Mateja Aristan

Flower Recognition

Final project report

Advanced Machine Learning Methods

2020/2021

Problem description

For the project, I decided to use a flowers dataset consisting of images of flowers with 5 possible class labels. The dataset contains 4323 images of flowers. The images are divided into five classes: chamomile, tulip, rose, sunflower, dandelion. For each class, there are about 800-1000 photos. According to Kaggle, all images were collected from Data Flicr, Google images, Yandex images.

The goal of this project is to create a flower recognition model for identifying types of flowers from images. In the project, I will be using the transfer learning method where I will take a pre-trained ImageNet model - use it to extract features and train a new layer on top for our own task of classifying images of flowers.

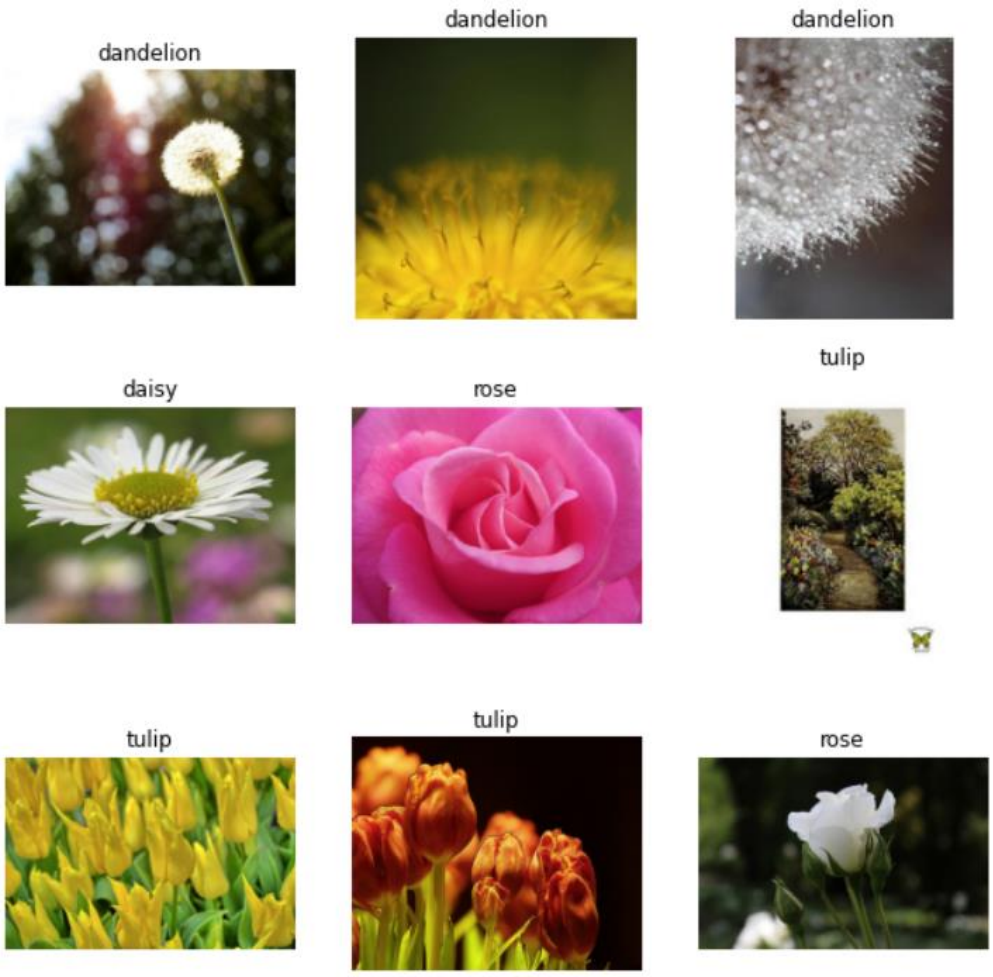

Figure 1 Example of flower dataset images

Methodology

Like previously mentioned, for the project pre-trained ImageNet was used. ImageNet is used to train deep neural networks using an architecture called convolutional layers. In this project it will be used to train a network that should classify flowers images into flower categories. Below are some of the steps from the project.

For the first step in the project, the libraries were imported. Here, torchvision.models will be used to download pre-trained networks.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import random
import PIL
import torch
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from torchvision import transforms, utils, models
from torch import nn
from sklearn.metrics import accuracy_score,confusion_matrix,
classification_report
from mlxtend.plotting import plot_confusion_matrix
```

Next, data directory was defined and subfolders with flower images were listed:

```
data_dir = '../input/flowers-recognition/flowers/'
folders = os.listdir(data_dir)
print(folders)
```

To check the distribution of images per classes, I created a bar chart:

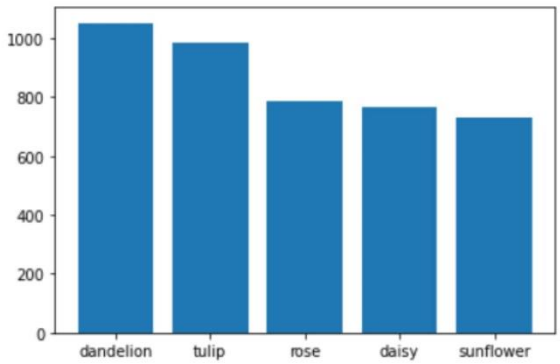

Figure 2 Distribution of flower images

Before further work I wanted to see the images I was working with. I checked 9 random images from the dataset and the 3 examples can be seen below.

Figure 3 Visualization of the dataset

By checking the dimensions and formats of the images, I noticed that picture dimensions are not uniformed so later in the project they will be resized before splitting the dataset.

I have then decided to split dataset into:

```
train_ratio = 0.70
validation_ratio = 0.20
test_ratio = 0.10
```

By counting the number of images per class I have found that there are 1052 dandelions, 984 tulips, 784 roses, 764 daisies and 733 sunflowers.
To overcome the problem of overfitting the train images were transformed. The images were resized to 224x224 because most of the pretrained images require that input. The normalization had to be done for each color channel separately, the means are [0.485, 0.456, 0.406] and the standard deviations are [0.229, 0.224, 0.225] which was done like in the examples from the course.

```
transform_train = transforms.Compose([
      transforms.Resize(224),
      transforms.CenterCrop(224),
      transforms.RandomHorizontalFlip(p=0.5),
      transforms.RandomVerticalFlip(p=0.5),
      transforms.ColorJitter(brightness=0.15, contrast=0.2,
      saturation=0.1),
      transforms.ToTensor(),
      transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])

transform_test = transforms.Compose([transforms.Resize(224),
                 transforms.CenterCrop(224),
```

```
transforms.ToTensor(),
transforms.Normalize([0.485, 0.456, 0.406],
[0.229, 0.224, 0.225])])
```

Preparation of data for training with data loaders:

1. Defined transforms for the training, valid and testing data

```
train_dataset = Flowers(train,transform=transform_train)
valid_dataset = Flowers(validation,transform=transform_test)
test_dataset = Flowers(test,transform=transform_test)

train_dataloader = torch.utils.data.DataLoader(train_dataset,batch_size=72,
shuffle=True)
valid_dataloader = torch.utils.data.DataLoader(valid_dataset,batch_size=72)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=72)
```

To check the transformation, I have checked the images to see if they were resiyed and flipped.

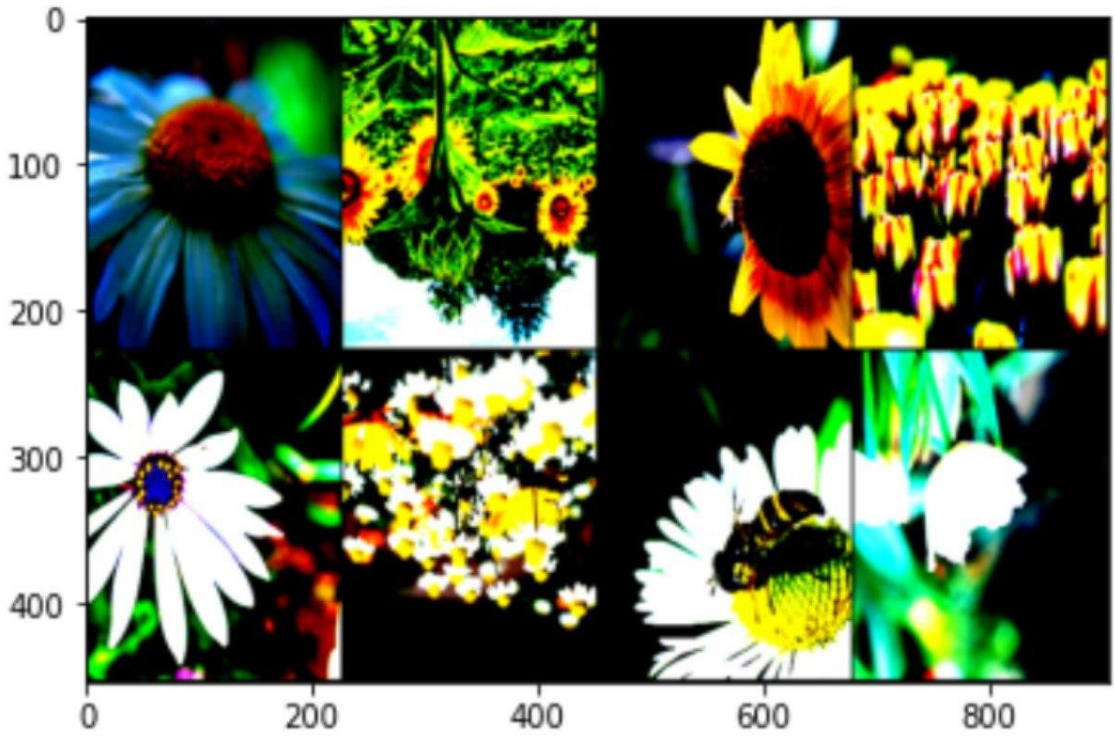

Figure 4 Transformed images

I have downloaded the pretrained model

```
model=models.googlenet(pretrained=True)
```

Freeze parameters so we don't backprop through them

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

Changing last layer:

```
from collections import OrderedDict
classifier = nn.Sequential(OrderedDict([
                          ('fc1', nn.Linear(1024, 512)),
                          ('relu', nn.ReLU()),
                          ('fc2', nn.Linear(512, 5)),]))

model.fc = classifier
```

Defining Criterion and optimizer and transferring model to device

```
criterion = nn.CrossEntropyLoss()
model.to(device)
optimizer = torch.optim.SGD(model.fc.parameters(), lr=0.001)
```

Experiments and evaluation

The training of the model is performed with 100 ephochs and the results for last epoch is as follows for train loss, test loss and test accuracy:

Train loss: 0.608..

Validation loss: 0.533..

Validation accuracy: 0.844

```
epochs = 100
running_loss = 0

train_losses, valid_losses = [], []
for epoch in range(epochs):
    for inputs, labels in train_dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        logps = model.forward(inputs)
        loss = criterion(logps, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
```

```
    valid_loss = 0
    accuracy = 0
    with torch.no_grad():
        model.eval()
        for inputs, labels in valid_dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            logps = model.forward(inputs)
            batch_loss = criterion(logps, labels)
            valid_loss += batch_loss.item()
            ps = torch.exp(logps)
            top_p, top_class = ps.topk(1, dim=1)
            equals = top_class == labels.view(*top_class.shape)
            accuracy += torch.mean(equals.type(torch.FloatTensor)).item()

    train_losses.append(running_loss/len(train_dataloader))
    valid_losses.append(valid_loss/len(valid_dataloader))
    print(f"Epoch {epoch+1}/{epochs}.. "
          f"Train loss: {train_losses[-1]:.3f}.. "
          f"Validation loss: {valid_losses[-1]:.3f}.. "
          f"Validation accuracy: {accuracy/len(valid_dataloader):.3f}")
    running_loss = 0
    model.train()
```

Evaluating the model accuracy on test set with the accuracy of 0,863:

```
acc, predictions, actuals = evaluate_model(test_dataloader, model)
print('Test Accuracy: %.3f' % acc)
```

```
print(classification_report(actuals, predictions, target_names=label_enc.classes_))
```

```
              precision    recall  f1-score   support

       daisy       0.91      0.90      0.90        77
   dandelion       0.93      0.89      0.91       105
        rose       0.86      0.85      0.85        78
   sunflower       0.87      0.82      0.85        73
       tulip       0.77      0.86      0.81        99

    accuracy                           0.86       432
   macro avg       0.87      0.86      0.86       432
weighted avg       0.87      0.86      0.86       432
```

The confution matrix is presented below:

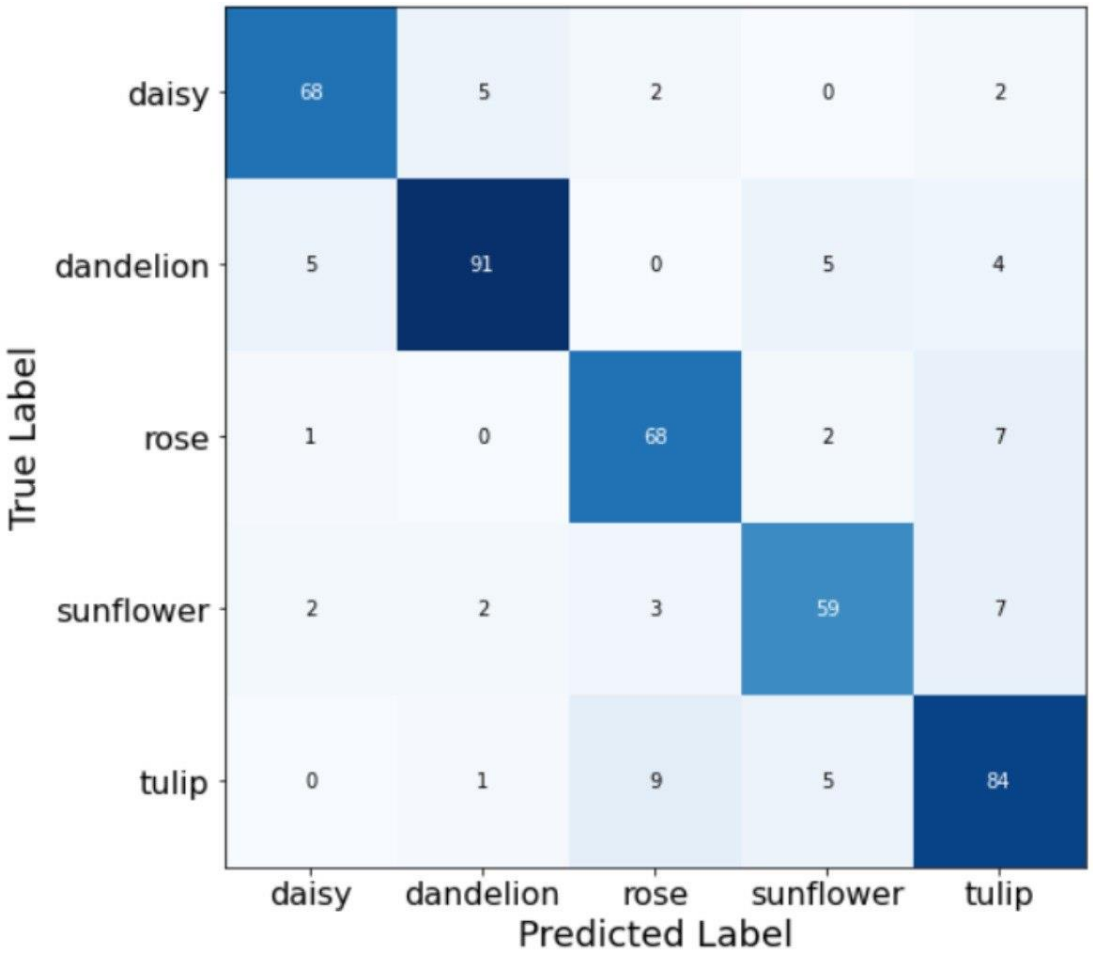

Figure 5 Confution Matrix

Conclusion

To prepare the dataset for training, I had to resize and uniform the images. After resizing, samples were divided into three parts for training, test and validation. Except for normalising pixel value with 224, center crop, random vertical flip, color jitter and horizontal flip were applied to input data. The batch size I used was 72, which was quite common and friendly to GPU's parallel computing. The epochs I choose is 100. By using pre-trained ImageNet model, the accuracy reached 0.844, train loss 0.608 And validation loss: 0.533.