Flower Name Classification

Jean-Sebastien Roger

October 25, 2020

Abstract

In this experiment we examined the task of classifying pictures of flowers; we attempted to classify each flower by its name. The set of images loaded for this analysis are of various flowers, each with its associated label. For each flower we have multiple examples of the image, at different angles and under different lighting conditions. Our goal of this experiment was to utilize a common pre-trained model built from using an 18-layer residual network (ResNet-18) which has been tuned to recognize and classify images. We trained this model on our dataset of flowers and then proceeded to build a visualizer which allows us to present random flower selections along side their classification and prediction.

1 Introduction

The dataset utilized for this experiment was in two folders (a 'train' folder and a 'validation' folder) filled with 102 sub-directories each a specific genus of flower with at least 5 example JPG images. Each image was of a genus of plant specific to the folder it was placed in, and each folder was named according to the number the genus name was assigned. Figure 1 shows a sample of the images that we have in our training set.

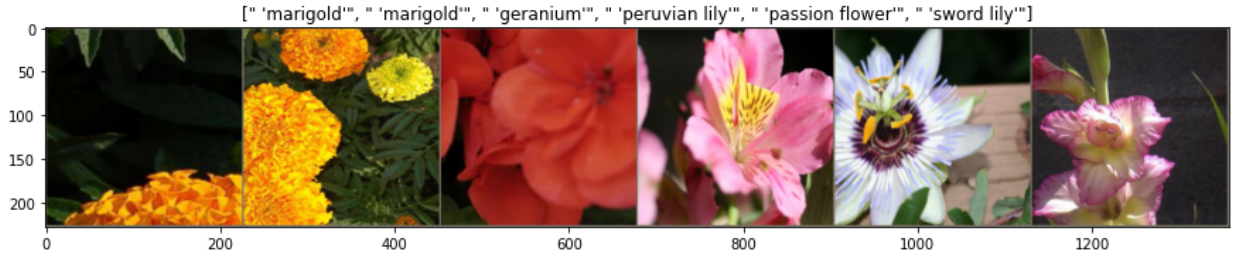

Figure 1: Shows a random sample of cropped images from our training dataset. In total, we had 6,552 training images and 818 validation images.

2 Theory

In lab 5, we are interested in identifying one genus of flower versus another. Our image classification task is now much more challenging than that of lab 4 because rather than

identify borders of an image based on the gradient in brightness, we now have to identify multiple key features within an image that will allow us to associate an image to a label. In order to match the increased complexity of the problem, we now use more advanced classification techniques built upon neural network models. The model we are using for our experiment is the ResNet-18. This model has been pre-trained, and using the pytorch library and torchvision functions, we are able to further train this model on our training set images and produce a model which can be used to predict genus labels of flowers in our validation set. Finally, using this newly trained model, we can construct a visualizer which can output the JPG image that the model is given as input, the prediction, and the true classification.

3 Procedures

Here we are establishing a standard way with which we will randomly crop each input image, convert it into a tensor image, and then normalize the resulting tensor so the it can be interpreted by the model easier and the model can converge towards a prediction much faster.

```
# Normalization measurements
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])

# Establishing transformations
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),
    'valid': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean, std)
    ]),
}
```

After reading in libraries and setting up the transformations which will crop and normalize our input images, we can run the below lines of code in order to load in the data and apply our normalization and labeling for further classification.

```
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                      data_transforms[x]) for x in ['train', 'valid']
                  }
```

```
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
                  batch_size=16,
                  shuffle=True,
                  num_workers=4) for x in ['train', 'valid']
                }

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'valid']}

class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Now that we have the loops created which will load random pictures and their associated labels from our imageset, we are able to define a function (code produced in the Appendix) which we can use to quickly visualize the cropped images that we will be training our model on. Figure 2 is a random selection of these images.

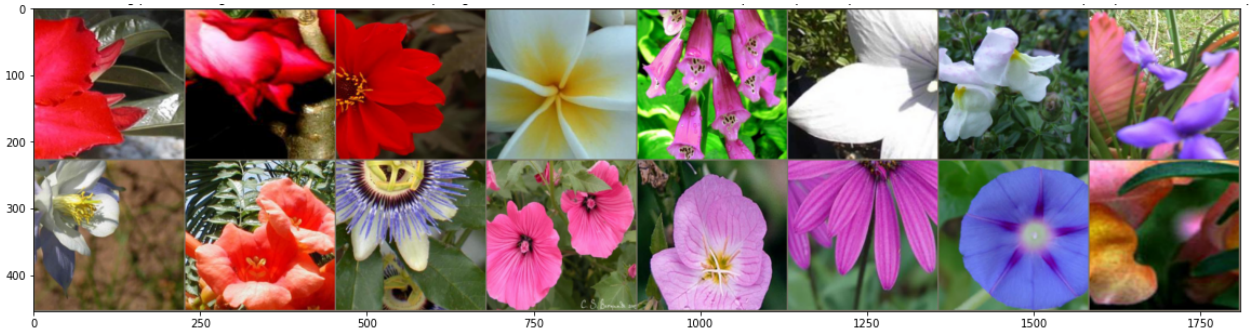

Figure 2: Above are 16 randomly selected images that come from the training set. These images have already been cropped and transformed into tensors which are then normalized.

4 Analysis

In this section we will review the model training, we have moved the package import section and various other code chunks to the Appendix.

With our images loaded, cropped, and normalized, we are now prepared to download our ResNet-18 model and train it to make predictions on our prepped dataset. Here we will briefly introduce a key part of the code and discuss calling the model and the training process.

```
model = models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 102)
model = train_model(model, num_epochs=6)
```

The above chunk of code downloads the pre-trained ResNet-18 model and then performs a linear transformation to the features of the model before training. Once we have loaded the model, we then train the model on our dataset of flower images. The full training definition has been included in the Appendix, however below we show a snippet of the function for the loss criteria used to tune the model, the method of optimization (stochastic gradient descent), and the step function which sets a training schedule using the learning rate based on our optimization method.

```
# Specifying our loss function
criterion = nn.CrossEntropyLoss()

# Setting stochastic gradient descent with a learning rate of .001
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Scheduling step function based on descent method and learning rate
scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

It is important to note that although our training is set to go through 6 epochs, after 3 epochs there is a significant decrease in the additional lift in accuracy with each new iteration. So in order to speed up training time, we could decrease the number of epochs to 3 and we would still have a very accurate classification model.

Now that our model has been trained and validated, we can proceed to how we wish to present the image along side our prediction and their respective class to the audience. Our visualization function is presented in the code chunk below and Figure 3 shows a sample of the output.

```
def visualize_model(model, num_images=16):
    model.eval()
    index = 0
    for i, (inputs, labels) in enumerate(dataloaders['valid']):
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = model(inputs)
        preds = torch.max(outputs, 1)[1]

        for j in range(inputs.size()[0]):
            index += 1
            title1 = 'predicted: ' +
            dataset_labels[int(class_names[preds[j]])-1] +
            '      class: ' +
            dataset_labels[int(class_names[labels[j]])-1]
            imshow(inputs.cpu().data[j], title1)

            if index == num_images:
                return
```

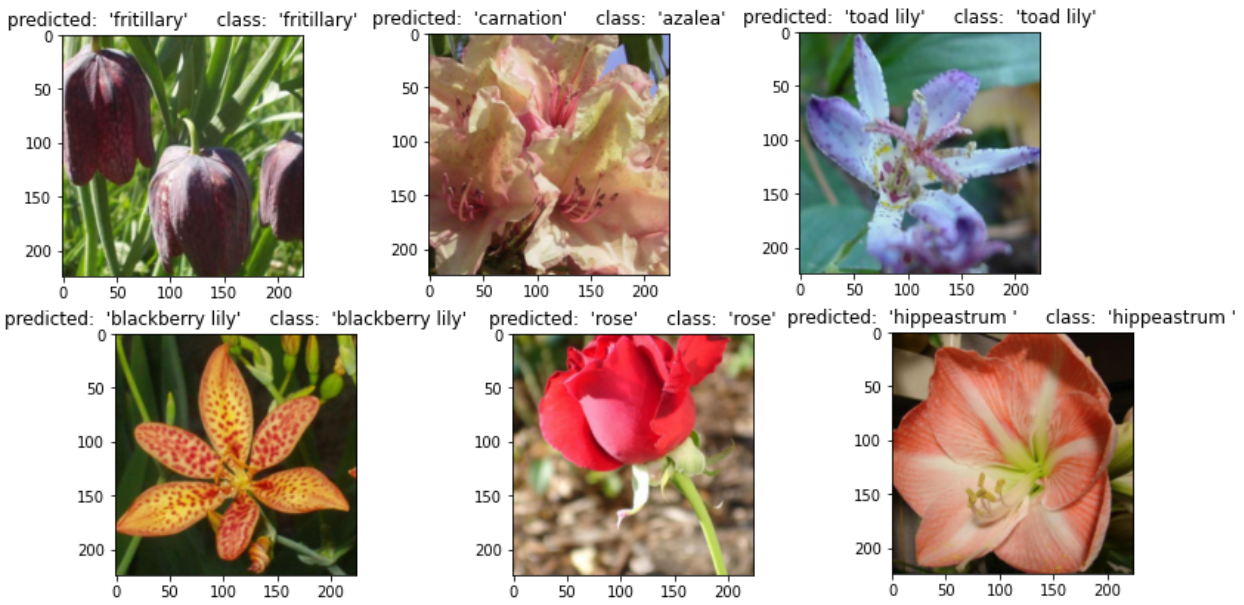

Figure 3: Validation images alongside a given prediction and true class.

As we can see in Figure 3, we are able to read in a set of PNG images from a folder and output a prediction with a single function call. Below we also show how we are able to read a single image from online and, after performing a similar tensor transformation in order to normalize the image, we can run our model on this single image and output a prediction.

```
imagePath =
'https://images-na.ssl-images-amazon.com/images/I/51dZp-%2B4W9L._AC_.jpg'
image = io.imread(imagePath)
plt.imshow(image);
```

Figure 4: .

Figure 4 is the single image we have taken from online and now we can apply our model to the image and output a prediction using the following code:

```
img = apply_transforms(image).\
    clone().detach().requires_grad_(True).to(device)

outputs = model(img)

preds = torch.max(outputs, 1)[1]

print('predicted: ' + dataset_labels[int(class_names[preds])-1])

predicted:  'pink primrose'
```

5 Conclusions

This lab provided some very valuable lessons on how to approach an image classification problem more advanced than that in Lab 4. In this lab we have taken our previous lab, dealing with with classifying rice seed image as either 'proper' or 'broken' and expanded on the task of image classification. Here, we have not only built a classifier that will take an image, or set of images, and output a prediction, but we have increased the difficulty of our prediction engine. From a simple threshold calculation up to a complete pre-trained residual network with additional tuning on our dataset. This scales up our prediction capabilities, which now will output a classification of flower genus after taking into account multiple characteristics within the image.

From this lab, we are able to take a pre-trained model and re-tune it to use our labels and image sets. Using this procedure, we can see the power of residual network models and their ability to identify key characteristics within an image and associate them to a label. The next steps in this lab would be to check our classifications. Although the model looks accurate from our random sampling of predictions, building out some sort of confusion matrix for correct predictions vs incorrect predictions on the validation set would give us a better understanding of the performance of our model. Using those results, we could then look to break down the predictions and see what images the model is mis-classifying.

6 Bibliography

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition [PDF]. Las Vegas: Institute of Electrical and Electronic Engineers.

"Torchvision.models." Torchvision.models - PyTorch 1.6.0 Documentation

7 Appendix

Package import

```
!pip install git+https://github.com/williamedwardhahn/mpcr
from mpcr import *

!pip install flashtorch
!pip install barbar

from flashtorch.utils import apply_transforms
from flashtorch.saliency import Backprop

import itertools
```

Image presentation function

```
def imshow(inp, title = " "):
    fig, ax = plt.subplots()
    inp = inp.numpy().transpose((1, 2, 0))
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    ax.imshow(inp)
    plt.title(title, loc='center')
    fig.set_size_inches(20, 20)
    plt.show()
```

Full training function

```
def train_model(model, num_epochs=25):

    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
    scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

    for epoch in range(num_epochs):

        print('Epoch: ',epoch+1,'/',num_epochs)

        ###Train
        model.train()
        running_corrects = 0
        for inputs, labels in Bar(dataloaders['train']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()
```

```
            outputs = model(inputs)

            preds = torch.max(outputs, 1)[1]
            running_corrects += torch.sum(preds == labels.data)

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

        print("Train ", 'Acc: {:.2f}'.\
        format(running_corrects.double()/dataset_sizes['train']))

        scheduler.step()

        ###Val
        model.eval()
        running_corrects = 0
        for inputs, labels in Bar(dataloaders['valid']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            preds = torch.max(outputs, 1)[1]
            running_corrects += torch.sum(preds == labels.data)

        print("Valid ", 'Acc: {:.2f}'.\
        format(running_corrects.double()/dataset_sizes['valid']))
        print("##########################")
    return model
```

Final prediction visualizer

```
def visualize_model(model, num_images=16):
    model.eval()
    index = 0
    for i, (inputs, labels) in enumerate(dataloaders['valid']):
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)

        preds = torch.max(outputs, 1)[1]

        for j in range(inputs.size()[0]):
            index += 1
```

```
title1 = 'predicted: ' +
dataset_labels[int(class_names[preds[j]])-1] +
'      class: ' +
dataset_labels[int(class_names[labels[j]])-1]

imshow(inputs.cpu().data[j], title1)

if index == num_images:
    return
```