B657 Assignment 2: Image transformations and deep learning

Spring 2024
Due: Friday April 12, 11:59PM
Late deadline: Sunday April 14, 11:59PM (with 10% grade penalty)

This assignment will provide you experience in thinking about image transformations, as well as building and evaluating deep learning models using PyTorch. It will also get you thinking about the limits of deep learning methods and the importance of using the "right" algorithm or architecture for the task at hand.

Coding guidelines. For fairness and efficiency, we use an automatic program to grade your submissions. This means you must write your code carefully so that our program can run your code and understand its output properly. In particular:

1. You must code this assignment in Python 3.
2. Be sure to use the program file names we specify.
3. Use the skeleton code we provide, and follow the instructions in the skeleton code (e.g., do not change the parameters of any functions).
4. You must use pytorch (`https://pytorch.org/`) for building your deep learning models.

Teams. We've assigned you to a group of 1-3 people according to the preferences you listed on our team request form. See below for how to check your assigned team. You should only submit one copy of the assignment for your team, through GitHub. All the people on the team will receive the same grade, except in unusual circumstances. Everyone is expected to actively contribute to the assignment; please note that **the project report must include a section indicating how each person contributed to the assignment** (see Report section below). After the assignment, we will also collect confidential information from all teammates about everyone's contributions to the project.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

Part 0: Getting Started

1. You can find your assigned teammate(s) by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b657-sp2024`. Then in the box below, you should see a repository called *userid1-userid2-userid3*-a2 or *userid1-userid2*-a2, where the other user ID(s) corresponds to your teammates.

2. While you may want to do your development work on a local machine (e.g. your laptop), remember that the code will be tested and thus must run on `burrow.luddy.indiana.edu`. After logging on to that machine via ssh, clone the github repository via one of these two commands:

   `git clone https://github.iu.edu/cs-b657-sp2024/`*your-repo-name*`-a2`
   `git clone git@github.iu.edu:cs-b657-sp2024/`*your-repo-name*`-a2`

   where *your-repo-name* is the one you found on the GitHub website above. (If this command doesn't work, you probably need to set up IU GitHub ssh keys. See Canvas A1 page for help.) This should fetch some test images and some sample code.

Part 1: The shuffletruffle ruffle

In a dimly lit research lab nestled within the heart of a bustling metropolis, a team of scientists embark on an extraordinary mission: to unravel the enigma of alien vision. Their obsession lies with a peculiar species from a distant galaxy, known for their shuffled eyes – an anatomical marvel unlike anything encountered on Earth. The aliens, dubbed the shuffletruffle, possess eyes that rearrange their individual lenses randomly at random times once they turned 20 years old. Strangely, this doesn't impact their day-to-day performance, granting them unparalleled visual versatility (or perhaps, great difficulty — it's hard to tell). So far, the scientists have discovered that the size of their receptive fields beyond their teenage years is influenced by the time of the day they were born. Being born during the day leads to a larger receptive field (we will refer to them as D-shuffletruffle), while being born at night results in a smaller receptive field (N-shuffletruffle). Given that the best way to understand something is to build it, the scientists need help from us to come up with models that mimic the visual behavior of the aliens.

We will be using CIFAR10 dataset for experimentation, which has 50,000 training images and 10,000 test images, where each image is a 32x32 RGB image (see Figure 1(a)). The dataset can be loaded directly using the PyTorch library (provided in the starter code). We have also provided a dataset class for loading other datasets provided as .npz files.

Here's what to do.

1. Create a training and validation set of 40,000 and 10,000 data points, respectively, and normalize the data using the mean and standard deviation values calculated from the training set.

2. **Best Performing Model for CIFAR10:**

   (a) Implement a deep neural network and train your model using the training and validation set created in the previous step. Evaluate the model using the validation set at a fixed number of epochs during training, and save necessary values for two separate plots: training loss and validation loss across epochs, and training accuracy and validation accuracy across epochs. (Feel free to use early stopping, learning rate schedules, etc.)

   (b) On completion of training, evaluate your model on the test set and record the results. Provide a brief account of your experiments in the report for all the models you tried building. (e.g., model name and corresponding architectures, a table of test accuracy and loss values for the models, separate plots comparing validation loss and accuracy across epochs for different models, etc.). Save the model weights of the best performing model for analysis later.

3. **D-shuffletruffle Model:**

   (a) The odd thing about shuffletruffles is that they learn to see the world based on viewing the world in one way, but then at age 20 their vision suddenly changes. This means their "training data" looks very different from the "test data" they experience after age 20. To simulate this, train the

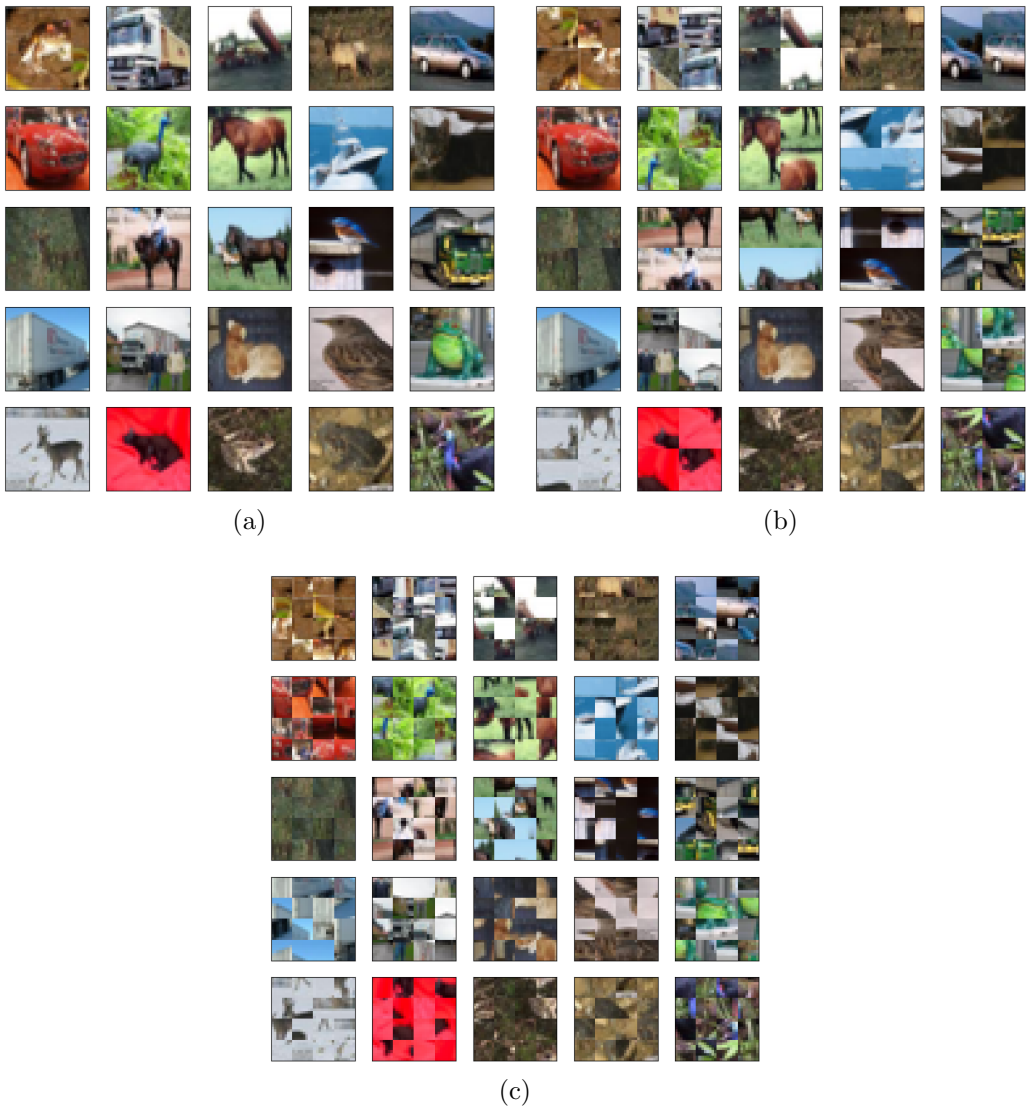

Figure 1: (a) CIFAR10 images; (b) D-shuffletruffle test set images; (c) N-shuffletruffle test set images.

same model above using the same training set and validation set used in the previous step but evaluate the model on the test set provided in test-patch-16.npz, which contains shuffled images like what they see after their 20th birthday. This test set contains images that have randomly shuffled 16×16 patches (see Figure 1(b)).

(b) As the model has not see examples with randomly shuffled patches during training or validation, your results in (a) are probably quite bad. Modify your model with an architecture that allows the performance of the model on the original CIFAR10 test set and on the test set retrieved from test-patch-16.npz to be similar (ideally <1% difference in accuracy). (Do not use data augmentation involving shuffling images; the point is to change the model architecture so that it is invariant to shuffles, not to change the training data.)

(c) Provide a brief account of your experiments in the report for all the models you tried building, including model name and corresponding architectures, a table of test accuracy and loss values

for the models, separate plots comparing validation loss and accuracy across epochs for different models, etc.

4. **N-shuffletruffle Model:**
   - Follow the same steps as above for the test set provided in test-patch-8.npz, which contains images that have randomly shuffled 8×8 patches (see Figure 1(c)).
5. **Analysis:**
   (a) In your report, including a table comparing the test accuracy and loss of the three models from the previous steps for the three test sets (original CIFAR10 test set, test-patch-16.npz and test-patch-8.npz).
   (b) Collect 25 unique examples from the original test set and label them 1 to 25. Create a small dataset of 75 examples by randomly shuffling 16x16 and 8x8 patches for each image without changing the labels.
   (c) For each model amongst the three best models, run principal component analysis (PCA) using the pre-final layer embeddings of the model with n_dim=2. (You can use a package for computing PCA and you don't have to submit source code for this.)
   (d) Transform and plot (in 2 dimensions) the collected data points with their respective labels for the three models.
   (e) What insights can you draw from the results? Are there any other conceptually different model architectures for d/n-shuffletruffle models that you think can provide the same accuracy on both the original and their respective test set?

(Thanks to Sahaj Singh Maini for this problem.)

What to turn in

You should submit: (1) Your source code for all three parts, and (2) Your report, as a Readme.md file in Github. To submit, simply put the finished version in your GitHub repository (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.