Lab 3: Galaxy image classification and the galaxy merger rate

Astro 128 / 256 (UC Berkeley)

November 6, 2024

Assigned: Wednesday, Nov 6, 2024
Checkpoints: Wednesday, Nov 13, 2024; Monday, Nov 25, 2024; Tuesday, Dec 3, 2024
Final Write Up Due: Friday, Dec 6, 2024

Introduction

In this lab, we'll build a model to morphologically classify images of galaxies *en masse*. When shown an image of a galaxy (or multiple galaxies, or something else that looks like a galaxy), the model should be able to estimate various properties that a human astronomer could estimate by eye. Is the galaxy a spiral or an elliptical? If it is a spiral, how many spiral arms does it have? Does it have a bulge or bar? If so, how prominent is it? Is it lensed? Does it have visible dust lanes? Is it merging with other galaxies? Humans are good at answering these questions, but they are slow. The model we build should perform almost as well as a human, but it should be much faster.

Once we have a trained galaxy classifier, we'll use it to search a large sample of galaxy images for galaxies that are currently undergoing major mergers. Mergers are pivotal events in the life cycle of galaxies and are signposts of hierarchical assembly. Finally, we'll estimate the galaxy merger *rate* and compare it to theoretical predictions from cosmological models.

We'll use images from the Sloan Digital Sky Survey. A large number of SDSS image have already been classified by humans through the Galaxy Zoo project. Those classifications will provide the training data for our model. The Galaxy Zoo project is described in Lintott et al. 2008. A recent high-level summary of results from Galaxy Zoo can be found in Masters et al. 2019. The morphological classification scheme used to generate the data used in this lab is described in Willett et al. 2013. A good practical description of image classification using convolution neural networks can be found in Krizhevsky et al. 2020.

Technical Components

- Correlation & covariance
- Image classification
- Decision trees
- Convolutional neural networks
- Overfitting & regularization
- Memory management
- Data augmentation
- Basics of galactic astronomy and cosmology

Useful Coding Tutorials for Getting Started

- A Gentle Introduction to PyTorch
- Introduction to PyTorch
- Deep Learning with PyTorch: A 60 min Blitz
- Python Generator Functions

Broad Summary of Steps

You should expect the lab to proceed in the following order:

- Learn more about galaxy classification and morphology by trying out Galaxy Zoo and reading linked papers. A solid understanding of this is important as many of the questions later in the lab depend on these details.
- Download the image training set and perform some qualitative and quantitative exploration to get familiar with the data.
- Resize the images in order to reduce RAM requirements.
- Build your own neural net to classify the images using cross-validation. Training the neural net for this lab can take hours and you will likely have to repeat it several times. Budget your time accordingly.
- Use a ResNet (with and without data augmentation) to classify your galaxy images. These steps can also take hours and you may have to iterate. Budget your time accordingly.
- Use your neutral net results to explore the merger rate of galaxies.

Let's begin!

1. Go to GalaxyZoo/Classify and classify a few galaxies. If you aren't sure how to classify an object, trying clicking "Need some help with this task?". Comment on the classification scheme. Did you encounter any ambiguous objects? Show an example (a screenshot is fine) of a galaxy you encountered with clear spiral structure, and one of a galaxy that is morphologically smooth.
2. Read through Willett et al. 2013 in detail, and briefly answer all of the following questions (no more than a few sentences each).
    - What cuts were applied to the SDSS DR7 imaging survey to construct the Galaxy Zoo 2 (GZ2) sample? How many galaxies are in GZ2?
    - Describe the galaxy images in GZ2. What are the angular dimensions of the images? Are they the same for all galaxies? Roughly what physical scale in kpc does this correspond to? (You may have to calculate this).
    - Describe the classification scheme in GZ2. How many questions are asked for each galaxy?
    - How did the classification scheme change from GZ1 to GZ2?
    - How does GZ2 decide which galaxies to show a user and in what order?
    - How many unique classifications did a typical galaxy get (at the time this paper was written)?
    - What steps were taken to deal with unreliable classifications?
    - What is the primary reason for apparent redshift evolution (i.e. changes in mean morphological type as a function of redshift) within the GZ2 sample?
    - Briefly describe (in words) how GZ2 attempts to correct for classification bias. What is meant by classification bias?

- Why are higher-redshift galaxies in GZ2 more likely to be classified as mergers?
- GZ2 classifies each galaxy using boolean yes/know questions for a finite set of possible classifications. Yet, the reported classifications are all floats (typically ranging from 0 to 1). Why is this?
- Summarize the most significant differences between GZ2 classifications and similar classifications from other morphological catalogs.

3. Why might astronomers care about morphological classification of galaxies? Check out Conselice, 2014. Give some examples of how morphological classification informs our understanding of the evolutionary history of a galaxy.

4. Why might astronomers want to systematically search for merging galaxies? Check out Lotz et al., 2011 and Rodriguez-Gomez et al, 2015. How have merging galaxies been identified in the literature in the past?

5. Download the training set (`training_images.tar.gz` and `training_classifications.csv`, here, or access them from the shared directory (`$HOME/shared/ay128-2024/lab3_galaxies`) on Astro Datahub. How many images are in the training set? What is their dimension? Plot 25 random images from the training set as RGB images (`plt.imread()` and `plt.imshow()` work well for this).

6. Let's inspect the labels that came with the training set. Plot distributions (i.e., normalized histograms) of the training labels for each of the 37 labels, and label them by the descriptive labels given in Table 2 of Willett et al. 2013 (i.e., "smooth", not "Class1.1"). Comment on the distributions. Are any of them multi-modal, or particularly skewed? What does that tell us about that label?

7. For each of the 37 labels, plot the image in the training set that is the prototype of that label; i.e., the image for which the label has the highest value. Label each image by its ID and by the label it represents. Based on these images, which labels do you think will be easy to classify (i.e., which are visually obvious)? Which are subtle and more likely to cause problems?

   Checkpoint 1: show (a) your label distributions from (6) and (b) your example images from (7). Submit this via gradescope.
   It should be a pdf, named Firstname_Lastname_lab3_cp1.pdf

8. Plot the correlation matrix of all labels in the training set. That is, define the correlation between labels i and j as

$$\rho_{ij} = \frac{\langle ij \rangle - \langle i \rangle \langle j \rangle}{\sqrt{\langle i^2 \rangle - \langle i \rangle^2}\sqrt{\langle j^2 \rangle - \langle j \rangle^2}}, \tag{1}$$

   compute this for each pair of labels, and plot the matrix $\rho_{i,j}$ as a 37x37 image or heatmap. Comment on anything interesting or surprising. Some of the labels are designed to be mutually exclusive (i.e., a galaxy cannot be both smooth and featureless (Class1.1) and have features or a disk (Class1.2). What is the expected correlation coefficient in such cases? The categories with each class (e.g. Class1.X, Class2.X, etc) are mutually exclusive. How do the actual correlation coefficients compare to the value they "should" have? What does this tell us about the quality of the classifications in the training set? Are some labels less reliable than others?

9. We are going to build a model to predict classifications (labels) based on images. To train the model, we will have to repeatedly compare predictions for each image in the training set to its known labels. What is the minimum amount of memory it would take (within a factor of ~ 2) to load each image in the training set into memory? Don't worry (yet) about the memory required to store the model. *Hint: You can estimate this by thinking about how many pixels are in the total set of images.*

10. One way to reduce the computational cost of classification and memory requirements is to downsize the images. In our experience, you can reduce the size of the dataset by at least a factor of $\sim$30 without much reduction in performance; you can do this through a combination of cropping the images (removing mostly empty space at the edges) and resampling them to a coarser pixel grid. Write a

function to downsize all the images. Aim for roughly a factor of 30 in size reduction for now; you can refine this later. Make plots comparing a few images before and after downsizing.

11. Given that we have finite memory available (8 GB minus overhead on Datahub) we'll need to read in the data in batches. You may be able to use the dataloader and dataset classes in PyTorch to help.

    If you find that these do not work well, another option is to write a generator function to sequentially read in and "yield" batches of images and their labels without keeping the full dataset in memory. You should pay attention to performance here: during training, you will need to repeatedly read in and run computations on ∼50,000 images. You don't want reading in to be the bottleneck, so you should aim for $\ll 0.01$ seconds per image for reading in and fetching the labels. There are various ways to achieve this; one option is to save the images and their labels to disk in batches of, say, 100-200, which is about the number we found can be comfortably trained on at once without running into memory issues on Datahub. You will probably find that it is more efficient to save images after downsizing than to resize them each time you read them in.

12. Split the images and labels into a training set (containing 80% of all objects) and a validation set (containing the remaining 20%). Compared normalized distributions of the training and validation labels (as in 6) to ensure there are no systematic differences between them. If there are, change your procedure for dividing the training and validation data to make it random.

13. Before implementing a neural network, let's first try something simpler. Let's simply guess that the labels for each image are the mean value of the corresponding label in the training set, without looking at the image at all. That is, guess the same label for each image. Obviously, this is a pretty crude guess. It will provide our a benchmark that our fancy neural network model had better outperform.

    The loss function we'll use throughout the lab is the **root mean squared error**, defined as

$$L_{\rm RMSE} = \sqrt{\left\langle \left(\overleftrightarrow{\ell}_{\rm true} - \overleftrightarrow{\ell}_{\rm pred} \right)^2 \right\rangle} \tag{2}$$

$$= \sqrt{\frac{1}{N_{\rm galaxies} N_{\rm labels}} \sum_{i}^{N_{\rm galaxies}} \sum_{j}^{N_{\rm labels}} \left(\ell_{{\rm true},ij} - \ell_{{\rm pred},ij} \right)^2}. \tag{3}$$

    Here $\overleftrightarrow{\ell}_{\rm pred}$ and $\overleftrightarrow{\ell}_{\rm true}$ are matrices (tensors, in PyTorch parlance) representing the predicted and true labels for each object.

    What is the RMSE loss for this simple model? Give the value for both the training and validation sets.

14. Implement a convolutional neural network (CNN) to predict labels based on images. We recommend using PyTorch, which is already available on Datahub. You are free to choose whatever network architecture you like; however, here are some suggestions that may be useful:

    - Include some convolution layers followed by some fully connected layers.
    - End with a sigmoid activation function, to force the outputs to be between 0 and 1.
    - Include a pooling layer (e.g., max or average) after each convolution layer.
    - Use the Adam or SGD optimizer.
    - Fight overtraining by including dropout layers that randomly shut off a given percent of neurons in each fully connected layer.

    In addition the lectures, we strongly suggest you seek out PyTorch examples on the web as a starting point for learning to train models and interface with PyTorch.

    Describe your network in words. What layers and activation functions does it have? How did you decide on this architecture? (One paragraph at most, please).

    Monitor the performance of the network during training by plotting both the training and validation loss as a function of the number of training images the network has seen. You'll be able to tell that the

network is trained once the validation loss stops dropping. When you finish training and are satisfied with your network's performance, save the network and training history `torch.save()` in PyTorch is a convenient way to save the network) so you don't have to retrain next time you run the notebook.

You will likely want to experiment with several different architectures until you find something that works well. A reasonable loss to aim for at this stage is RMSE ≤ 0.11.

Training the network will take some time – much more than the computations required in previous labs. But it shouldn't be anything too unreasonable. On Datahub, we find a good CNN can be trained in an hour or two (or less) of wallclock time.

Checkpoint 2: Submit a PDF plot of the training and validation loss as a function of training epoch from (15). Also submit your description (in words) of the network. Submit this via gradescope.
It should be a pdf, named Firstname_Lastname_lab3_cp1.pdf

15. How many total trainable parameters does your model have? In PyTorch, you can count the number of elements in a layer using `p.numel()`, where `p` in an element of `model.parameters`. You can tell whether a parameter is trainable using `p.requires_grad()`. Does the number of free parameters make you nervous? Explain. Compare it to the number of pixels in your compressed training set.

16. In building your networks, you may have struggled to decide on the right network architecture. How many layers? What kind of pooling? How much dropout? You might also have found that as the size of the network increases, it gets pretty tedious to set up the layers by hand and keep track of their dimension when they meet.

    One solution to both of these issues is to use a network architecture that has already been proven to work well for image classification. Check out He et al. 2015, which describes ResNets, one of the most popular networks introduced in computer vision in the last several years. Describe in a few sentences what ResNets are. How are they different from just stacking more and more layers on a CNN?

17. Implementations of several ResNets exist within Pytorch; see here. Try using the 18-layer ResNet (Pytorch) on your galaxy images. The process of training the model should be very similar to what you used for training your own network; just the initialization of the network will be different. As before, track the model and validation loss as you train, and save a copy of the model to disk once you're done training. Plot the training and validation loss. How does ResNet compare to your custom CNN? How many trainable parameters does it have?

18. You can likely improve the performance of the network by adjusting the learning rate of the optimizer. Use a scheduler (e.g., `torch.optim.lr_scheduler.ReduceLROnPlateau`) to decrease the learning rate (by a factor of your choice) every time the validation loss plateaus, still with the ResNet model. Again, plot the training and validation loss as a function of epoch. Do you see features in the loss that correspond to reductions in the learning rate? At this stage, a loss of RMSE ≤ 0.10 is a good target, and RMSE ≤ 0.09 is excellent.

19. You have probably found that after training for a while, your training loss continues to decrease while the validation loss reaches a plateau. This is evidence of overtraining. One way to reduce overtraining is data augmentation; that is, by artificially increasing the size of the training set. We can generate more training data by exploiting rotational and reflection symmetry: the labels of a galaxy should not change when you rotate or mirror it. Modify the functions you use to read in and compress batches of images, such that each image gets rotated by a random angle $\theta \in [0, 360]$ deg prior to being cropped and resized. Now, the network will never see the same image twice, even after many epochs of training. Try retraining your ResNet model using this data augmentation scheme, as well as a learning rate scheduler. Again, plot the training and validation loss and save the model. If all went well, there should be much less overtraining now (validation loss should track training loss).

20. Make another plot comparing the validation loss from three different models: (a) your custom CNN, (b) out-of-the-box ResNet, and (c) your best ResNet with data augmentation and learning rate scheduler.

21. Make a scatter plot comparing the "true" and "predicted" values of all 37 labels for the validation set, using your best model (i.e., the one with the lowest validation loss). As always, be thoughtful about your axis limits, marker sizes, etc. Comment on the model's performance. Are there labels for which the model performed particularly well? Particularly poorly?

    Checkpoint 3: Submit a plot of (a) a PDF of your scatter plot from (22) and (b) a PDF plot showing the training and validation loss vs step number for the three models from (21). Submit this via gradescope.
    It should be a pdf, named Firstname_Lastname_lab3_cp1.pdf

22. For the following 7 labels, plot the images of the 5 objects in the validation set that are the most extreme example of that label (i.e., those that have the highest probability assigned to it). Do this (a) using the actual labels in the validation set, and (b) using the labels predicted by your model. Labels: (1) smooth, (2) star/artifact, (3) edge-on disk, (4) odd: ring, (5) odd: lens/arc, (6) spiral: 2 arms, (7) odd: merger. Comment on the reliability of your model's classifications for each label. If there are classes for which the model performed poorly, comment on what may have led to the poor performance.

23. Now run your trained model on the second set of images (`test_images.tar.gz`, which is available in the Box directory or from the shared directory (`$HOME/shared/ay128-2024/lab3_galaxies`) on Astro Datahub). Estimate the fraction of them that are mergers. How does your estimate compare to the $z \approx 0$ merger rate inferred by Lotz et al? In particular, compare the merger fraction you infer for this sample to the theoretically expected and observed major merger fractions at $z \approx 0$, shown in the upper right panel of their Figure 13. A variety of factors complicate this comparison. Discuss what some of them are and make any appropriate corrections for them. Hint: what are the units of the merger fractions in the paper? Do they differ from yours?