Challenge 4: Into the Deep

Background

The Adventure Works data science team wants to try *deep learning* algorithms on the data to see if the accuracy of gear classification improves. The team anticipates an influx of catalog and outdoor pictures soon, and deep learning shines when the data size gets bigger. Also, the Data Science team is eager to learn more about deep learning and convolutional neural networks (CNNs) for image analysis because CNNs naturally lend themselves to performing well on complex data such as images, and can require less pre-processing and feature engineering.

A CNN consists of a series of layers in which *weights* are applied to the pixel values in the images based on filters that are convolved across the image (hence the name). The data is passed through the layers in a series of iterations (or *epochs*) in which you train the model using a subset of the image data, and validate it with another subset (usually, within each epoch the training and validation data is passed through the network in multiple *mini-batches*). For each batch of data passed through the network, a *loss function* is used to calculate the error (or *loss*) in the predictions based on the known labels in the source data, and then a process of *backpropagation* is used to adjust the weights and decrease the amount of loss.

Prerequisites

- A development environment for sharing code and working in Jupyter
- The resized ***gear*** image data fom the previous challenges.
- An installation of the latest version of your chosen deep learning framework(s) based on the **References** section below.

Challenge

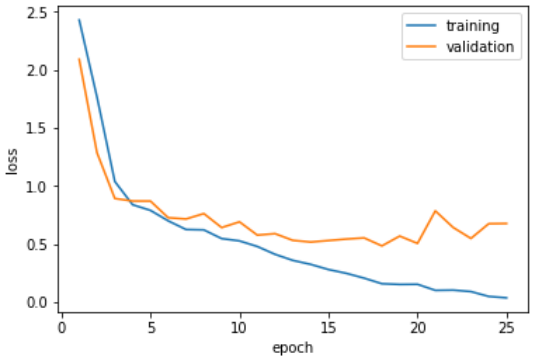
There are three elements to this challenge: 1. Create and train a convolutional neural network (CNN) model 2. Use your model with new data 3. Use Transfer Learning

1. Create and train a convolutional neural network (CNN) model

Create a CNN that predicts the class of an image based on the *gear* dataset.

- The architecture of your model should consist of a series of *input*, *convolutional*, *pooling*, and *dense* layers that you define.

- For each epoch in your training process, you should record the average *loss*for both the training and validation data; and when training is complete you should plot the training and loss values like this:



Training and Validation Loss

Hints

- Training a deep learning model is faster on a GPU machine. If the team environment does not already include GPU, consider working with your coach to adjust the environment before proceeding. The N-series virtual machine sizes in Azure support GPUs.
- Just as with classical machine learning, you should randomly split the data into training and validation subsets with which to train and validate the model.
- As you iterate on the model architecture, save the best trained models to disk.

- Consider the following suggested starting point architecture:

    1. Input Layer (3 channel image input layer)
    2. Convolutional (2D)
    3. Max Pooling
    4. Convolutional (2D)
    5. Max Pooling
    6. Dense (Output layer)

- Try to avoid *overfitting* your model to the training data. One sign of this is that after your training and validation loss metrics converge, the training loss continues to drop but your validation loss stays the same or rises (as shown in the image above). The end result is a model that performs well when predicting

the classes of images that it has been trained on, but which does not generalize well to new images.

- See **References** for resources and more information.

2. Use your model with new data

Use your model to predict the class of at least five images that are not included in the ***gear*** dataset. You can use the same five images you found in the previous challenge.

3. Use Transfer Learning

Create a second CNN, but this time use a technique called *transfer learning* to build a classifier on top of the feature extraction layers defined in an existing model. See **References** below for resources and more information about transfer learning.

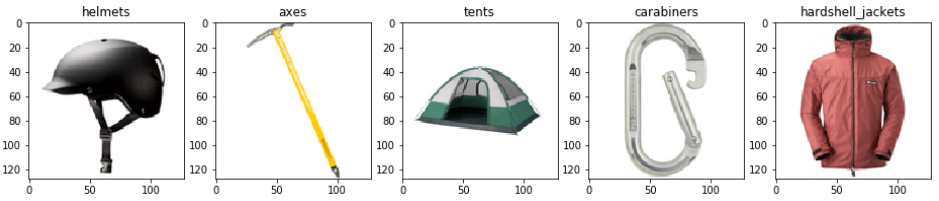
Hints

- You may need to resize the images to match the size used to train the base model you select.

Success Criteria

- Successfully train a convolutional neural network model
- Plot the average training and validation loss observed when training your model
- Achieve model accuracy of **0.9** (90%) or greater using your test data set.

- Show predictions for the five images you identified in the **Challenge** section, like this:



Gear predictions

(Note: Your model is not required to predict the correct class for all of the images, but it would be good if it does!)

References

CNN Concepts

- An Intuitive Explanation of Convolutional Neural Networks
- How Convolutional Neural Networks work (video)
- Demystifying AI (video)

Deep Learning Frameworks

- PyTorch
  - Documentation
  - Tutorials
- Keras (an abstraction layer that uses a TensorFlow or CNTK backend)
  - Documentation
  - Tutorials
- TensorFlow
  - Documentation
  - Tutorials
- Cognitive Toolkit (CNTK)
  - Documentation
  - Tutorials

Transfer Learning

- Transfer Learning Notes
- Transfer Learning with PyTorch
- Transfer Learning with Keras
- Transfer Learning with TensorFlow
- Transfer Learning with CNTK