Classifying space rocks

1. Importing packages

In [17]:

```
#!/usr/local/bin/python3.7


# classifying_space_rocks.py


## Setting the current working directory automatically
import os
project_path = os.getcwd() # getting the path leading to the current working directory
os.getcwd() # printing the path leading to the current working directory
os.chdir(project_path) # setting the current working directory based on the path leading to the cu
rrent working directory


## Required packages
import matplotlib.pyplot as plt # for plotting data
import numpy as np # for processing large numerical matrices (i.e. images)
import torch # for training and processing deep learning and AI models
from torch import nn, optim
from torch.autograd import Variable
import torch.nn.functional as F
import torchvision # for processing images and doing manipulations like cropping and resizing
from torchvision import datasets, transforms, models
# Python Imaging Library (PIL) for visualizing the images
from PIL import Image

# For ensuring plots are shown inline and with high resolution
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

2. Importing and cleaning data about photos of space rocks

In [18]:

```
# Telling the machine what folder contains the image data
data_dir = './data'

# Function to read the data; crop and resize the images; and then split it into test and train chu
nks
def load_split_train_test(datadir, valid_size = .2):
    # This line of code transforms the images
    train_transforms = transforms.Compose([
                                        transforms.RandomResizedCrop(224),
                                        transforms.Resize(224),
                                        transforms.ToTensor(),
                                        ])

    test_transforms = transforms.Compose([transforms.RandomResizedCrop(224),
                                          transforms.Resize(224),
                                          transforms.ToTensor(),
                                      ])

    train_data = datasets.ImageFolder(datadir, transform=train_transforms)
    test_data = datasets.ImageFolder(datadir, transform=test_transforms)

    num_train = len(train_data)
    indices = list(range(num_train))
    split = int(np.floor(valid_size * num_train))
    np.random.shuffle(indices)
    from torch.utils.data.sampler import SubsetRandomSampler
    train_idx, test_idx = indices[split:], indices[:split]
    train_sampler = SubsetRandomSampler(train_idx)
```

```
    test_sampler = SubsetRandomSampler(test_idx)
    trainloader = torch.utils.data.DataLoader(train_data, sampler=train_sampler, batch_size=16)
    testloader = torch.utils.data.DataLoader(test_data, sampler=test_sampler, batch_size=16)
    return trainloader, testloader

# We're using 20% of data for testing
trainloader, testloader = load_split_train_test(data_dir, .2)
print(trainloader.dataset.classes)
```

```
['Basalt', 'Highland']
```

3. Reading each image and assigning each image with a corresponding rock type

In [19]:

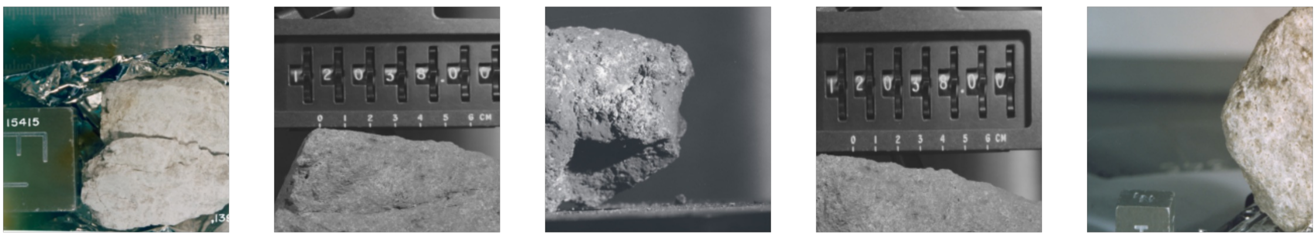
```
# Transform the new image into numbers and resize it
test_transforms = transforms.Compose([transforms.RandomResizedCrop(224),
                                      transforms.Resize(224),
                                      transforms.ToTensor(),
                                    ])

# A function to randomly select a set of images
def get_random_images(num):
    data = datasets.ImageFolder(data_dir, transform=test_transforms)
    classes = data.classes
    indices = list(range(len(data)))
    np.random.shuffle(indices)
    idx = indices[:num]
    from torch.utils.data.sampler import SubsetRandomSampler
    sampler = SubsetRandomSampler(idx)
    loader = torch.utils.data.DataLoader(data, sampler=sampler, batch_size=num)
    dataiter = iter(loader)
    images, labels = dataiter.next()
    return images, labels
```

4. Showing some images loaded into the program

In [20]:

```
# How many images do you want to see? It's set to 5, but you can change the number
images, labels = get_random_images(5)
to_pil = transforms.ToPILImage()
fig=plt.figure(figsize=(20,20))
classes=trainloader.dataset.classes
for ii in range(len(images)):
    image = to_pil(images[ii])
    sub = fig.add_subplot(1, len(images), ii+1)
    plt.axis('off')
    plt.imshow(image)
plt.show()
```



5. Building a neural network to classify space rocks

In [21]:

```
# Determine whether you're using a CPU or a GPU to build the deep learning network.
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
model = models.resnet50(pretrained=True)

# Builds all the neurons
for param in model.parameters():
    param.requires_grad = False

# The parameters of our deep learning model
# (Wire the neurons in an appropriate way (there are thousands of ways to wire neurons))
model.fc = nn.Sequential(nn.Linear(2048, 512),
                                 nn.ReLU(),
                                 nn.Dropout(0.2),
                                 nn.Linear(512, 2),
                                 nn.LogSoftmax(dim=1))
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.003)
model.to(device)
print('done')
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to
/Users/anthony/.cache/torch/checkpoints/resnet50-19c8e357.pth
```

```
done
```

6. Training a neural network to accurately classify space rocks in photos

In [22]:

```python
epochs = 5 # tells the program how many times to search for associations in features (i.e. how man
y times it will pass through all the data)
steps = 0
running_loss = 0
print_every = 5
train_losses, test_losses = [], []

for epoch in range(epochs):
    for inputs, labels in trainloader:

        steps += 1
        print('Training step ', steps)
        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()
        if steps % print_every == 0:
            test_loss = 0
            accuracy = 0
            model.eval()
            with torch.no_grad():
                for inputs, labels in testloader:
                    inputs, labels = inputs.to(device), labels.to(device)
                    logps = model.forward(inputs)
                    batch_loss = criterion(logps, labels)
                    test_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(trainloader))
            test_losses.append(test_loss/len(testloader))
            print(f"Epoch {epoch+1}/{epochs}.. "
                  f"Train loss: {running_loss/print_every:.3f}.. "
                  f"Test loss: {test_loss/len(testloader):.3f}.. "
                  f"Test accuracy: {accuracy/len(testloader):.3f}")
            running_loss = 0
            model.train()
```

```
Training step  1
```

```
Training step  2
Training step  3
Training step  4
Training step  5
Epoch 1/5.. Train loss: 1.822.. Test loss: 0.625.. Test accuracy: 0.646
Training step  6
Training step  7
Training step  8
Training step  9
Training step  10
Epoch 2/5.. Train loss: 0.762.. Test loss: 0.455.. Test accuracy: 0.806
Training step  11
Training step  12
Training step  13
Training step  14
Training step  15
Epoch 2/5.. Train loss: 0.627.. Test loss: 0.608.. Test accuracy: 0.744
Training step  16
Training step  17
Training step  18
Training step  19
Training step  20
Epoch 3/5.. Train loss: 0.452.. Test loss: 0.388.. Test accuracy: 0.873
Training step  21
Training step  22
Training step  23
Training step  24
Training step  25
Epoch 4/5.. Train loss: 0.308.. Test loss: 0.350.. Test accuracy: 0.838
Training step  26
Training step  27
Training step  28
Training step  29
Training step  30
Epoch 4/5.. Train loss: 0.541.. Test loss: 0.391.. Test accuracy: 0.808
Training step  31
Training step  32
Training step  33
Training step  34
Training step  35
Epoch 5/5.. Train loss: 0.408.. Test loss: 0.218.. Test accuracy: 0.935
Training step  36
Training step  37
Training step  38
Training step  39
Training step  40
Epoch 5/5.. Train loss: 0.256.. Test loss: 0.241.. Test accuracy: 0.935
```

7. Determining the accuracy of a neural network in classifying space rocks

In [23]:

```
print(accuracy/len(testloader))
```

```
0.9354166686534882
```

8. Saving the model

In [24]:

```
torch.save(model, 'aerialmodel.pth')
```

9. Predicting the type of space rock in a random photo

Let's predict rock types. To predict the type of rock that's shown in a new image, we need to complete the following steps:

1. Convert the new image to numbers.
2. Transform the image: crop and resize it to 224 × 224 pixels.

3. Extract the features and characteristics of the image.
4. Predict the type of rock that's shown in the image by using the associations we learned in step 6.

9.1 Loading the neural network

In [25]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model=torch.load('aerialmodel.pth')
```

9.2 Creating a function that predicts the new image type

In [26]:

```
def predict_image(image):
    image_tensor = test_transforms(image).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input = Variable(image_tensor)
    input = input.to(device)
    output = model(input)
    index = output.data.cpu().numpy().argmax()
    return index
```

10. Testing a neural network that classifies photos of space rocks

In [28]:

```
# Geting five random images and storing their data in variables
images, labels = get_random_images(5)

# Visualizing the new images and adding captions indicating what type of rock the model determines
the photo contains
to_pil = transforms.ToPILImage()
images, labels = get_random_images(5)
fig=plt.figure(figsize=(20,10))

classes=trainloader.dataset.classes
for ii in range(len(images)):
    image = to_pil(images[ii])
    index = predict_image(image)
    sub = fig.add_subplot(1, len(images), ii+1)
    res = int(labels[ii]) == index
    sub.set_title(str(classes[index]) + ":" + str(res))
    plt.axis('off')
    plt.imshow(image)
plt.show()
```