Đồ án phân loại rác thải dựa trên mô hình học sâu

Nguyễn Nhật Hiếu

1 Giới thiệu

Đồ án này nhằm mục tiêu xây dựng một mô hình phân loại hình ảnh rác thải tự động sử dụng học sâu (Deep Learning). Đồ án sử dụng tập dữ liệu về các loại rác thải, bao gồm sáu loại riêng biệt: **thủy tinh (glass)**, **kim loại (metal)**, **nhựa (plastic)**, **bìa cứng (cardboard)**, **giấy (paper)**, và **rác thải hỗn hợp (trash)**. Tập dữ liệu này được tải về từ Kaggle và chứa các hình ảnh rác thải được phân loại theo từng nhóm. Đồ án này nhằm mục đích ứng dụng để hỗ trợ quản lý và tái chế chất thải hiệu quả hơn.

Mục tiêu:

1. **Chuẩn bị dữ liệu**: Thu thập và tiền xử lý dữ liệu, đảm bảo dữ liệu sẵn sàng cho quá trình huấn luyện và đánh giá mô hình.
2. **Trực quan hóa**: Phân tích và trực quan hóa tập dữ liệu để hiểu rõ phân bố lớp và sự đa dạng của mẫu dữ liệu.
3. **Thiết kế mô hình**: Xây dựng mô hình học sâu tùy chỉnh dựa trên ResNet-18, sử dụng kỹ thuật học chuyển tiếp *(transfer learning)* để cải thiện độ chính xác.
4. **Huấn luyện và đánh giá**: Huấn luyện mô hình với tập dữ liệu đã chuẩn bị, đánh giá hiệu suất và tinh chỉnh để đạt kết quả tốt hơn.
5. **Ứng dụng thực tế**: Lưu mô hình đã huấn luyện và trình bày khả năng dự đoán thông qua trực quan hóa kết quả.

2 Nhập các thư viện cần thiết

Trong phần này, chúng ta nhập tất cả các thư viện cần thiết cho xử lý dữ liệu, xử lý hình ảnh, trực quan hóa và xây dựng mô hình. Các thư viện chính bao gồm:

- **Pandas**: Thao tác và quản lý dữ liệu dưới dạng bảng.
- **Seaborn và Matplotlib**: Trực quan hóa dữ liệu với đồ thị và biểu đồ.
- **PyTorch và TorchVision**: Xây dựng mô hình học sâu và xử lý dữ liệu hình ảnh.
- **PIL (Pillow)**: Đọc và xử lý các hình ảnh.
- **KaggleHub**: Tải dữ liệu trực tiếp từ Kaggle.

[1]:
```
import os

import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

import torchvision.models as models
from torchvision.io import read_image
from torchvision import transforms

from PIL import Image

from matplotlib import pyplot as plt

import kagglehub
```

3 Tải và chuẩn bị dữ liệu

Trong dụ án này, chúng ta sử dụng tập dữ liệu **Garbage Classification Dataset**. Đây là một tập dữ liệu mã nguồn mở được chia sẻ trên Kaggle. Dữ liệu được tải về từ Kaggle bằng cách sử dụng thư viện `KaggleHub`.Đường dẫn đến tập dữ liệu sau khi tải về được lưu trong biến `path_to_dataset`.

[2]:
```
path = kagglehub.dataset_download("asdasdasasdas/garbage-classification")
path_to_dataset = path + '/Garbage classification/Garbage classification'
print("Path to dataset files:", path_to_dataset)
```

```
Path to dataset files: /root/.cache/kagglehub/datasets/asdasdasasdas/garbage-
classification/versions/2/Garbage classification/Garbage classification
```

Sau khi tải dữ liệu, chúng ta thu thập đường dẫn đến các tệp hình ảnh trong tập dữ liệu vào biến `img_paths` và gán nhãn tương ứng cho từng hình ảnh, dựa trên tên thư mục, rồi lưu vào biến `labels`. Tập dữ liệu được tổ chức dưới dạng `DataFrame` để dễ dàng thao tác.

[3]:
```
classes_list = ['glass', 'metal', 'plastic', 'cardboard', 'paper', 'trash']
img_paths = []
labels = []
for class_name in classes_list:
    class_path = os.path.join(path_to_dataset, class_name)
    for img in os.listdir(class_path):
        img_path = os.path.join(class_path, img)
        img_paths.append(img_path)
        labels.append(class_name)
```

```
df = pd.DataFrame({
    'img_path':img_paths,
    'label':labels
})

df = df.sample(frac=1, random_state=42).reset_index(drop=True)

df.head()
```

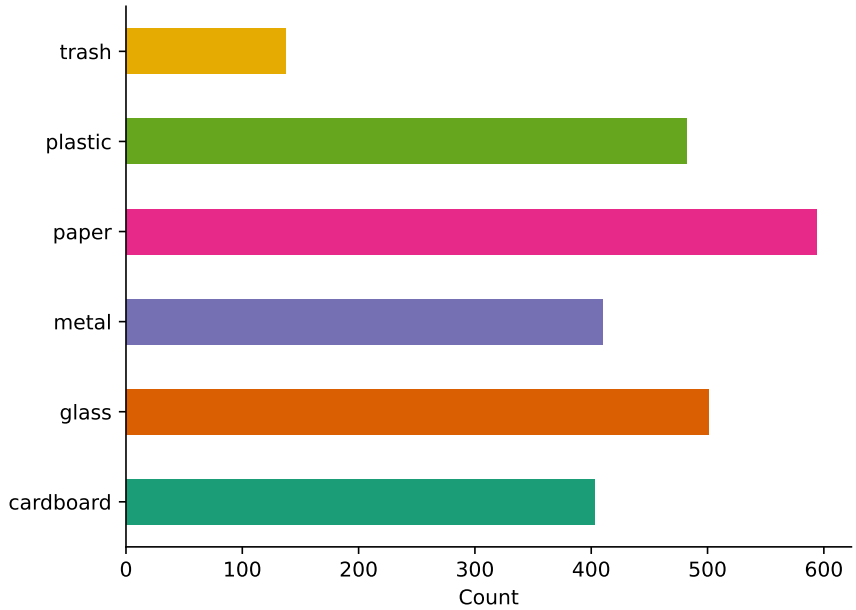
[3]:

	img_path	label
0	/root/.cache/kagglehub/datasets/asdasdasasdas/...	cardboard
1	/root/.cache/kagglehub/datasets/asdasdasasdas/...	metal
2	/root/.cache/kagglehub/datasets/asdasdasasdas/...	glass
3	/root/.cache/kagglehub/datasets/asdasdasasdas/...	plastic
4	/root/.cache/kagglehub/datasets/asdasdasasdas/...	plastic

4 Trực quan hóa dữ liệu

Trực quan hóa phân bố các lớp hình ảnh trong tập dữ liệu:

[4]:
```
df.groupby('label').size().plot(
    kind='barh',
    color=sns.palettes.mpl_palette('Dark2')
)
plt.gca().spines[['top', 'right']].set_visible(False)
plt.xlabel('Count')
```



5 Tạo ánh xạ nhãn

Tiếp theo, chúng ta sẽ tạo hai từ điển để ánh xạ giữa nhãn và chỉ số. Sử dụng ánh xạ này, ta có thể chuyển đổi nhãn của một tệp hình ảnh thành chỉ số để phục vụ cho bài toán phân loại trong học máy. Ngược lại, khi mô hình đưa ra dự đoán dưới dạng chỉ số, ta sẽ sử dụng ánh xạ ngược để chuyển đổi nó thành nhãn dán tương ứng cho tệp hình ảnh đầu vào.

[5]:
```
index_to_label = {index: label for index, label in enumerate(classes_list)}
label_to_index = {label: index for index, label in enumerate(classes_list)}

print("Index to Label Mapping:", index_to_label)
print("Label to Index Mapping:", label_to_index)
```

```
Index to Label Mapping: {0: 'glass', 1: 'metal', 2: 'plastic', 3: 'cardboard',
4: 'paper', 5: 'trash'}
Label to Index Mapping: {'glass': 0, 'metal': 1, 'plastic': 2, 'cardboard': 3,
'paper': 4, 'trash': 5}
```

Áp dụng ánh xạ chuyển đổi từ nhãn thành chỉ số cho tập dữ liệu và lưu kết quả vào cột `label_index` của bảng dữ liệu:

[6]:
```
df['label_index'] = df['label'].map(label_to_index)
```

Tập dữ liệu sẽ được chia thành hai phần: tập huấn luyện (80%) và tập đánh giá (20%):

[7]:
```
df_train, df_test = train_test_split(df, test_size=0.2, random_state=42)
df_train = df_train.reset_index(drop=True)
df_test = df_test.reset_index(drop=True)
```

[8]:
```
df_train.head()
```

[8]:

	img_path	label	label_index
0	/root/.cache/kagglehub/datasets/asdasdasasdas/...	plastic	2
1	/root/.cache/kagglehub/datasets/asdasdasasdas/...	metal	1
2	/root/.cache/kagglehub/datasets/asdasdasasdas/...	metal	1
3	/root/.cache/kagglehub/datasets/asdasdasasdas/...	glass	0
4	/root/.cache/kagglehub/datasets/asdasdasasdas/...	cardboard	3

6 Tạo class Dataset tùy chỉnh

Trong bài toán phân loại hình ảnh này, chúng ta cần xử lý dữ liệu hình ảnh trước khi đưa vào mô hình học sâu. PyTorch cung cấp lớp `Dataset` để giúp người dùng dễ dàng quản lý và truy xuất dữ liệu. Lớp `Dataset` này cho phép chúng ta tổ chức dữ liệu theo cách tùy chỉnh và sử dụng cùng với `DataLoader` để thực hiện huấn luyện mô hình.

[9]:
```
class ImageDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.img_labels = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = self.img_labels.iloc[idx]['img_path']
        image = read_image(img_path).float() / 255.0
        label = self.img_labels.iloc[idx]['label_index']
        if self.transform:
            image = self.transform(image)
        return image, label
```

7 Áp dụng biến đổi dữ liệu và tạo DataLoader

Trong bài toán học sâu với dữ liệu hình ảnh, một bước quan trọng trước khi đưa dữ liệu vào mô hình là thực hiện các phép biến đổi để chuẩn bị hình ảnh sao cho phù hợp với yêu cầu của mô hình. Các phép biến đổi này giúp cải thiện hiệu suất của mô hình và làm cho dữ liệu đồng nhất hơn.

Trong đồ án này, chúng ta chủ yếu sử dụng các phép biến đổi cơ bản như thay đổi kích thước và chuẩn hóa. Trong PyTorch, để thay đổi kích thước hình ảnh, chúng ta sử dụng lớp `Resize` từ `torchvision.transforms`, để chuẩn hóa hình ảnh chúng ta sử dụng lớp `Normalize`.

[10]:
```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

train_dataset = ImageDataset(df_train, transform=transform)
test_dataset = ImageDataset(df_test, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

print(f"Training samples: {len(train_dataset)}, Testing samples:␣
 ↪{len(test_dataset)}")
```

```
Training samples: 2021, Testing samples: 506
```

8 Trực quan hóa hình ảnh mẫu

Hiển thị một vài hình ảnh mẫu từ tập dữ liệu.

[11]:
```
def visualize_sample(dataset, num_samples=5):
    plt.figure(figsize=(15, 5))
    for i in range(num_samples):
        image, label = dataset[i]
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(image.permute(1, 2, 0))
        plt.title(f"Label: {index_to_label[label]}")
        plt.axis('off')
    plt.show()

org_dataset = ImageDataset(df)
visualize_sample(org_dataset)
```

9 Xây dựng mô hình ResNet tùy chỉnh

Trong phần này, chúng ta xây dựng mô hình phân loại hình ảnh sử dụng kiến trúc `ResNet-18`, một trong những mô hình mạng nơ-ron sâu (Deep Neural Network) có hiệu suất cao, đã được huấn luyện trước trên tập dữ liệu ImageNet.

Mô hình gốc của ResNet-18 có đầu ra là 1000 lớp tương ứng với các lớp trong tập dữ liệu ImageNet. Tuy nhiên, để giải quyết bài toán phân loại rác thải, chúng ta cần thay đổi mô hình này sao cho nó phù hợp với số lượng lớp trong tập dữ liệu của mình, cụ thể là 6 lớp cho các loại rác thải khác nhau: thủy tinh, kim loại, nhựa, bìa cứng, giấy và rác thải.

Chúng ta thực hiện các bước điều chỉnh mô hình ResNet-18 như sau:

1. **Loại bỏ lớp phân loại cuối cùng**: Mô hình gốc của ResNet-18 có một lớp phân loại cuối cùng với 1000 đầu ra. Để phù hợp với số lớp phân loại của chúng ta (6 lớp), chúng ta loại bỏ lớp này.
2. **Thêm các lớp Fully Connected (FC)**: Sau khi loại bỏ lớp phân loại cuối, chúng ta thêm một lớp Fully Connected mới để giảm kích thước đặc trưng xuống còn 64, tiếp theo là một lớp `BatchNorm` giúp chuẩn hóa dữ liệu qua các lớp và một lớp dropout để tránh overfitting.
3. **Lớp phân loại cuối cùng**: Lớp cuối cùng là một lớp FC với 6 đầu ra, mỗi đầu ra tương ứng với một lớp phân loại rác thải. Kết quả của lớp này được xử lý qua hàm `Softmax` để cho xác suất phân loại.

[12]:
```
class CustomResNet18(nn.Module):
    def __init__(self, num_classes):
        super(CustomResNet18, self).__init__()

        self.resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

        self.resnet = nn.Sequential(*list(self.resnet.children())[:-1])

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(512, 64)
        self.bn1 = nn.BatchNorm1d(64)
        self.dropout = nn.Dropout(0.1)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.resnet(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.bn1(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return nn.Softmax(dim=1)(x)
```

10 Huấn luyện và đánh giá mô hình

Sau khi xây dựng mô hình phân loại hình ảnh, bước tiếp theo là huấn luyện mô hình với tập dữ liệu huấn luyện và đánh giá hiệu quả của mô hình thông qua độ chính xác trên tập đánh giá. Chúng ta xây dựng các vòng lặp huấn luyện (training), đánh giá (validation) và theo dõi hiệu suất mô hình.

[13]:
```
def train_model(model, train_loader, test_loader, criterion, optimizer,␣
↪num_epochs=10, device=None):
    history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': []}
    best_val_acc = 0.0
    best_model_wts = None

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0
        train_correct = 0

        # Training
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
```

```
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * images.size(0)
            _, preds = torch.max(outputs, 1)
            train_correct += torch.sum(preds == labels.data)

        train_loss /= len(train_loader.dataset)
        train_accuracy = train_correct.double() / len(train_loader.dataset)

        # Validation
        model.eval()
        val_loss = 0.0
        val_correct = 0

        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)

                val_loss += loss.item() * images.size(0)
                _, preds = torch.max(outputs, 1)
                val_correct += torch.sum(preds == labels.data)

        val_loss /= len(test_loader.dataset)
        val_accuracy = val_correct.double() / len(test_loader.dataset)

        if val_accuracy > best_val_acc:
            best_val_acc = val_accuracy
            best_model_wts = model.state_dict()

        # Store history
        history['train_loss'].append(train_loss)
        history['val_loss'].append(val_loss)
        history['train_acc'].append(train_accuracy.item())
        history['val_acc'].append(val_accuracy.item())

        print(f"Epoch {epoch+1}/{num_epochs} -> Train Loss: {train_loss:.4f},␣
↪Train Acc: {train_accuracy:.4f}, Val Loss: {val_loss:.4f}, Val Acc:␣
↪{val_accuracy:.4f}")

    # Load the best model weights
    model.load_state_dict(best_model_wts)

    return model, history
```

Xây dựng hàm vẽ biểu đồ trực quan hóa lịch sử huấn luyện của mô hình, bao gồm mất mát (loss) và độ chính xác (accuracy) trong suốt quá trình huấn luyện và đánh giá.

[14]:
```
def plot_training_history(history):
    epochs = range(1, len(history['train_loss']) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history['train_loss'], label='Train Loss')
    plt.plot(epochs, history['val_loss'], label='Val Loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history['train_acc'], label='Train Accuracy')
    plt.plot(epochs, history['val_acc'], label='Val Accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.show()
```

Để huấn luyện mô hình, chúng ta sử dụng phương pháp tối ưu hóa `Adam` và hàm mất mát `CrossEntropyLoss`, đay là lựa chọn phổ biến cho bài toán phân loại nhiều lớp.

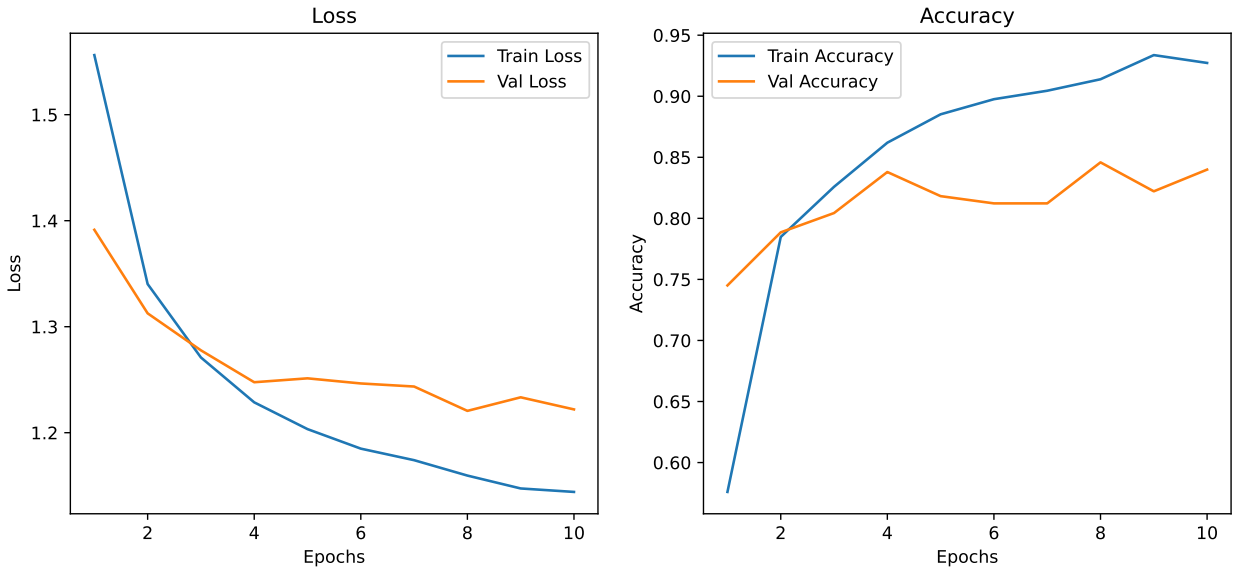
[15]:
```
model = CustomResNet18(num_classes=len(classes_list))

for param in model.resnet.parameters():
    param.requires_grad = False

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

model, history = train_model(model, train_loader, test_loader, criterion,␣
 ↪optimizer, num_epochs=10, device=device)

torch.save(model.state_dict(), 'custom_resnet_garbage.pth')
```

```
Epoch 1/10 -> Train Loss: 1.5716, Train Acc: 0.5933, Val Loss: 1.3958, Val Acc:
0.7609
Epoch 2/10 -> Train Loss: 1.3453, Train Acc: 0.8001, Val Loss: 1.3044, Val Acc:
0.8004
Epoch 3/10 -> Train Loss: 1.2697, Train Acc: 0.8347, Val Loss: 1.2796, Val Acc:
```

```
0.8043
Epoch 4/10 -> Train Loss: 1.2307, Train Acc: 0.8605, Val Loss: 1.2536, Val Acc:
0.8379
Epoch 5/10 -> Train Loss: 1.2026, Train Acc: 0.8837, Val Loss: 1.2385, Val Acc:
0.8340
Epoch 6/10 -> Train Loss: 1.1896, Train Acc: 0.8946, Val Loss: 1.2424, Val Acc:
0.8103
Epoch 7/10 -> Train Loss: 1.1725, Train Acc: 0.9065, Val Loss: 1.2346, Val Acc:
0.8360
Epoch 8/10 -> Train Loss: 1.1577, Train Acc: 0.9189, Val Loss: 1.2302, Val Acc:
0.8419
Epoch 9/10 -> Train Loss: 1.1503, Train Acc: 0.9238, Val Loss: 1.2287, Val Acc:
0.8360
Epoch 10/10 -> Train Loss: 1.1393, Train Acc: 0.9327, Val Loss: 1.2362, Val Acc:
0.8320
```

Trực quan hóa lịch sử huấn luyện:

[16]:
```
plot_training_history(history)
```



11 Trực quan hóa dự đoán

[17]:
```
def visualize_images_with_predictions(image_paths, true_labels, model,␣
↪index_to_label, device, num_samples=5):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])

    plt.figure(figsize=(15, 10))
    for i in range(num_samples):
        image_path = image_paths[i]
        true_label = true_labels[i]

        image_org = Image.open(image_path).convert('RGB')
        image = transform(image_org).unsqueeze(0).to(device)

        model.eval()
        with torch.no_grad():
            outputs = model(image)
            _, predicted = torch.max(outputs, 1)

        predicted_label = index_to_label[predicted.item()]

        plt.subplot(1, num_samples, i + 1)
        plt.imshow(image_org)
        plt.title(f"True: {true_label}\nPred: {predicted_label}")
        plt.axis('off')

    plt.show()
```

Để đánh giá chất lượng dự đoán, chúng ta sẽ trực quan hóa các hình ảnh cùng với nhãn thực và nhãn dự đoán từ mô hình đã được huấn luyện.

[18]:
```
image_paths = df_test['img_path'].iloc[:5].tolist()
true_labels = df_test['label'].iloc[:5].tolist()
visualize_images_with_predictions(image_paths, true_labels, model,␣
↪index_to_label, device, num_samples=5)
```

12 Kết luận

Trong đồ án này, chúng ta đã xây dựng và triển khai một mô hình học sâu để phân loại các loại rác thải sử dụng mô hình `ResNet-18` với những điều chỉnh thích hợp. Quá trình thực hiện bao gồm nhiều bước quan trọng, từ việc chuẩn bị và tiền xử lý dữ liệu, tạo ánh xạ giữa nhãn và chỉ số, đến việc xây dựng và huấn luyện mô hình.

Đầu tiên, chúng ta đã tiến hành tiền xử lý dữ liệu, bao gồm việc thay đổi kích thước và chuẩn hóa hình ảnh, giúp dữ liệu đầu vào phù hợp với yêu cầu của mô hình `ResNet-18`. Sau đó, chúng ta đã xây dựng lớp Dataset tùy chỉnh để dễ dàng xử lý và cung cấp dữ liệu cho mô hình.

Mô hình `ResNet-18` đã được triển khai và huấn luyện trên tập dữ liệu phân loại rác thải. Sau khi huấn luyện, mô hình đạt được độ chính xác khá cao trên tập kiểm tra (hơn 80%). Cuối cùng, chúng ta đã thu được một mô hình phân loại hình ảnh rác thải có khả năng phân loại chính xác và có thể ứng dụng trong thực tế, đặc biệt là trong các hệ thống quản lý rác thải thông minh.

Bên cạnh đó, vẫn còn nhiều hướng phát triển và cải tiến cho mô hình, như việc tăng cường tập dữ liệu, thử nghiệm với các mô hình phức tạp hơn hoặc tinh chỉnh các tham số của mô hình để đạt được kết quả tốt hơn. Hy vọng rằng đồ án này có thể hữu ích cho các nghiên cứu và ứng dụng trong việc tự động phân loại rác thải trong các hệ thống xử lý rác thải hiện đại.