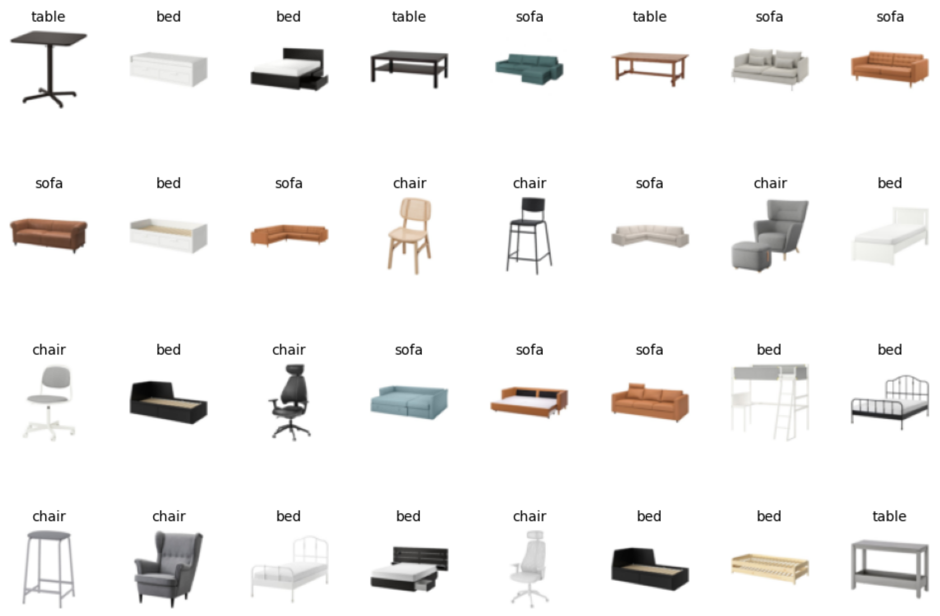
CS 380: Artificial Intelligence

Assignment 4: Deep Learning

IKEA Furniture Classifier (20 pts)

In lecture, we learned about neural networks and deep learning, including classifiers, autoencoders, and stacked autoencoders. In this assignment, we will use stacked autoencoders to build a classifier for images of IKEA furniture.

Our data comprises images taken from IKEA's web site of individual furniture items divided into four classes: beds, chairs, sofas, and tables. Here are some sample images from the data set:



The images are convenient for our purposes because they contain only a single piece of furniture with a white background.

Your challenge is to build a classifier that will take a single image and correctly classify it as one of the four classes. In particular, you are given a sample classifier and autoencoder, and your task is to modify the given models to build stacked autoencoders and finally a classifier for the given images. Note that there are some difficult images in the data set — for example, a futon-like chair that could reasonably be classified as either a chair or a sofa — so we would not expect our classifier to get 100% accuracy; nevertheless, a classifier with an accuracy of 95% or better is very much attainable.

Setup

We will be using several libraries as part of this assignment. Please install the following libraries as part of your Python setup:

```
> python3 -m pip install torch torchinfo torchvision numpy matplotlib
```

Next, please download the base code and explore the contents. The files include:

- **data.py** : Data class to load data, split data (e.g., into training and test sets), etc.
- **model.py** : Model class that wraps pytorch's models into a convenient class
- **classifier.py** : _Classifier class that serves as a handy parent class for all classifiers

- **sample_cl.py** : sample Classifier class that extends _Classifier
- **autoencoder.py** : _Autoencoder class that serves as a parent class for all autoencoders
- **sample_ae.py** : sample Autoencoder class that extends _Autoencoder

The sample classifier (**sample_cl.py**) and autoencoder (**sample_ae.py**) can both be run from the command line with one optional argument, namely the # epochs to train. To train these sample models, you can enter:

```
> python3 sample_cl.py 100
> python3 sample_ae.py 100
```

This will train the model for 100 epochs and save the model in a **models/** directory. (If the file already exists, it will ask if it should overwrite the existing file.) If you would like to re-run the model by loading a pretrained model instead of training, simply call the scripts without an argument:

```
> python3 sample_cl.py
> python3 sample_ae.py
```

Either way (with or without training), the scripts finish by displaying results using the **matplotlib** library — either a sampling of classified images (for the classifier), or a sampling of real images followed by generated images (for the autoencoder). Note that your Python setup must allow matplotlib to open a display window on your machine (like our last assignment).

Autoencoder #1 (AE1)

Once you are familiar with the base code, your next task is to build the first autoencoder for stacking. As you know from lecture, an autoencoder is a network that aims to reproduce the input as output: the **encoder** transforms the input to a latent (hidden) representation, and the **decoder** transforms the latent representation to the output, which has exactly the same form as the input. The sample autoencoder **sample_ae.py** provides a deep autoencoder that you can use as an example. Part of your task will involve breaking up the pieces of an autoencoder like this one into several separate autoencoders, using their encoders to stack together into a classifier.

Copy the sample autoencoder to a new file **ae1.py**, and modify this file to create your first autoencoder with class **AE1**. The encoder should do a single convolution + max-pooling — essentially, the first three layers of the sample autoencoder, with **Conv2d**, **ReLU**, and **MaxPool2d** layers. The decoder should do a single deconvolution, namely with **ConvTranspose2d** and **Sigmoid** layers. Please keep the same image size as the sample (3x64x64) and the same number of kernels (64). You will need to make sure of several things as you build the autoencoder: that the encoder and decoder "fit" together in terms of shape (i.e., the output of the encoder should be the same shape as the input to the decoder); the **input_shape** parameter is correct for each model; and the output of the decoder matches the input of the encoder (since the input and output of the autoencoder must have the same shape).

When you're done constructing this autoencoder, it should be trainable as follows:

```
> python3 ae1.py 100
```

Please be sure to change the path in **__main__** to **models/ae1.pt**, which will save the AE1 model in the **models/** directory.

Autoencoder #2 (AE2)

Repeat the process above to create a second autoencoder, **AE2**. Before coding AE2, it's important to understand what we're trying to do. You've already trained AE1 to be a reasonably good autoencoder; however, as mentioned in class, what we really care about is AE1's **encoder**, not decoder — because AE1's encoder generates a good compact representation of the input, we want to take the output of AE1's encoder and feed this output into AE2.

To build AE2, copy **ae1.py** to **ae2.py** and flesh out this second autoencoder. The architecture of this network is similar to AE1: The encoder should again have a single convolution, with **Conv2d**, **ReLU**, and **MaxPool2d** layers. The decoder should again have a single deconvolution; this time, after the **ConvTranspose2d** layer, you should use **ReLU** (instead of **Sigmoid**). Note that, because of the input of AE2 is the output of AE1's encoder, the shape of AE2's input and output is the same as the latent representation in AE1.

The training of AE2 is just like AE1, with one catch: you need to run the data through AE1's encoder before sending on to AE2. To do this, in the main section of the code, you should load the AE1 autoencoder and use its **encode(data)** method to run the data through its encoder and return a new transformed data set.

Autoencoder #3 (AE3)

Repeat the process above yet again to create a third autoencoder, **AE3**. AE3 will have the same layers as AE2. This time, though, you'll need to load both AE1 and AE2, and pass the original data through AE1's encoder and then AE2's encoder to transform it for input into AE3.

Classifier #1 (CL1)

At this point, you have 3 trained autoencoders (AE1, AE2, AE3) that successively extract features from the data. We now want to stack these in sequence and add a classification network to build our final classifier.

Start your classifier **CL1** by copying **sample_cl.py** into a new file **cl1.py**. You will need to modify this file as follows. First, it should load the three trained autoencoders. Next, it should pass these autoencoders to the **__init__()** method of the classifier to embed the autoencoders as the first layers of the network; of course, it should include *only* their encoders, not their decoders (we no longer need the decoders at this point). After the encoding layers, please add the following layers: **Flatten**, **Dropout(p=0.1)**, **Linear**, **ReLU**, **Dropout(p=0.1)**, **Linear**, **ReLU**, **Linear**. The first **Linear** layer should bring the flattened network down to 256 units; the second, to 64 units; and the third, to 4 units (equal to our number of furniture classes).

(Subtle note: Sometimes with a stacked autoencoder, we might freeze the pretrained encoding layers before training. In this assignment, we will leave them unfrozen — that is, when training the classifier, all the layers will be (re-)trained, including the pretrained encoding layers. There are advantages and disadvantages to this approach, especially depending on the size and complexity of the data set, but we will assume a completely unfrozen network here.)

Your final classifier should be trained roughly 100 epochs (like the autoencoders) and should perform well, with a final classification accuracy >95% across all the data. (Naturally, we expect that the model will do better on the training data and worse on the test data; the total accuracy above, which is built into the base code, is computed from the entire data set.)

Submission

For this assignment, you should submit all your files in the entire directory. You can omit the data files if you'd like, of course, but please submit everything else. This includes:

- The code for your autoencoders: **ae1.py**, **ae2.py**, **ae3.py**
- The code for your final classifier: **cl1.py**
- The saved models in **models/**: **ae1.pt**, **ae2.pt**, **ae3.pt**, **cl1.pt**
- Plus all given base code files (except **data/**)

In testing your submission, we will run the saved models and will also re-run the training, so we would like everything together in one package.

Academic Honesty

Please remember that all material submitted for this assignment (and all assignments for this course) must be created by you, on your own, without help from anyone except the course TA(s) or instructor(s). Any material taken from outside sources must be appropriately cited.