Assignment 2
Parallel deep neural networks and Data Augmentation

Due Date: 31/12/2018 23:59

Questions:

- Questions regarding this assignment should only be sent to Anny.
- Problems with connecting to servers should only be sent to Or.
- Postponements can only be authorized by the TA in charge Ido.

Changes are marked with yellow.
Part 1

Brief Background:

Deep artificial neural networks require a large corpus of training data in order to effectively learn. But every data collection process is associated with a cost. This cost can be in terms of dollars, human effort, computational resources and of course time consumed in the process. Therefore, we may need to augment existing data to increase the data size that we feed to our network and to compensate for the cost involved in further data collection.

Data augmentation is the creation of altered copies of each instance within a training dataset.

There are many ways to augment data. In images, you can rotate the original image, change lighting conditions, crop it differently, so with one image you can generate many different sub-samples.

Implementation:

As a continuation of the last exercise (Ex 1.), in part 1, you will implement two main classes, which are:

- **Preprocessor**
  - Inherits from Process (from the multiprocessing library in python)
  - Implement multiple data augmentation functions
  - Implement a producer function

- **IPNeuralNetwork**
  - Inherits from NeuralNetwork
  - Use multiple preprocessors to create batches
Worker implementation in preprocess.py:

```python
def __init__(self, jobs, result):
    # Initializes the class and its members as you think will help you in this exercise. Note you may add more arguments to the function.
```

```python
def rotate(image, angle):
    # Rotate the image by the given angle. (Hint: You should look at scipy library)
    # Example for using the rotate function:
```

![Image](image1.png)

```python
def shift(image, dx, dy):
    # Shift the given image by dx cells to the left and dy cells to up.
    # If the coordinates are out of range replace it with black (which is 0).
    # You may only use numpy library in the function. Example for using the shift function:
```

![Image](image2.png)

```python
def step_func(image, steps):
    # Step function is a function that increases or decreases abruptly from one constant value to another.
```

![Image](image3.png)
You should manipulate every pixel according to the step function that matches the given argument.

The argument "steps" acts as the number of steps in the range \([0-1]\) increasing in a steady interval. In the general case: 
\[
f(x) = \frac{1}{steps} \lfloor steps \cdot x \rfloor
\]

You may only use numpy library in the function.

Example: if \(steps = 4\) then the function should hold that:

\[
f(x) = \begin{cases} 
0, & 0 \leq x < 0.25 \\
\frac{1}{3}, & 0.25 \leq x < 0.5 \\
\frac{2}{3}, & 0.5 \leq x < 0.75 \\
1, & 0.75 \leq x < 1 
\end{cases}
\]

---

**def skew(image, tilt):**

Manipulate image by given tilt.

For your convenience, we give you a skew formula to use:

\[
SkewedImage[y][x] = Image[y][x + y \cdot tilt]
\]

If the coordinates are out of range replace it with black.

You may only use numpy library in the function.

Example for using skew function:
def process_image(self, image):
    Run the previous functions in order to manipulate the given image. You should use the functions you implemented before (in any order you would like).
    Use python’s random to decide on arguments for the functions.
    Try to get better accuracies when using the augmented data.

def run(self):
    The function should produce augmented images.

IPNeuralNetwork Implementation ip_network.py:

def create_batches(self, data, labels, batch_size):
    You should override this function, so it will collect batches of augmented images from workers, instead of creating them.

def fit(self, training_data, validation_data=None):
    You should override this function, so it will create Preprocessor workers depending on the number of CPUs used before running the super function and destroy them after.
    Trivial solution with only one Worker or a Worker for every single job will not get full points.

Important Notices:

You must to add utils.py from your previous HW to your working directory to run NeuralNetwork, but you should not submit it again!

You were provided an implementation of NeuralNetwork with some changes to the structure of the class.

They are listed below:
- The class NeuralNetwork can be initialized with number_of_batches.
- The function create_batches() is a class function (Instead of being in Utils file).
- create_batches() implementation in NerualNetwork is returning random batches according to the number_of_batches unlike the definition given in exercise.
Part 2

In this part, you will implement a simple queue and replace the result queue in part 1 with this implementation (you don’t need to replace jobs queue!).

You will have to use Pipe and Lock which you have seen in class. The class needs to work on multiple writers and one reader. Think where you need to synchronize and were you can do without.

In my_queue.py implement the following methods of class MyQueue:

```python
def __init__(self):
    Initialize the queue and it’s members.

def put(self, msg):
    Send a msg through a pipe.

def get(self):
    Read a msg from a pipe.
```
Part 3

Convolutions

In image processing, a kernel is a small matrix. It is used for blurring, sharpening, embossing, edge detection and more. This is accomplished by doing a **convolution** between a kernel and an image.

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel. This is related to a form of mathematical convolution.

For example, if we have two three-by-three matrices, the first a kernel, and the second an image piece, convolution is the process of flipping both the rows and columns of the kernel and then multiplying locally similar entries and summing. The element at coordinates [2, 2] (that is, the central element) of the resulting image would be a weighted combination of all the entries of the image matrix, with weights given by the kernel:

\[
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i \\
\end{bmatrix} \cdot 
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\begin{bmatrix}
2 & 2 \\
\end{bmatrix} = (a \cdot 1) + (b \cdot 2) + (c \cdot 3) + (d \cdot 4) + (e \cdot 5) + (f \cdot 6) + (g \cdot 7) + (h \cdot 8) + (i \cdot 9).
\]

The other entries would be similarly weighted, where we position the center of the kernel on each of the boundary points of the image and compute a weighted sum.

The values of a given pixel in the output image are calculated by multiplying each kernel value by the corresponding input image pixel values.

You should read about convolutions [here](#).

Implementations in filters.py:

In this part you will implement convolution, once using **njit** to speedup calculations and another time using GPU’s.

1. **def** convolution_numba(kernel, image):
   - Implement the function using **njit** to speed up the calculation
2. **def** convolution_gpu(kernel, image):
   - Implement the function using **cuda.jit**

Make sure that the results of gpu and numba calculations are equal to scipy’s convolve2d.
Report

Part 1

1. Run main.py with [8, 16, 32] cores (flag -c<core_number>).
2. Compare runtime of IPNerualNetwork between different core numbers. Include a screenshot and a short explanation about what number of cores gave the best performance.
3. Run main.py again. Compare the accuracy percentage for different epochs between NeuralNetwork and IPNeuralNetwork. Include a comparison table and an explanation.
4. Answer: Why are we using processes and not threads?

Part 2

5. Explain your implementation in part 2 and the reason you decided to implement it that way.

Part 3

6. Give a detailed explanation of your convolution_numba and convolution_gpu implementation.
7. Run filters_test.py on 1 core (flag -c1) to see time comparison.
8. Include screenshot and calculate the speedup between them and the scipy’s convolve2d.
9. Answer: what will happen if we use a larger kernel?

Notes and Tips

1. Notice that in part 1, the image is a numpy array of size 784 while you should manipulate it as a matrix of size 28X28.
2. Notice that in part 1, rotate, shift, step_func and skew are static functions.
3. In filters_test.py there is a function called show_image(image). It will help you understand what your filters are actually doing.
4. You can add variables and prints as you need, but your code must be clear and organized.
5. Don’t remove prints or comments already in the code, adhere to instruction comments.
7. It’s recommended that you work with PyCharm, but performance should only be measured in the course server, you can simulate a gpu by setting the environment variable NUMBA_ENABLE_CUDASIM to 1, but take into consideration that it will be very very slow.
**Server**

Connecting to the course server:

1. ssh to rishon server using your name: ssh -X user@rishon.cs.technion.ac.il
2. ask for a bash shell and then start working: srun -p 236370 -c1 –gres=gpu:1 –pty bash

For more info:

[https://hpc.cs.wp.cs.technion.ac.il/2018/04/09/%D7%9E%D7%93%D7%A8%D7%99%D7%9A-%D7%9C%D7%9E%D7%A9%D7%AA%D7%9E%D7%A9-2366056-%D7%90%D7%91%D7%99%D7%91-2018/](https://hpc.cs.wp.cs.technion.ac.il/2018/04/09/%D7%9E%D7%93%D7%A8%D7%99%D7%9A-%D7%9C%D7%9E%D7%A9%D7%AA%D7%9E%D7%A9-2366056-%D7%90%D7%91%D7%99%D7%91-2018/)

Install the necessary packages before you start working (through bash shell, only once):

pip3 install --user --upgrade pip

pip3 install --user numpy numba

Notice the extra packages from HW1:

pip3 install --user matplotlib

pip3 install --user scipy

**Submission**

Submit a hw1.zip with the following files only:

1. preprocessor.py with your implementation.
2. ip_network.py with your implementation.
3. my_queue.py with your implementation.
4. filters.py with your implementations.