Challenge 1: Gearing Up for Machine Learning

Background

Before working on a computer vision solution, the data science team at Adventure Works wants to standarize on a collaborative development environment that will work well with common Python libraries.

After provisioning the development environment, you can use it to explore the images that Adventure Works has collected, and prepare them for use in training a machine learning model.

Challenge

This challenge has three elements:

1. Set up an environment for collaborative development
2. Explore Image Data
3. Pre-Process Image Data

Each challenge includes some detailed requirements and hints to help you. Additionally, there's a **References** section at the bottom of this page with links to useful resources.

1. Set up an environment for collaborative development

Your development environment should include:

- Python 3.6
- Jupyter or JupyterHub access
- pip (Python package manager)

You can use:

- An Azure-based Data Science Virtual Machine (DSVM) (**recommended**)

- A local data science development environment on your machine

In addition, the team is encouraged to set up tools for code sharing and team collaboration such as Git.

Data Science Virtual Machine

Create a single Azure Data Science Virtual Machine (DSVM) for the team, using one of the environment logins provided. The following DSVM configuration has been found to work well, and is the recommended environment for this OpenHack:

- DSVM Image: *Data Science Virtual Machine for Linux (Ubuntu)*
- Region: *(Ask your coach for the appropriate region for your OpenHack)*
- Size: *NC6* Standard (GPU Family)
- Authentication type: *Password*
- Username: *(Specify a **lowercase** user name of your choice)*
- Password: *(Specify a complex password)*

Hints

- Your subscription has limited resources, so create one DSVM for the team.
- When provisioning the DSVM, specify a **lowercase** user name and be sure to choose **Password** as the authentication type. All team members will be able to use these credentials to connect to Jupyterhub on this virtual machine - for information about using Jupyterhub, see this video or this document.
- The Jupyterhub is at **https://*your.dsvm.ip.address*:8000**
- To get to the Jupyterhub, you must click through the non-private connection warnings in browser - this is expected behavior.
- Use the **"Python 3"** kernel when creating new notebooks.
- Use `pip` to install Python packages. Example commands for the Ubuntu DSVM are shown in the documentation.

See the **References** section below for more guidance and help.

Local Computer Alternative to DSVM setup

- Install Anaconda if you don't have it for your system:
  - Review the installation information here
  - Create an environment with Python 3.6: `conda create -n py36 python=3.6`
  - Activate the environment. Windows: `activate py36` Unix/Linux: `source activate py36`

    - You will be able to `pip` or `conda` install packages into this environment as needed going forward
- If *not* using Anaconda, but rather a system Python install, use the Python package manager (**pip**) to at least install Jupyter:
    - `pip install jupyter`
- Install other Data Science packages as the need arises

2. Explore Image Data

After setting up your environment, you must download the Adventure Works image data from https://challenge.blob.core.windows.net/challengefiles/gear_images.zip, and extract the image files. A convenient way to download the data is to run OS commands within a Jupyter notebook cell by prefixing them with a `!` character, for example:

```
! curl -O https://challenge.blob.core.windows.net/challengefiles/gear_images.zip
! unzip -o gear_images.zip
```

When you have downloaded and extracted the images, you can use Python to explore them. In this case, you can use the following Python code to iterate through the folders of images and display the first image in each folder with the name of the folder (which is the category, or *class*, of product shown in the image):

```
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# Needed to display matplotlib plots in Jupyter
%matplotlib inline

imgdir = 'gear_images' # Folder containing extracted images

# Create a figure to display the images
fig = plt.figure(figsize=(12, 16))

# loop recursively through the folders
dir_num = 0
for root, folders, filenames in os.walk(imgdir):
    for folder in folders:
        # in each folder, get the first file
        imgFile = os.listdir(os.path.join(root,folder))[0]
        filePath = os.path.join(root,folder,imgFile)
        # Open it and add it to the figure (in a 4-row grid)
```

```
        img = Image.open(filePath)
        a=fig.add_subplot(4,np.ceil(len(folders)/4),dir_num + 1)
        imgplot = plt.imshow(img)
        # Add the folder name (the class of the image)
        a.set_title(folder)
        dir_num = dir_num + 1
```

Running the code should produce an output similar to this:

Gear Images

Hints

- The **os** Python module includes functions for interacting with the file system.
- The **matplotlib** Python library provides functions for plotting visualizations and images.

- To ensure that plots are displayed in a notebook, you must run the following *magic* command before creating the first plot:

```
%matplotlib inline
```

3. Pre-Process Image Data

The images collected by Adventure Works data scientists are a variety of sizes and shapes, and need to be standardized before they can be used to create a machine learning model. You must write Python code to create resized versions of all of the images that are 128x128 pixels in size. The resized images should not be skewed or deformed, and non-square source images should result in a square 128x128 color (RGB) image with white padding where necessary.

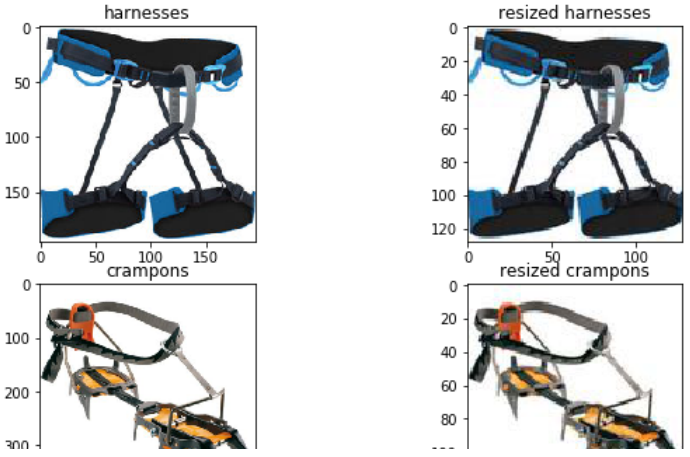
To accomplish this, create a Python function that:

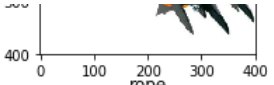
1. Ensures the image has three color channels (RGB)
2. Resizes the image so that the largest dimension matches the specified size
3. Pads the image as necessary to ensure it is square

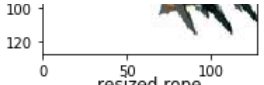
After defining your function, write code that uses it to create new 128x128 JPG versions of all of the images. Your code should save the resized images in a folder structure that matches the original **gear_images** folder, like this:

- resized_images
  - axes
  - boots
  - etc.

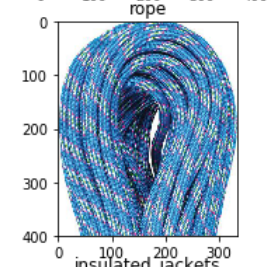
Finally, write code to display the first image in each folder along with its resized counterpart, like this:



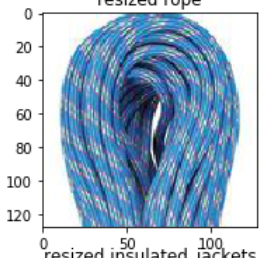




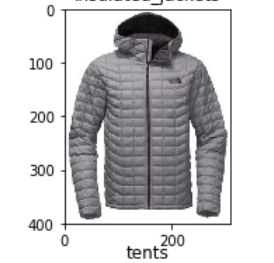
rope



resized rope



insulated_jackets



resized insulated_jackets

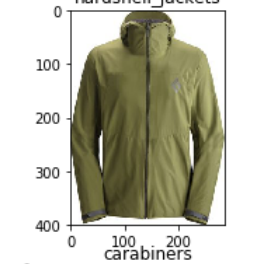
tents

resized tents

gloves

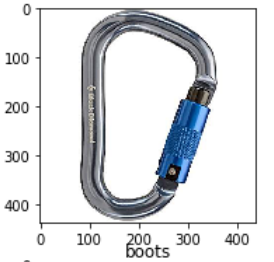
resized gloves

hardshell_jackets



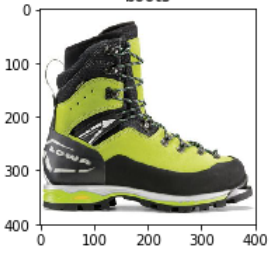
resized hardshell_jackets

carabiners

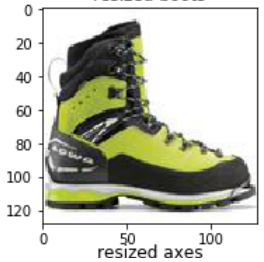


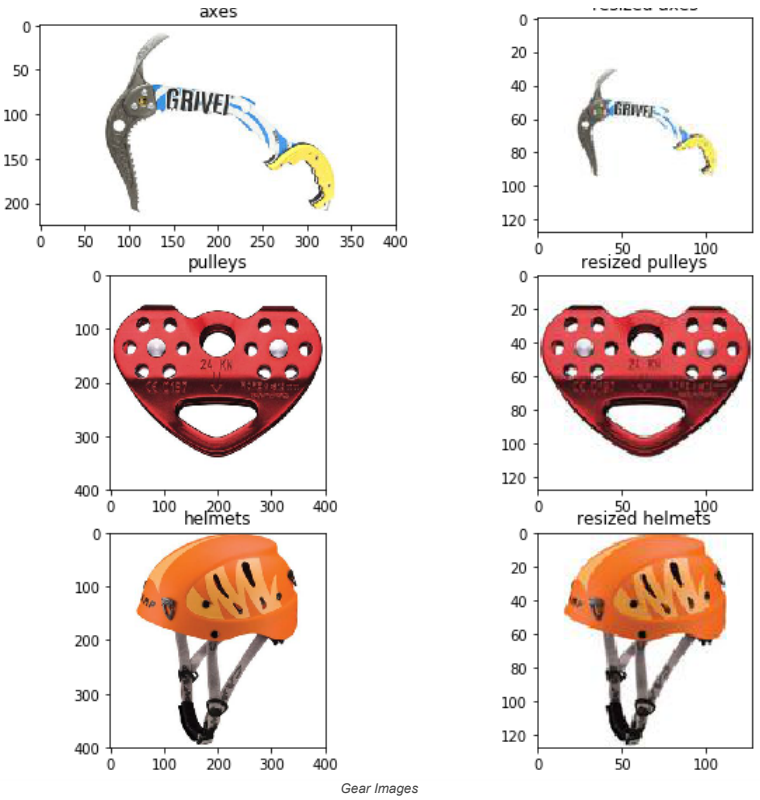
resized carabiners

boots



resized boots





Gear Images

Hints

- Images are essentially just numeric arrays. In the case of color images, they are three-dimensional arrays that contain a two-dimensional array of pixels for each color channel. For example, a 128x128 Jpeg image is represented as three 128x128 pixel arrays (one each for the red, green, and blue color channels). The Python *numpy* library provides a great way to work with multidimensional arrays. For example, you can use:
  - `numpy.array(my_img)` to exlicitly convert an image object to a numpy array.
  - `my_array.shape` to determine the size of the array dimensions - an image has three dimensions (height, width, and channels)
- There are several Python libraries for working with images, as noted in the **References** section. You can use whatever combination of these packages works best to process your images, and rely on the **numpy** array data type as an intermediary format.
- The *PIL* library uses a native format for images, but you can easily convert PIL images to numpy arrays using the `numpy.array()` function, and you can convert a numpy array to a PIL Image object by using the `Image.fromarray()` function. You can also convert PIL images between image formats (for example, from a 4-channel PNG to a 3-channel JPG) using the `my_img.convert()` function.

- To open a file as a PIL Image object, use the `Image.open()` function. To save a PIL image as a file, use the `my_img.save()` function.
- A common strategy to resize an image while maintaining its aspect ratio is to:

    1. Scale the image so that its largest dimension (height or width) is set to the target size for that dimension. You can use the PIL `my_image.thumbnail()` method to accomplish this.
    2. Create a new image of the required size and shape with an appropriate background color. You can use the PIL `Image.new()` function to accomplish this.
    3. Paste the rescaled image into the center of the new background image. You can use the PIL `my_bg_img.paste()` function to accomplish this.

- When using *matplotlib* to plot multiple images in a grid format, create a figure and add a subplot for each image by using the `my_figure.add_subplot()` function. The parameters for this function are:
    - The number of *columns* in the grid.
    - The number of *rows* in the grid.
    - The *ordinal position* of this subplot in the grid.

Success Criteria

1. Verify you can open or create a Jupyter notebook in the development environment.
2. Run a code cell that displays the first image in each folder.
3. Run a code cell that displays the first image in each folder alongside its resized version.

References

Data Science Virtual Machine

- Provisioning a Ubuntu DSVM

Jupyter and Conda

- Getting started with conda
- Creating and activating a conda environment
- Connecting a Jupyter Notebook to a specific conda environment

Python

- Python 3.6 documentation

Image Processing

- *matplotlib* for image I/O and plotting
- *numpy* for image manipulation/processing/visualization
- *numpy* for image I/O
- *PIL Image* module for I/O and more
- *PIL ImageOps* module for image manipulation
- *OpenCV* - another library for image processing

(*Note: In **opencv** images are read in a BGR format, whereas **matplotlib** reads and expects images as RGB. Conversion information can be found here*)

> **Note**: Data pre-processing is often the most time-consuming part of a machine learning project. When working with image data, there are many ways in which the data can be enhanced for use in machine learning, for example by scaling the images to be a consistent size and shape, adjusting contrast to correct for over/under-exposure, or cropping to include only the most relevant visual elements. The goal of pre-processing image data is to ensure that all of the images used to train a model are consistent, minimizing any bias caused by variation in size, orientation, contrast, and so on. In this OpenHack, we've restricted ourselves to ensuring that all of the images are the same size and shape; but you should be aware that in a real-world project you may need to spend significant time and effort on image pre-processing.