ML@B NMEP HW 6
Fall 2018

Your task is to develop a model which takes in images, and predicts whether they are one of three classes:

1. Deer:

2. Sheep:

3. The elusive and majestic hartebeest:

We're gonna use the power of Convolutional Neural Networks to classify our images. But, we don't have weeks to train our networks, so we'll leverage other people's work to make our training faster. In other words, we'll fine-tune a pretrained model for our task. In particular, output a 3 dimensional vector: [Pr(Image is a Hartebeest), Pr(Image is a Deer), Pr(Image is a sheep)], given an image of size h * w (default is 224 in the code)

To start, change your directory to the project root, then:

```
bash get_model.sh
```

This will create an "experiments" and a "pretrained_models" directory, then download a pretrained ResNet-50 into the latter directory.

The code you will have to fill out is in 2 files: `model.py` and `image_dataset.py`. The goal of our code is as follows: we run main.py with some arguments, which makes a TransferModel object (defined in `model.py`), and then does some stuff based on those arguments. If we tell it to train, it will either restore a previous training session, or initialize a new model and start training it on our data. We can also tell it to take some model we've already trained and compute the probabilities of a particular image (running the model is called inference). Look at `main.py` to get a sense of how this happens. We pass in logistical arguments to `main.py`, as well as the name of a configuration file (`config.yaml`), which defines hyperparameters.

We keep data in the `data` folder. Right now, there are only a couple images for each class for training and a single image for validation. We will release the entire dataset shortly,

which will be large enough to give decent performance. However, you can still work with this mini-dataset (as it will still test your basic functionality). Note that we split the data into training and validation. The state of our files will be, when we are training, as follows:

```
├── config.yaml
├── data
│   ├── train
│   │   ├── deer
│   │   │   ├── d1.jpg
│   │   │   └── ...
│   │   ├── hartebeest
│   │   │   ├── h1.jpg
│   │   │   └── ...
│   │   └── sheep
│   │       ├── s2.jpg
│   │       └── ...
│   └── val
│       ├── deer
│       │   └── d3.jpg
│       │   └── ...
│       ├── hartebeest
│       │   └── h3.jpg
│       │   └── ...
│       └── sheep
│           └── s3.jpg
│           └── ...
├── experiments
│   ├── m0 # The folder where we keep everything associated with a single experiment
│   │   ├── ckpts #These are tensorflow's saved weights files--read about these
│   │   │   ├── checkpoint
│   │   │   ├── model.ckpt-12.index
│   │   │   ├── model.ckpt-16.index
│   │   │   ├── model.ckpt-4.index
│   │   │   ├── model.ckpt-4.meta
│   │   │   └── model.ckpt-8.index
│   │   ├── config.yaml # The hyperparameter file, for bookkeeping
│   │   └── logs #Tensorboard output logs, serialized
│   │       ├── train
│   │       │   └── events.out.tfevents.1539762258...
│   │       └── val
│   │           └── events.out.tfevents.1539762262...
├── get_model.sh #Quick script to get set up
├── image_dataset.py
```

```
├── main.py
├── model.py
├── pretrained_models
│   └── resnet_v2_50.ckpt
├── test_on_img.sh #Script that runs main.py on an image
├── train.sh #Script that runs main.py to train
└── utils.py
```

Tasks are indicated by #TODO's in `image_dataset.py` and `model.py`.
`model.py` represents your model and should be able to train and to predict. While training, it should be saving its parameters periodically while logging its performance to tensorboard, in particular train/val loss and accuracy.
`image_dataset.py` represents your dataset, which should be able to go through epochs of data and serve batches of images on request. At no point should you keep the entire dataset in memory! The exact implementation is up to you.

This should be enough to get you started!

Deliverables:

1. All code written for `image_dataset.py` and `model.py`.
2. Final performance on validation set.
3. Screenshots of logs in tensorboard for train/validation loss (should be on the same graph) and accuracy as your model trains