Transfer Learning on Image Data

The purpose of this set of notes is to give you background information for your next programming assignment.

1 Assignment

Create a program that classifies images of bees versus ants. You should follow the PyTorch tutorial on "Transfer Learning for Computer Vision" located at
`https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html`
I strongly recommend working through the "Introduction to PyTorch" tutorials before completing this tutorial. Start with the "Tensors" tutorial and work your way through the "Save and Load a Model" tutorial.

At the end of the transfer learning tutorial, you will generate a figure demonstrating the output of your code. You should upload your completed python file and this image to sakai.

2 Image Data

Bees vs Ants:

- 2 classes
- 398 images
- Pre-determined train/test split (245/153 images)

ImageNet dataset:

- 1000 classes, 1.2 million images
- test set size = 150,000

Image data is usually "noisy"

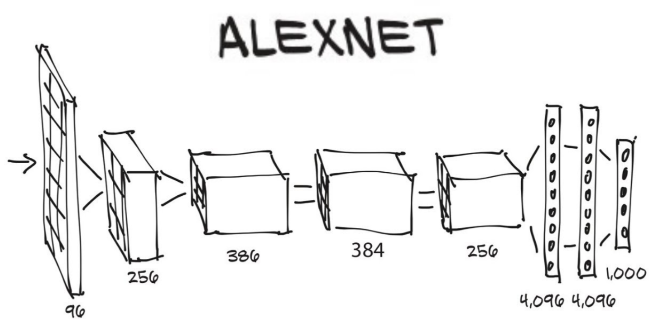
- Non-statistical "noise"
  - images are all of different resolutions
  - size of the bug is different for each image
  - some images have effects applied to them
- Statistical "noise"
  - scraped from the web, so there's errors
  - human annotations, and humans make mistakes
    Karpathy: human error rate is about 5%
    `http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/`
  - some images can legitimately have multiple labels
    Reference: `https://www.unite.ai/assessing-the-historical-accuracy-of-imagenet/`

3 Deep Learning

Reference: `https://cs231n.github.io/`

- All famous deep learning image classification models are all trained on ImageNet
- ImageNet performance change over time:
  `https://paperswithcode.com/sota/image-classification-on-imagenet`
- Best results use ensembles of deep learning algorithms
  For example: "Deep Residual Learning for Image Recognition"

What deep learning looks like:



Theoretical Problems

- Optimization: Highly non-convex optimization problems

- Statistical: VERY high VC dimensions

  VC dimensions are VERY difficult to directly compute, so people typically report the number of parameters used in the model as a "proxy" for the VC dimension.

4 Tricks for our dataset

Three tricks:

1. Use an ImageNet-trained CNN for feature extraction

    - "transfer knowledge" from the ImageNet task to the bees vs ants task

    - greatly reduces the VC dimension

    - Computational notes:
        - pytorch's `CrossEntropyLoss` is the logistic loss
        - Lots of variations of SGD (Momentum, Adam, AdaGrad)
        - Batch size
        - Learning rate scheduler
        - Still expensive computationally

2. Solution 2: dataset augmentation

   - Reference: `http://d2l.ai/chapter_computer-vision/image-augmentation.html`
   - "effective number of data points" equals number of epochs times training set size
   - technically these new data points are not independent, and so we're not "really" increasing our dataset in the VC dimension sense... but it's really close and works well in practice

3. Solution 3: add regularization

   - weight decay
   - early stopping
   - dropout
   - residual layers
   - batch normalization