In this project, you will be using convolutional neural networks for image processing, and comparing results with or without pre-training and with or without data-augmentation. You are welcome to work in either or both of Tensorflow and Pytorch. You are encouraged to start with a `scratchwork` notebook, and you will eventually submit a Jupyter notebook called `writeup.ipynb`, but these have not been provided in the starting-point repo.

The `writeup.ipynb` notebook you submit should be a clean and well-organized presentation of your implementation, experiments, and results, including explanatory text made readable with proper markdown formatting, and citations where appropriate. Your `writeup` should describe the things you tried, but code and outputs should be pruned down to the key experiments and the most interesting and relevant conclusions. Large blocks of code that you need to call, but that don't contribute to the explanation should be moved to a `.py` file and imported.

Partners and Groups

You are required to work with your assigned partner on this project. You are responsible for ensuring that you and your partner both fully understand all aspects of your submission; you are expected to work collaboratively and take responsibility for each other's learning.

You will also be assigned a code-review partner after the deadline. To prepare for that code review, make sure you understand all of your code and that it is cleaned up to be readable by people outside your group. See the code review guidelines for more information.

Data Sets

For this project, there are two primary image data sets in the `/home/DAVIDSON/brwiedenbeck/public/` directory: `brain_scans`, and `birds`. The `brain_scans` data set has a relatively small number of low-resolution grayscale images of brian-scans from healthy individuals and from patients with Alzheimer's. Your task is to train a model that can help with diagnosis. The `birds` data set has a large number of medium-resolution color-images of many species of birds. Your task is to train a model that can help with species-identification. There is also a third data `chest_xrays` that you may play around with as an optional extension; it has a small number of high-resolution grayscale images of chest-xrays from healthy individuals and from patients with Pneumonia.

You can make these data sets easier to access by creating a symlink with a command like:

```
ln -s /home/DAVIDSON/brwiedenbeck/public/brain_scans
```

These data sets will present very different challenges. For the `brain_scans` data set, the small number of images will make data augmentation particularly important, and the small grayscale images may require transformation before you can apply convolution or transfer learning. For the `birds` data set, the large number of images will make it hard to hold everything in memory, so your training loop will need smart data loading, and the higher resolution may turn out to be more of a hassle than a help. The optional `chest_xrays` data set has the additional complication of irregularly shaped images.

Models

For both tasks, start by training a convolutional network. You should experiment with different architectures and hyperparameters, but your network is required to have at least two convolutional layers. For large data sets like `birds`, it makes sense to start with a small subset of the data to try and get the basics working before you train on everything. In each case, try to optimize your basic convolutional network to achieve the best performance you can before we compare it against other approaches.

Next, you should explore data augmentation, including stretching, zooming, rotation, flipping, adding noise, and any other ideas you think might be helpful. The deep learning libraries have some tools for data augmentation, but the Python Imaging Library is also a helpful resource. Determine experimentally which sorts of augmentation are helpful with respect to performance on the validation sets.

Finally, you should perform transfer learning. Select a pre-trained convolutional model, and replace its last layer(s) with your own, and re-train it on your image-processing task. TensorFlow and Pytorch each have "model zoos" with many examples of pre-trained models you can use for transfer learning. However, these models will generally require a specific type of input, so make sure you read about your chosen model and think through what input transformations it will require. Also note also that pre-trained models vary substantially in size and architecture, which can have significant impacts on training time, memory use, and performance.

As an optional extension, you can also try transfer learning with a residual network. These tend to be much larger than convolutional models, so pre-training is even more important.

Writeup

You should submit a `writeup.ipynb` notebook with a clear explanation of what you've implemented and what you found in your experiments. This should show comparisons on both problems between (1) basic convolution, (2) data augmentation, and (3) transfer learning. Explain what went into training each model variant, how it performed, and what you learned. You should use markdown features including headings, paragraphs, lists, and so on to make your write-up read like a blog post. Think of this as a mini-version of the sort of write-up you will be doing for the implementation project at the end of the semester.

As a part of your write-up, you should include citations to all resources you used in coming up with your solutions. The expected format for citations is a numbered list of web-links, along with a one-sentence description of what you learned from each resource, and in-text references to those citation numbers.

Resources

Each of the data sets we're using for this project comes from Kaggle, and on Kaggle you can find many examples of how other people have worked with these data sets. *You are encouraged* to read other people's Kaggle notebooks and learn from them; don't try to figure everything out on your own! But to ensure that you're learning from those approaches and not just parroting them, we'll need to establish some ground rules. You may:

- view any Kaggle notebooks you wish, and
- take notes on any functions they use and how you think they might be useful.

But you may not:

- have both someone else's notebook and your own open at the same time, or
- copy any amount of code from anyone else's notebook.

The key idea here is that you will use existing notebooks as a source of ideas for the sorts of tools you might want to use, but you will then read the documentation on those tools to figure out how to use them yourself. If you're uncertain about what's allowed, don't hesitate to ask in class, in office hours, or on Slack. Make sure you keep track of links to any notebooks you find useful so that you can cite them in your write-up.