Biostatistics (STA5195) Lab 5 Report

Flower Classification

Christopher Rutherford

October 19, 2020

Abstract

In this paper, we demonstrate the use of a neural network for classifying images of various species of flowers. More specifically, we will use the pretrained ResNet18 model [1] to assist us in training on the images we provide. This lab will also demonstrate how to perform transformations and create our train/test splits directly from Google Drive.

1 Introduction

Neural networks have proven themselves to be extremely powerful and useful deep learning algorithms. In particular, convolutional neural networks (CNNs) have found their way into being able to classify images and detect certain objects within those images [2]. To demonstrate the predictive power of the CNN, we will use images of approximately 100 different species of flowers.

2 Theory

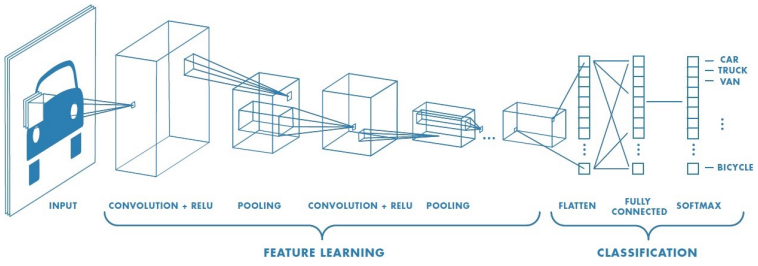

Figure 1: CNN architecture [5]

CNNs have a few main components in their inner workings: convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for extracting features from the images through the use of filters (also called kernels). These kernels "slide" over the image to extract these features. Typically, these features include some sort

of edge detection, or some other characteristic detected by these kernels after transforming the images. Pooling layers are responsible for reducing the dimensionality of the data. This helps reduce processing time and power by storing only the most critical information needed to classify the image. The two most common types of pooling include max pooling and average pooling. Like convolution, pooling slides over the image to extract features, but can incorporate stride as well, which decides how many pixels to shift over at a time.

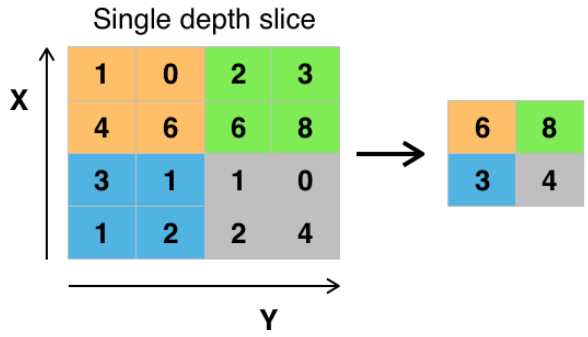

Figure 2: Max pooling with a 2x2 filter and stride of 2 [6]

At this point, we get to the fully connected layers, named so because each node/neuron in these layers is connected to each neuron in the preceding and succeeding layers. These layers are responsible for receiving the transformed data and deciding which class the image belongs to. When training the network, this also includes a loss layer, such as softmax or cross-entropy, which penalizes misclassifications.

3 Procedures

As usual, we begin by installing the core libraries we'll be using throughout this lab:

```
!pip install git+https://github.com/williamedwardhahn/mpcr
!pip install flashtorch
!pip install barbar
from mpcr import *
from flashtorch.utils import apply_transforms
from flashtorch.saliency import Backprop
import itertools
```

After importing and installing the libraries we need, we can mount our Google Drive, which is where we'll be downloading our dataset to:

```
drive.mount('/content/drive')
```

Now, we need to direct Python to our desired download directory. We'll create a folder called `flower_data` and set that to our current working directory:

```
os.chdir('/content/drive/My Drive/Data/flower_data')
```

And now we download the files. The first file contains the labels for each flower, and the second file contains the images we'll be classifying. The `-N` flag prevents the file from being downloaded again if we already have the file that is requested.

```
!wget -N https://gist.githubusercontent.com/JosephKJ/94
   c7728ed1a8e0cd87fe6a029769cde1/raw/403325
   f5110cb0f3099734c5edb9f457539c77e9/Oxford-102
   _Flower_dataset_labels.txt
!wget -N https://s3.amazonaws.com/content.udacity-data.com/courses/
   nd188/flower_data.zip
!unzip 'flower_data.zip'
```

We'll grab the labels from the text file and save them in a new variable:

```
dataset_labels = pd.read_csv('Oxford-102_Flower_dataset_labels.txt',
   header=None)[0]
```

For convenience, we'll also save our current working directory as a string for later use.

```
data_dir = '/content/drive/My Drive/Data/flower_data/flower_data/'
```

By having our directory as a string, we can simply call the `data_dir` variable whenever we want to refer to the directory. We'll need this for when we create our training and validation sets.

Now, we can get started on the fun part: training our model and making predictions to see how well it can predict!

4 Analysis

Now we begin transforming our data. This helps establish some uniformity across the different flower images, as they will now all be the same dimensions.

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

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

Now we can create our training and validation sets, along with instructing our neural network to run on the GPU.

```
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
    data_transforms[x]) for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
   batch_size=16, shuffle=True, num_workers=4) for x in ['train', '
   val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val'
   ]}
class_names = image_datasets['train'].classes
device = torch.device("cuda:0" if torch.cuda.is_available() else "
   cpu")
```

Verifying the data was split correctly:

```
dataset_sizes
```

```
{'train': 6552, 'val': 818}
```

We create a small helper function to plot the images to make sure the data is in working order - the images and labels need to be configured properly.

```
#function to plot set of images from the data
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(5, 5)
    plt.show()
```

```
#plot some images
inputs, classes = next(iter(dataloaders['train']))
inputs = inputs[:4]
classes = classes[:4]
out = torchvision.utils.make_grid(inputs)
imshow(out, title=[dataset_labels[int(class_names[x])-1] for x in
   classes])
```

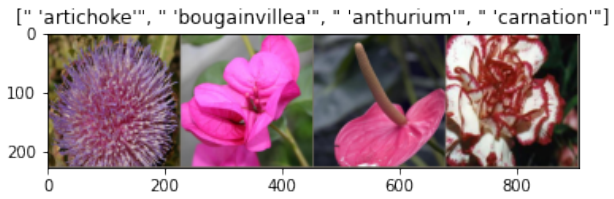

Figure 3: Sample of flowers from the training data

Since this can take some time due to the amount of images we have, we'll add some progress bars to get a better idea of our model's training progress.

```
from barbar import Bar
```

And now our function for training the model with its hyperparameter configuration:

```
#function for actually training the model
def train_model(model, num_epochs=25):
    model = model.to(device) #train on GPU if available
    criterion = nn.CrossEntropyLoss() #use cross entropy as our loss
    function
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum
   =0.9) #stochastic gradient descent optimizer
    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['val']):
            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['val']))
        print("########################")
    return model
```

We'll bring in the pretrained ResNet18 model to help out our own model:

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

And now we can train the model.

```
model = train_model(model, num_epochs=3)
```

```
Epoch:  1 / 3
6552/6552: [==============================>] - ETA 0.6s
Train  Acc: 0.41
818/818: [==============================>] - ETA 0.6s
Valid  Acc: 0.78
########################
Epoch:  2 / 3
6552/6552: [==============================>] - ETA 0.6s
Train  Acc: 0.76
818/818: [==============================>] - ETA 0.7s
Valid  Acc: 0.90
########################
Epoch:  3 / 3
6552/6552: [==============================>] - ETA 0.6s
Train  Acc: 0.85
818/818: [==============================>] - ETA 0.6s
Valid  Acc: 0.94
########################
```

We make another helper function that prints out an image along with its true and predicted classes.

```
#function for printing images and labels of flowers
def visualize_model(model, num_images=16):
    model.eval()
    index = 0
    for i, (inputs, labels) in enumerate(dataloaders['val']):
        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
```

```
visualize_model(model)
```

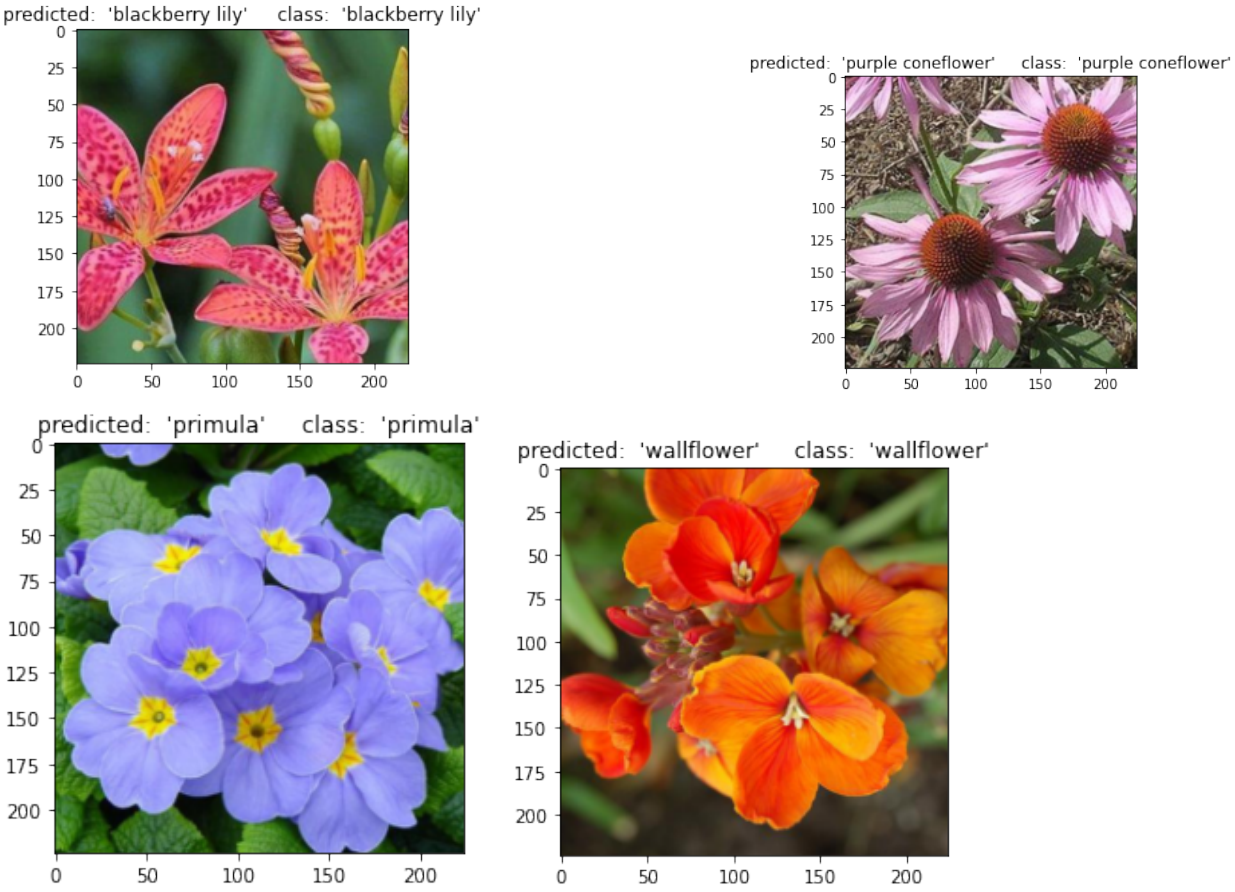

Figure 4: Predictions and true classes for four flowers

We can also test out how well this model can predict on a new, unseen image from the internet.

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

Before running this image through the model, we'll need to perform the necessary transformations so that it resembles the other images fed through the model:

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

```
#make prediction on image
outputs = model(img)
preds = torch.max(outputs, 1)[1]
```

And finally, we can print out the model's prediction:

```
print('predicted: ' + dataset_labels[int(class_names[preds])-1])# print out the model's prediction
```

Figure 5: Sample flower image

`predicted: 'pink primrose'`
A quick google search [3] confirms that this prediction is correct. Not bad.

5 Conclusions

Overall, our neural network seemed to perform quite well on this data set. After only 3 epochs, it attained 85% accuracy on the training set and 94% on the validation set. Given more time to train over more epochs, this model could likely easily attain even better predictive power.

References

[1] ResNet | Pytorch: https://pytorch.org/hub/pytorch_vision_resnet/.

[2] *Convolutional neural network*, available at https://en.wikipedia.org/wiki/Convolutional_neural_network.

[3] *Oenothera speciosa*, available at https://en.wikipedia.org/wiki/Oenothera_speciosa.

[4] ML Practicum: Image Classification | Machine Learning Practica: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks.

[5] Understanding of Convolutional Neural Network (CNN) — Deep Learning: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[6] By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581