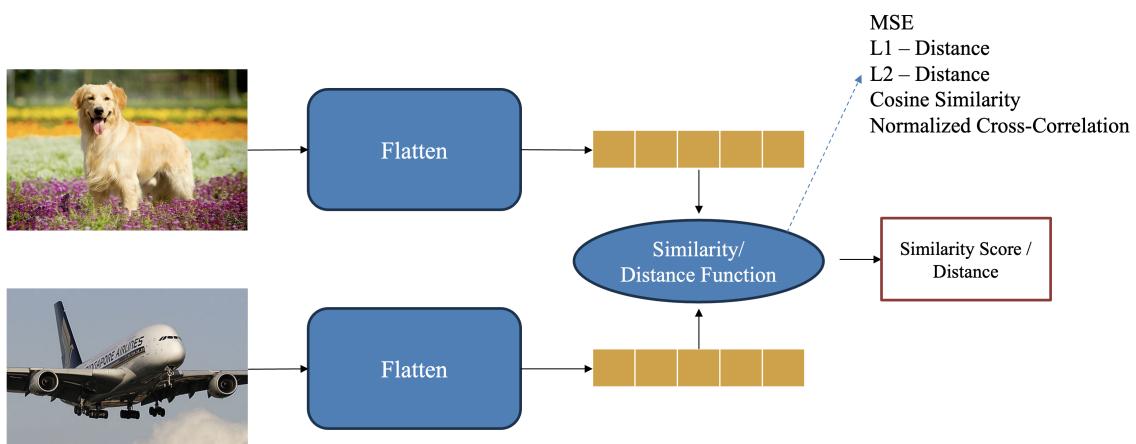
Phương pháp tiếp cận: chúng ta chuyển ảnh truy vấn và các ảnh trong bộ dữ liệu thành các vector đại diện. Sau đó sử dụng độ đo như cosine similarity,... để tính độ tương đồng giữa các ảnh và chọn ảnh có độ tương đồng lớn nhất.



Hình 10: Phương pháp đánh giá độ tương đồng hai ảnh.

2.1. Phương pháp cơ bản

Chúng ta sử dụng flatten layer để chuyển các hình ảnh thành các vector đại diện.



Hình 11: Phương pháp đánh giá độ tương đồng hai ảnh.

1. Tải về bộ dữ liệu ảnh

```

1 # download unzip dataset
2 !gdown 1aZVKTlWlrQ01LvepLasxy6qFuCl4SJm
3 !unzip /content/images_mr.zip -d /content/drive/MyDrive/img
4
5 # load dataset

```

```

6 # Image transformation: resize and convert to tensor
7 transform = transforms.Compose([
8     transforms.Resize((86, 128)),
9     transforms.ToTensor(),
10])
11
12 # Function to load and preprocess the image
13 def load_image(path, transform):
14     img = Image.open(path).convert('RGB')
15     img_tensor = transform(img)
16     return img_tensor

```

2. Tiền xử lý dữ liệu

```

1 # Img folder path and Save path (Change if needed)
2 img_path = '/content/drive/MyDrive/img/images_mr'
3 save_path = '/content/drive/MyDrive/img/images_mr.pt'
4
5 # Save images as tensor
6 lists = [i for i in range(9908)]
7 images = []
8
9 for index in tqdm(lists):
10     # Load image
11     try:
12         img_tensor = load_image(f'{img_path}/{index}.jpg', transform)
13         images.append(img_tensor)
14     except FileNotFoundError:
15         print(f'Image file not found: {img_path}/{index}.jpg')
16
17 # Stack all image tensors into one big tensor
18 img_tensor = torch.stack(images)
19 print(img_tensor.shape)
20
21 # Save the tensor
22 torch.save(img_tensor, save_path)
23 print(f"Saved tensor to {save_path}.")
24
25 # Load the img tensor file
26 img_tensors = torch.load(save_path)
27
28 # Convert the tensor to a PIL image
29 image_pil = to_pil_image(img_tensors[0])
30
31 # Display the image using matplotlib
32 plt.figure(figsize=(4, 4))
33 plt.imshow(image_pil)
34 plt.axis('off') # Turn off axis numbers
35 plt.show()

```

3. Load query image

```

1 # Load query image
2 query_img_path = '/content/drive/MyDrive/img/q2.jpg' # Update with the correct path
3 query_tensor = load_image(query_img_path, transform)
4 print(f'Img tensor shape: {query_tensor.shape}')
5
6 # Display query image
7 plt.figure(figsize=(4, 4))
8 plt.imshow(transforms.ToPILImage()(query_tensor))

```

```

9 plt.title("Query Image")
10 plt.axis('off')
11 plt.show()
12
13 # Flatten the query img
14 query_tensor_flat = query_tensor.view(1, -1)
15 print(f'Img tensor shape: {query_tensor_flat.shape}')
16
17 # Load the saved tensor file containing all images
18 data_tensors = torch.load('/content/drive/MyDrive/img/images_mr.pt')
19 print(f'Data tensors shape: {data_tensors.shape}')
20 data_tensors_flat = data_tensors.view(data_tensors.size(0), -1)
21 print(f'Data tensors shape: {data_tensors_flat.shape}')

```



Hình 12: Ảnh truy vấn.

4. Truy vấn sử dụng độ đo khoảng cách L1

```

1 # Calculate L1 distance (absolute difference)
2 distances = torch.abs(data_tensors_flat - query_tensor_flat)
3 distances = torch.sum(distances, dim=1)
4
5 # Sort the distances and get indices
6 sorted_indices = torch.argsort(distances, descending=False)
7
8 # Print top 8 values and their indices
9 for i in range(8):
10     index = sorted_indices[i].item()
11     distance = distances[index].item()
12     print(f'Index: {index}, Distance: {distance}')
13
14 # Display top 8 similar images
15 fig = plt.figure(figsize=(8, 8))
16 columns = 3
17 rows = 3
18 for i in range(columns * rows):
19     index = sorted_indices[i].item()
20     img = data_tensors[index]
21
22     ax = fig.add_subplot(rows, columns, i + 1)
23     ax.axis('off')
24     ax.imshow(to_pil_image(img))
25
26 plt.show()

```

```

Index: 3425, Distance: 0.0
Index: 3077, Distance: 5221.431640625
Index: 2611, Distance: 5247.2900390625
Index: 3109, Distance: 5289.16845703125
Index: 3445, Distance: 5358.61962890625
Index: 960, Distance: 5378.94873046875
Index: 4788, Distance: 5395.0078125
Index: 3094, Distance: 5435.6904296875

```



Hình 13: Kết quả truy vấn sử dụng độ đo L1.

5. Truy vấn sử dụng độ đo tương đồng cosine

```

1 # Calculate cosine similarity
2 sims = cosine_similarity(query_tensor_flat, data_tensors_flat)
3
4 # Sort the similarities and get indices
5 sorted_indices = torch.argsort(sims, descending=True)
6
7 # Print top 8 values and their indices
8 for i in range(8):
9     index = sorted_indices[i].item()
10    similarity = sims[index].item()
11    print(f'Index: {index}, Similarity: {similarity}')
12
13 # Display top 8 similar images
14 fig = plt.figure(figsize=(8, 8))
15 columns = 3
16 rows = 3
17 for i in range(columns * rows):
18     index = sorted_indices[i].item()
19     img = data_tensors[index]
20
21     ax = fig.add_subplot(rows, columns, i + 1)
22     ax.axis('off')
23     ax.imshow(to_pil_image(img))
24
25 plt.show()

```

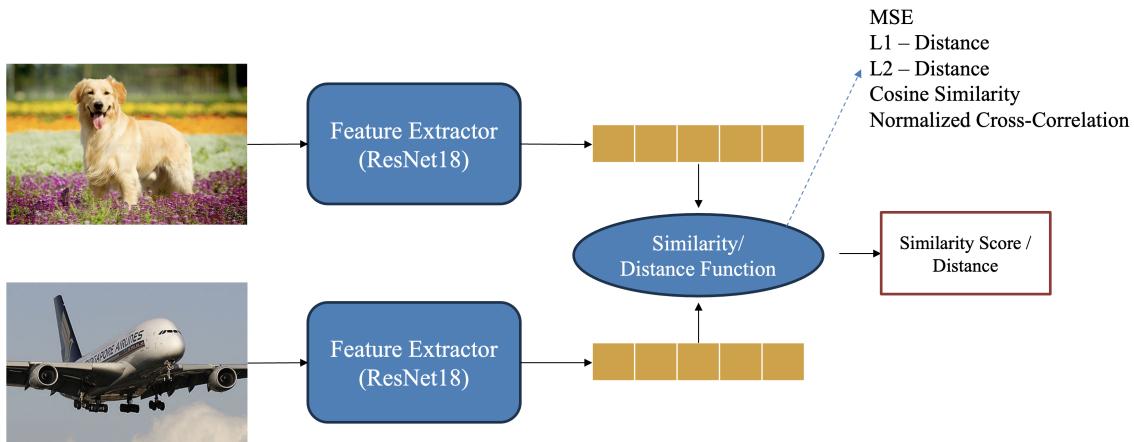
Index: 3425, Similarity: 0.999997615814209
Index: 3306, Similarity: 0.8691879510879517
Index: 4645, Similarity: 0.8683653473854065
Index: 3330, Similarity: 0.8676211833953857
Index: 3374, Similarity: 0.867255449295044
Index: 4317, Similarity: 0.8664509057998657
Index: 3053, Similarity: 0.8658316135406494
Index: 3780, Similarity: 0.8647940158843994



Hình 14: Kết quả truy vấn sử dụng độ tương đồng cosine.

2.2. Sử dụng pretrained model

Chúng ta sử dụng pretrained model (feature extractor) để chuyển các hình ảnh thành các vector đại diện.



Hình 15: Phương pháp đánh giá độ tương đồng hai ảnh sử dụng pretrained model.

1. Tải về pretrained ResNet18 model

```

1 # Load the pretrained ResNet18 model
2 resnet18 = models.resnet18(pretrained=True)
3 # Remove the last layer to use as a feature extractor
4 modules = list(resnet18.children())[:-1]
5 resnet18 = torch.nn.Sequential(*modules)
6
7 resnet18.eval() # Set the model to evaluation mode

```

2. Trích xuất đặc trưng

```

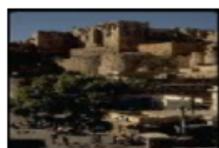
1 # Function to preprocess and extract features
2 def extract_features(tensor, model):
3     with torch.no_grad():
4         # Add batch dimension and get features
5         features = model(tensor.unsqueeze(0))
6         # Flatten the features
7         features = features.view(features.size(0), -1)
8         return features
9
10 # Extract features for each image and store them
11 feature_list = []
12 for tensor in data_tensors:
13     # print(tensor.unsqueeze(0).shape)
14     features = extract_features(tensor, resnet18)
15     feature_list.append(features)
16
17 # Stack all features into a single tensor
18 feature_tensor = torch.stack(feature_list).squeeze()
19 print(feature_tensor.shape)
20 torch.save(feature_tensor, '/content/drive/MyDrive/img/images_mr_features.pt')

```

3. Truy vấn sử dụng độ tương đồng cosine

```
1 # Function to preprocess and extract features
2 # Load the query image and extract features
3 query_tensor = load_image(query_img_path, transform)
4 query_features = extract_features(query_tensor, resnet18)
5
6 # Load the feature tensor
7 feature_tensor = torch.load('/content/drive/MyDrive/img/images_mr_features.pt')
8
9 # Calculate cosine similarity
10 sims = cosine_similarity(query_features, feature_tensor)
11
12 # Sort the similarities and get indices
13 sorted_indices = torch.argsort(sims, descending=True)
14
15 # Print top 8 values and their indices
16 for i in range(8):
17     index = sorted_indices[i].item()
18     similarity = sims[index].item()
19     print(f'Index: {index}, Similarity: {similarity}')
20
21 # Display top 8 similar images
22 fig = plt.figure(figsize=(8, 8))
23 columns = 3
24 rows = 3
25 for i in range(columns * rows):
26     index = sorted_indices[i].item()
27     img = data_tensors[index]
28
29     ax = fig.add_subplot(rows, columns, i + 1)
30     ax.axis('off')
31     ax.imshow(to_pil_image(img))
32
33 plt.show()
```

Index: 3425, Similarity: 1.0000001192092896
Index: 3407, Similarity: 0.6934022903442383
Index: 370, Similarity: 0.687757670879364
Index: 3214, Similarity: 0.6866506338119507
Index: 3667, Similarity: 0.6857972741127014
Index: 3509, Similarity: 0.6846740245819092
Index: 248, Similarity: 0.6844337582588196
Index: 3793, Similarity: 0.6839574575424194



Hình 16: Kết quả truy vấn sử dụng độ tương đồng cosine dựa vào đặc trưng trích xuất từ mô hình pretrained ResNet18.

Phần 3. Vector Database

Để tăng tốc độ truy vấn và quản lý tốt hơn cơ sở dữ liệu ảnh. Phần này chúng ta sử dụng vector database với thư viện pymilvus và lưu trữ dữ liệu trên Zilliz Cloud.



Hình 17: Vector Database với thư viện pymilvus và Zilliz Cloud.

3.1. Tạo tài khoản Zilliz

- Bước 1: Truy cập trang chủ Zilliz: <https://zilliz.com/> và tạo tài khoản miễn phí.
- Bước 2: Tạo Cluster để lưu trữ các Collection
- Bước 3: Tạo Collection: ResNet18

3.2. Kết nối đến Zilliz Cloud

```

1 # install libs
2 !pip install grpcio==1.49.1 pymilvus==2.3.3
3
4 # connect database
5 from pymilvus import MilvusClient, connections
6
7 CLUSTER_ENDPOINT = "####" # get from Zilliz Cloud after create cluster
8 TOKEN = "####" # get from Zilliz Cloud after create cluster
9
10 connections.connect(
11     "default",
12     uri=CLUSTER_ENDPOINT,
13     token=TOKEN
14 )
15
16 client = MilvusClient(
17     uri=CLUSTER_ENDPOINT,
18     token=TOKEN
19 )

```

3.3. Tạo Collection

Tạo một Collection để lưu trữ embedding của các ảnh với 2 trường: ID số thứ tự ảnh và Embedding: vector biểu diễn của ảnh

```

1 from pymilvus import Collection, DataType, FieldSchema, CollectionSchema, utility
2

```

```

3 # create a collection
4 COLLECTION_NAME = "ResNet18"
5 check_collection = utility.has_collection(COLLECTION_NAME)
6 if check_collection:
7     drop_result = utility.drop_collection(COLLECTION_NAME)
8
9 EMBEDDING_DIM=512
10 image_id = FieldSchema(
11     name="image_id",
12     dtype=DataType.INT64,
13     is_primary=True,
14     description="Image ID"
15 )
16
17 image_embedding = FieldSchema(
18     name="image_embedding",
19     dtype=DataType.FLOAT_VECTOR,
20     dim=EMBEDDING_DIM
21 )
22
23 schema = CollectionSchema(
24     fields=[image_id, image_embedding],
25     auto_id=False,
26     description="Image Retrieval Using ResNet18")
27
28 collection = Collection(
29     name=COLLECTION_NAME,
30     schema=schema
31 )

```

3.4. Thêm dữ liệu vào Collection

Trích xuất đặc trưng ảnh từ ResNet18 sau đó chúng ta thêm các dữ liệu vào ResNet18 Collection

```

1 import os
2 from tqdm import tqdm
3
4 from PIL import Image
5 import torch
6 import torch.nn as nn
7 from torchvision import models, transforms
8
9 transform = transforms.Compose([
10     transforms.Resize((224, 224)),
11     transforms.ToTensor(),
12 ])
13
14 def preprocess_image(image_path):
15     img = Image.open(image_path).convert('RGB')
16     processed_img = transform(img)
17     return processed_img
18 # load model
19 resnet18_model = models.resnet18(
20     weights=models.ResNet18_Weights.IMAGENET1K_V1
21 )
22
23 modules = list(resnet18_model.children())[:-1]
24 resnet18_model = torch.nn.Sequential(*modules)
25 resnet18_model.eval()
26
27 def extract_feature(processed_image):
28     input = processed_image.unsqueeze(0).to(device)

```

```

29     with torch.no_grad():
30         prediction = resnet18_model(input)
31     return prediction.squeeze().cpu().tolist()
32
33 # download image
34 !gdown 1aZVKTlWlrQ01LvepLasxy6qFuC14SJm
35 image_folder = "./images_mr"
36
37 # extract feature
38 image_ids = sorted([
39     int(image_name.split(".")[0]) for image_name in os.listdir(image_folder)
40 ])
41 image_embeddings = []
42 for file_name in tqdm(image_ids):
43     file_name = str(file_name) + ".jpg"
44     image_path = os.path.join(image_folder, file_name)
45     processed_image = preprocess_image(image_path)
46     processed_image = extract_feature(processed_image)
47     image_embeddings.append(processed_image)
48
49 # insert entities
50 entities = [image_ids, image_embeddings]
51 ins_resp = collection.insert(entities)
52 ins_resp
53
54 #flush
55 collection.flush()

```

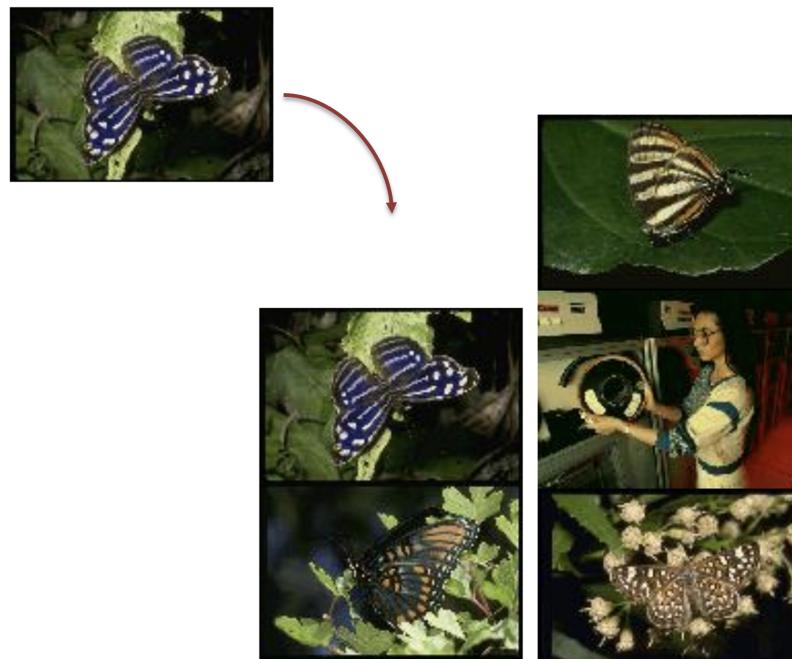
3.5. Searching

```

1 # create index with L2 score
2 index_params = {
3     "index_type": "IVF_FLAT",
4     "metric_type": "L2",
5     "params": {}
6 }
7 collection.create_index(
8     field_name=image_embedding.name,
9     index_params=index_params
10 )
11
12 # load collection
13 collection.load()
14
15 # search
16 search_params = {
17     "metric_type": "L2",
18     "params": {"level": 2}
19 }
20 def search_images(image_path, topk=5):
21     processed_image = preprocess_image(image_path)
22     processed_image = extract_feature(processed_image)
23     results = collection.search(
24         [processed_image],
25         anns_field=image_embedding.name,
26         param=search_params,
27         limit=topk,
28         guarantee_timestamp=1
29     )
30     return results[0]

```

```
31
32 # query image
33 !gdown 1aZHrliKuLOTtzM6vvLj82lwXT9Vjrsfd
34 image_path = "./q1.jpg"
35 results = search_images(image_path)
36 results
```



Hình 18: Kết quả truy vấn hình ảnh sử dụng vector database.

Phần 4. Câu hỏi trắc nghiệm

Câu hỏi 1 Mô hình pretrained được sử dụng trong bài tập này là?

- a) ResNet18
- b) VGG16
- c) ResNet50
- d) LeNet

Câu hỏi 2 Phát biểu đúng cho kỹ thuật transfer learning là?

- a) Sử dụng trọng số mô hình pretrained là trọng số khởi tạo cho một hoặc một số lớp của mô hình mới
- b) Sử dụng khối feature extraction của mô hình pretrained để trích xuất đặc trưng cho ảnh và huấn luyện phần classifier mới
- c) Không sử dụng bất kỳ trọng số của mô hình pretrained
- d) Huấn luyện mô hình mới từ đầu

Câu hỏi 3 Phát biểu đúng cho kỹ thuật Fine Tuning là?

- a) Sử dụng trọng số mô hình pretrained là trọng số khởi tạo cho một hoặc một số lớp của mô hình mới
- b) Sử dụng khối feature extraction của mô hình pretrained để trích xuất đặc trưng cho ảnh và huấn luyện phần classifier mới
- c) Không sử dụng bất kỳ trọng số của mô hình pretrained
- d) Huấn luyện mô hình mới từ đầu

Câu hỏi 4 Bộ dữ liệu thường được sử dụng để huấn luyện các mô hình pretrained là?

- a) CIFAR10
- b) CIFAR100
- c) MNIST
- d) ImageNet

Câu hỏi 5 Bộ dữ liệu được sử dụng để so sánh các kỹ thuật transfer learning và fine tuning là?

- a) CIFAR10
- b) CIFAR100
- c) Flower
- d) MNIST

Câu hỏi 6 Phương pháp nào có accuracy cao nhất trên bộ dữ liệu Flower?

- a) Train từ đầu
- b) Transfer Learning
- c) Fine Tuning (Với một số layers)
- d) Fine Tuning (Khởi tạo trọng số)

Câu hỏi 7 Độ đo để so sánh độ tương đồng hai ảnh là?

- a) L1 Distance
- b) L2 Distance
- c) Cosine Similarity
- d) Tất cả các đáp án trên

Câu hỏi 8 Vector Database dùng để lưu trữ và truy vấn với kiểu dữ liệu nào sau đây?

- a) Văn bản
- b) Âm thanh
- c) Hình ảnh
- d) Tất cả các đáp án trên

Câu hỏi 9 Thư viện hỗ trợ lưu trữ và quản lý dữ liệu vector được sử dụng là?

- a) PyMilvus
- b) Sklearn
- c) Pandas
- d) NLTK

Câu hỏi 10 Độ đo được sử dụng để đánh index trong bài là?

- a) L1 Distance
- b) L2 Distance
- c) Cosine Similarity
- d) Tất cả các đáp án trên

- *Hết* -