**AI for Star Cluster Analysis Project Resources**

*Please see the project packet for a general project summary.*

**General Goals/Steps**

1. *Data and Environment Set-up*
   1. Get acquainted with using Google Colab IPython notebooks / coding Python in general, and loading Google Drive folders into Colab environments.
      1. See my colab with a few useful tooling examples at <https://colab.research.google.com/drive/1aIADt3r5i3zJP4pM2VywUDSe_ixBACm->
   2. Explore the data and its format.
   3. Get acquainted with the basics of PyTorch: see this tutorial <https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html>. PyTorch is the foremost library for working with neural networks/deep learning.
   4. Set up code for dataset and data loading with PyTorch; start with the preceding tutorial. More info here: <https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#creating-a-custom-dataset-for-your-files>
      1. Remember: you’ll need to split your dataset into the training set, the validation set (for selecting the best mode iteration during training), and the test set (for evaluating the trained model on totally new data to see if it can *generalize*).
      2. **Hint:** for the training set data loader, I highly recommend using **image augmentations**, especially on our (somewhat) small dataset!
      3. Try loading images from your dataloader and plotting them?
   5. Attach your Colab notebook to a GPU for computation, which you will load your neural network and data onto.
2. *Train your neural network for galaxy classification with PyTorch*
   1. Load/create your neural network
      1. **Do you want the network to regress, or classify, age? My suggestion is to start with classification, and maybe try regression later.**
   2. Set up the following for training your model (including but not limited to):
      1. training criterion prediction error measure
      2. optimizer/error minimizer
      3. Set up the saving of your model during training
   3. Train your model! Set up your *training loop* that trains for some number of *epochs* (epochs are passes over the entire training set)
   4. Experiment with different settings for training your model to get the best performance on the test set! Such as:
      1. Learning rate (the “lr” parameter when you load your optimizer)
      2. The resolution that we change the images to before giving to the neural network (higher resolution = more detail for the network to learn from, but also quadratically more computational cost)
      3. Other possibilities… training optimizer type, batch size, network type, using image patches instead of full-size images, etc…
3. *Test the network: basics*
   1. Test your model’s prediction ability on a set of star cluster images from the dataset that were not in your training set.
4. *Test the network: more.* ***Please see the sections below with more details on some of these***
   1. Comparison to human performance: accuracy/error and compute time
   2. What are images that the network did best or worst with?
   3. Does it do better with open or globular clusters? Why?
      1. Bonus if you have time: train a network to classify the cluster type in the image. But these labels may be UNBALANCED.
      2. A better training **LOSS** metric for classification will be weighting each class in the cross entropy loss function, via:

class\_weights = []

for c in CLUSTER\_AGE\_CLASSES:

w = 1. / len(train\_dataset.image\_labels[train\_dataset.image\_labels[label\_name] == c])

class\_weights.append(w)

print(class\_weights)

criterion = torch.nn.CrossEntropyLoss(weight=torch.tensor([class\_weights].squeeze())) if label\_type == 'classification' else torch.nn.MSELoss()

* + 1. A better metric in accuracy may be AUC <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html>. For converting network’s outputted class logits to probabilities, use <https://pytorch.org/docs/stable/generated/torch.nn.functional.softmax.html#torch.nn.functional.softmax>
       1. To convert a torch tensor (**x**) to a numpy array, use **x = x.cpu().numpy().** Because the AUC score function needs numpy arrays.
  1. Incorporate uncertainty into network predictions for \*\*better science\*\*-- talk to Nick about how to do this
  2. Test on *totally* *new* star cluster images -- find them yourself (**Skynet**)! How did the model do?

1. **Advanced cluster analysis** -- we’ll get to this if we have time

**Resources**

Pytorch / deep learning

* **START HERE: Basic PyTorch + deep learning crash course**: <https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html>
* **more in-depth 60 min tutorial (four parts):** good for reference <https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html>
* **Creating a custom PyTorch Dataset** <https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#creating-a-custom-dataset-for-your-files>

**Suggested sub-tasks/groups for the project:** (some will be more or less needed as the project goes on)

1. Data piping (e.g., setting up data pre-processing, data loading, etc.)
2. Network architecture design
3. Training, evaluation and testing loop creation
4. Network hyperparameter optimization: how can you tune training settings (e.g., learning rate, batch size, image augmentations, optimizer and optimizer settings, regularization, etc…) and/or network settings (depth, architecture, etc…) to get better performance?
5. Visualization/results (beyond just basic performance numbers, how else can you showcase how your trained neural network behaves?)
6. Additional testing

* Number of cluster stars vs. field stars prediction?
* Visually, you can see the cluster, but in the histogram, the cluster is hard to see
* Star cluster analysis:
  + A finite set of situations, that could complicate the picture making it harder to do the analysis
* Main thing is field star removal

**Resources: XAI/explainable AI**

* **Saliency maps** (“what regions in the image were most important for a network’s prediction on it?”)
  + This is fairly involved, so I’ve just **implemented as an option** in the code. But the details are:
    1. Pytorch-gradcam library; talk to nick about how to use it <https://github.com/jacobgil/pytorch-grad-cam?tab=readme-ov-file#using-from-code-as-a-library>
  + Learned feature analysis:
    1. A neural network converts (image) data to a feature representation that is learned to be useful for the given task; this feature representation of input data can be extracted from the penultimate layer of the network. **How to do this in pytorch?** Using something called forward hooks - also talk to nick for this
       - <https://discuss.pytorch.org/t/how-can-l-load-my-best-model-as-a-feature-extractor-evaluator/17254/6>
    2. Once we have these high dimensional features, we can visualize them in 2D using **t-SNE** <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>. Some questions:
* Do the learned features naturally cluster training data by class? What about test data?
* In general, how is the way that the distribution of the training data as represented in the network compares to the testing data?
* Lots of other directions here - but let’s start with this