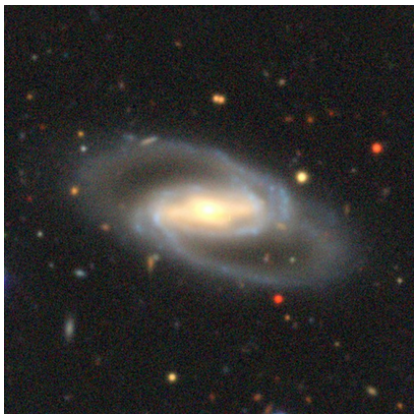
PHYS-467 Assignment 3
Recognizing Galaxies with Deep Neural Networks

Due: Thursday 21 December 2023 at 11.59 pm



Instructions

This assignment focuses on training neural networks using PyTorch. Since training on a GPU is much faster than on a CPU, we recommend using Google Colab. This platform requires no installations, and you can easily request a session with a GPU by navigating to Edit → Notebook Settings, and then selecting GPU from the Hardware Accelerator menu. Just make sure to only connect to the GPU when actually training the networks, since excessive usage of GPU sessions will lead to google blocking this feature for you temporally. Alternatively, you can use your local GPU if it is CUDA-compatible. This requires installing PyTorch and possibly CUDA, depending on your setup. This may or may not be a smooth experience, so consider switching to Colab if there are problems. Installation instructions for CUDA on Linux can be found here.

Your submission should be a single `.ipynb` file uploaded to Moodle. This file should include the working code, plots, results, and discussion of the questions posed. Remember to properly acknowledge any sources or individuals consulted during your work. The assignment provides you with a `.ipynb` skeleton, complete with a pre-defined Dataset class, which will guide you through the data preprocessing stage. It is crucial to use it as it addresses several technical aspects, ensuring a smoother workflow.

Overall plan

Your task is to classify images of galaxies. Why not train a neural network to do it? The Galaxy10 SDSS dataset provides about 20k images of galaxies at `69x69x3` resolution [1], falling into 10 classes. You will first pre-process the data, hold back 2 of the 10 classes for later, and create a PyTorch-compatible dataset object using the predefined data class `MyDataset` we're providing for you. Then, you'll train a fully connected and convolutional network, and compare their performances on the first 8 classes. Finally, you'll implement transfer learning, i.e., using the trained convolutional layers as a fixed feature extractor, you will train a linear layer on top to classify the 2 held-out classes.

Question 1 (5 Pt.)

- Familiarize yourself with the description of the Galaxy10 S data and download the `.h5` file (http://www.astro.utoronto.ca/~bovy/Galaxy10/Galaxy10.h5) to your (Colab) environment. Use h5py to read the file and create a torch.Tensor from the `"images"` data and another for the labels in `"ans"`. Hint: first create numpy arrays from the data and then make torch.Tensors from those, since the direct conversion is very slow [2].

[1] A newer version of this dataset with nicer pictures and more balanced classes can be found here. But it is too big to be easily manipulated with 16 GB RAM, so we use the smaller version.

[2] Because h5py provides lazy access to the data, see related https://github.com/pytorch/pytorch/issues/13918

- There are two technical adjustments necessary to use the data in PyTorch, both are already implemented for you in the skeleton code and described here for completeness:
  i) Changing the type of the image tensor from `uint8` to the expected default `torch.float32`, and the type of the label tensor to `torch.LongTensor`[3].
  ii) Changing the shape of the tensor from (b, h, w, c)[4] to (b, c, h, w), that is moving the color channel axis in front of the height & width axes, since PyTorch expects the (h, w) axes to be last.
- Crop and normalize the images using `torchvision.transforms`. The plan is to define a transform function that will be passed as an argument to the `MyDataset` object (the transform will then be applied to each batch pulled by the data loader). We want the transform to first create a random `48x48` pixel crop of the image and then normalize each color channel. However, `transforms.Normalize()` needs the mean and std of the three channels as arguments. Therefore, first, apply a `transforms.RandomCrop` to `48x48` to a batch of, e.g., 1000 images, and compute mean and standard deviation in each of the color channels. Notice that cropping first is important since it will change the statistics. With these numbers, use `transforms.Compose` to create a new transform function that first applies `RandomCrop` with size `48x48` and then `Normalize`. Later, you can verify that your normalization procedure worked by checking that the normalized data have approximatively zero mean and unitary std.
  Hint: Normalizing is particularly important since the pixel values otherwise vary between 0 and 255. If you have trouble with the transformation, make sure to at least divide the image tensor by 255 to get values in [0,1].
- Split the images into two datasets. The first will be your main dataset and should contain the first 8 classes. The second should contain all images from the last two classes "Disk, Face-on, Medium Spiral" and "Disk, Face-on, Loose Spiral" (labels 8 and 9), which will be held out for the transfer learning task in Question 5. For the held-out data containing classes 8 and 9, change the labels to 0,1 (e.g., by subtracting 8).
  Create `MyDataset` objects for train and test data for both datasets using the pre-implemented class supplied with this assignment. Pass the transform you created before as an argument! For the train-test split, use a training set of 18000 images for the main task and a training set of 1000 images for the transfer-learning task.
  Create train and test data loaders with batch size 64, setting `shuffle=True` for the train data loaders.
  Now it's time to create and train the neural networks!

Question 2 (5 Pt.)

- Define a fully connected network with one hidden layer, using `nn.Sequential` with the components:
  Flatten the image; linear layer to 64 units; ReLU nonlinearity; linear layer to 8 output units.
- Train for 50 epochs with batch size 64 and learning rate 0.001 using the optimizer `torch.optim.Adam` and `nn.CrossEntropyLoss`.
- Log the train and test losses after each epoch and, at the end of the training process, plot them against the number of epochs. Describe their behavior.

Question 3 (5 Pt.)

- Define a convolutional network made of two blocks. The first block will consist of the main CNN – let's call it cnn_backbone – and the second block will consist of a simple linear layer on top – let's call it classifier_head. We are doing this because in Question 5 we will reuse the trained cnn_backbone, but replace the classifier_head with a new one.
  Define the cnn_backbone using `nn.Sequential`:
  - Conv2d layer with 8 output channels and kernel size 5, then ReLU nonlinearity, then `Maxpool2d` with kernel size 2 and stride 2;
  - Conv2d layer with 16 output channels and kernel size 5, then ReLU nonlinearity, then `Maxpool2d` with kernel size 2 and stride 2;
  - Linear layer from 1296 features to 128 features, then ReLU;
  - Linear layer from 128 features to 64 features, then ReLU.

  Define the classifier_head as a single linear layer from 64 to 8 output features.
  Stack the cnn_backbone and the classifier_head blocks using `nn.Sequential` to obtain the full CNN model.

[3] Otherwise, the data type of the network parameters would need to be changed.
[4] Here, b denotes the batch size, h the height of the images, w the width of the images, and c the color channels.

- Train for 50 epochs with batch size 64 and learning rate 0.002 using the optimizer `torch.optim.Adam` and `nn.CrossEntropyLoss`.
- Plot train and test loss curves against the epochs as for the FCN before.
- Compare the final test accuracy of the CNN with the final test accuracy of the FCN and try to explain your results.

Question 4 (2 Pt.)

- Let's delve deeper into the predictions of our models and, in particular, understand how they perform across the 8 different classes. Write a function that given the predictions of a model and the ground truth labels computes the *confusion matrix*. You may use the `confusion_matrix` and `ConfusionMatrixDisplay` functions from `sklearn.metrics`.
- Using the full test set, compute the confusion matrices for both the FCN and the CNN you trained in Questions 2 and 3 respectively.
- Analyze the obtained results. In particular, consider that the training dataset is strongly unbalanced, i.e., the number of training examples strongly varies depending on the class. Firstly, compute the number of training examples belonging to each class. Secondly, observe which classes are more prone to be misclassified and consider why this might be the case.

Question 5 (3 Pt.)

- The aim is now to solve the additional task of distinguishing the two galaxy classes we have held back. Since we have already trained the "big" CNN on the first 8 galaxy classes in Question 4, we would like to reuse this trained model and just fine-tune it to solve this new smaller task. To do so, we seek to reuse the trained cnn_backbone and freeze its weights but put a new classifier_head on top which we train to solve the new downstream task. Think about the big network as so big that it would be expensive and difficult to re-train it, so it is better to use an existing one.
  - Define a new `nn.Sequential` model containing the trained cnn_backbone instance and a new randomly-initialized linear layer going from 64 to 2 output features.
  - Give only the parameters of the final linear layer to the `torch.optim.Adam` optimizer.
  - Tell PyTorch that it does not need to compute the gradients for the cnn_backbone parameters by looping through its `.parameters()` iterator and setting their `requires_grad` flag to False.
- Train for 20 epochs with batch size 64 and learning rate 0.005 using the optimizer `torch.optim.Adam` and `nn.CrossEntropyLoss`. This should be fast. Note that since in this simple case the classification head is just linear, this approach is equivalent to training logistic regression.
- Compute the test accuracy[5].

[5]Notice that this exercise aims to introduce you to the powerful idea of transfer learning. Certainly, the performance in this example could be considerably improved, for instance using a larger convolutional backbone pre-trained on much more data!