Assignment 3

Key Information

Overview

Objective

- Work individually.
- Develop and unit test a generative program with a text-based file IO (see tasks in next pages).
- Practice the basic programming concepts from weeks 1-12.
- Download the following scaffold to populate with your solutions:

A3_Scaffold.zip

Please read the individual docstrings for further details.

Submission Requirements

- You will work on each task as different functions & classes in your Ed workspace.
- You must make the final submission on Moodle.
- Your final submission must be one zip file with the necessary and sufficient files, as shown below.

```
A3_<student_id>.zip
└─> ai.py
└─> biases.txt
└─> generative.py
└─> image.txt
└─> test_generative.py
└─> weights.txt
```

Make sure that all functions are in the file they need to be in, with appropriate function names and arguments.

- As far as external modules, you are only allowed to import and use `unittest` in the file named `test_generative.py`.
- You are allowed (and expected) to import and use the `ai` module that is provided to you in the scaffold. However, you are not allowed to modify the `ai` module.
- You are allowed (and expected) to import and use your own `generative` module in the file named `test_generative.py`.

Documentation

In this assignment, some documentation is provided as part of the scaffold code. However, any function, method, class or file you add must have docstrings.

In-line comments

You are expected to use in-line comments throughout your program to make sure it is clear to the reader. The amount of comments necessary depends on how clear the code is by itself. Therefore, a good habit is to write clear code, so that the amount of in-line comments required is reduced. This will also reduce the number of bugs created.

Comment your code as you write it rather than after reaching a final solution, as it's a lot easier to remember why certain decisions were made and the thought process behind them when the code is fresh in your mind.

Ensure your variable names are meaningful and concise so that your code is understandable at first read. When a reader is going through your program, it should read like a story that is self-explanatory, using in-line comments to explain the logic behind blocks of code and any additional information that is relevant to the functionality of the code.

Avoid repeating yourself by commenting on lines of code that are fairly self-explanatory, and instead reserve your comments to explain higher level logic, such as explaining the overall purpose behind a small block of code.

Recommended Approach

Testing your code:

- The assignment is broken up into individual coding tasks.
- You can code and test using the Ed lessons.
- These lessons are **not** to submit your work, only to test your code against some basic automated checks.

Scaffold code:

- All tasks start with scaffold code.
- Make sure you do not modify the variable, function, method and class names; these will be needed to pass automated tests. Your TA will rely on these existing to understand your program and mark efficiently.

Academic Integrity

- Your code will be checked against all other students' with a similarity checker.
- Anything you are able to google can be easily found by the teaching team and compared against your work.
- In this assessment, you must not use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task. This includes ChatGPT. Obviously, this does not include the use of the Python file named `generative.py` that you will create for this assignment.

How can I avoid academic integrity issues?

- Never copy code from anywhere. If you learn something useful online, rewrite it from scratch. It's also the best way to make sure you have understood it.
- If a fellow student asks you for a solution to a question, try to help them build their solution. Do **not** give them yours.
- Giving your solution is just as much of an Academic Integrity breach as receiving it.
- If you feel like you physically cannot submit the assignment on time, please submit an extension request, and seek help.
- For extensions, see Moodle AU or Moodle MA.
- For help, see Moodle AU or Moodle MA.

How will I be assessed?

Your assignment mark will come from:

1. your *submission,*
2. your *interview colour.*

There will be an interview scheduled with a TA (Teaching Assistant) after the submission deadline, from which you will get a colour.

1. Submission

This comes from a review of your submission by a TA, following a rubric. For this assignment, we are providing a less detailed rubric.

Note that automated tests do not contribute to the mark. They are here to help you check the correctness of your programs. But a program that passes all tests can still be incorrect. Furthermore, test cases do not evaluate the readability of your code, which our TAs will assess.

2. Interview colour

After your individual interview with your tutor, you will be assigned one of two colours.

- **Green (1)**
  - If you attend the interview and answer questions to the best of your ability.
- **Black (0.1)**
  - Otherwise, i.e. you do not attend the interview or do not answer questions to the best of your ability.
  - Otherwise. Note that you can apply for special consideration (see how to apply for a short extension on Moodle).

The purpose of the interview is to help us determine whether you have engaged in academic misconduct, and produce a recording which may be used as evidence as part of an Academic Integrity investigation. We will be using this in conjunction with code similarity detection tools.

Because we stipulate that you must attempt the interview to the best of your abilities, you cannot get away from an academic integrity investigation that uses the interview as evidence by saying you weren't answering questions to the best of your abilities.

Late penalties

Up to 7 days

- 10% lost per calendar day (or part thereof).

More than 7 days

- Zero (0) marks with no feedback given.

See on Moodle AU or Moodle MA for how to apply for special consideration.

Overall mark (individual)

Your overall mark for the assignment will be computed according to the program below.

```
max_mark = 100

submission_mark = float(input("Submission mark [0, {}] = ".format(max_mark)))
interview_factor = float(input("Interview coefficient {0.1, 1} = "))
late_days = int(input("Days late {0, 1, ...} = "))

mark = submission_mark * interview_factor

if late_days <= 7:
    late_penalty = 10 * late_days
else:
    late_penalty = max_mark

final_mark = max(0, mark - late_penalty)

print("The final mark is", final_mark)
```

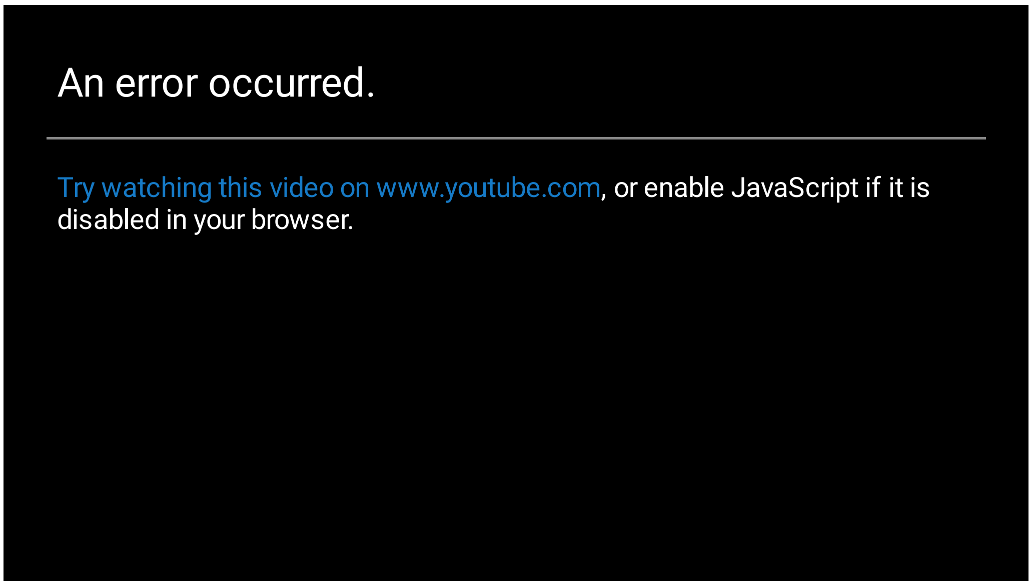
Project description

Generative AI for Images

The high-level description of A3 is provided below.

Introduction and Motivation

The problem of generating new data based on a given input is a hard one – and is one of the fundamental areas of research in Artificial Intelligence (also known as generative AI). The video that is provided below is a fun use of a generative AI which outputs a video (i.e., a sequence of images and an audio) given some input image(s) and prompt(s).



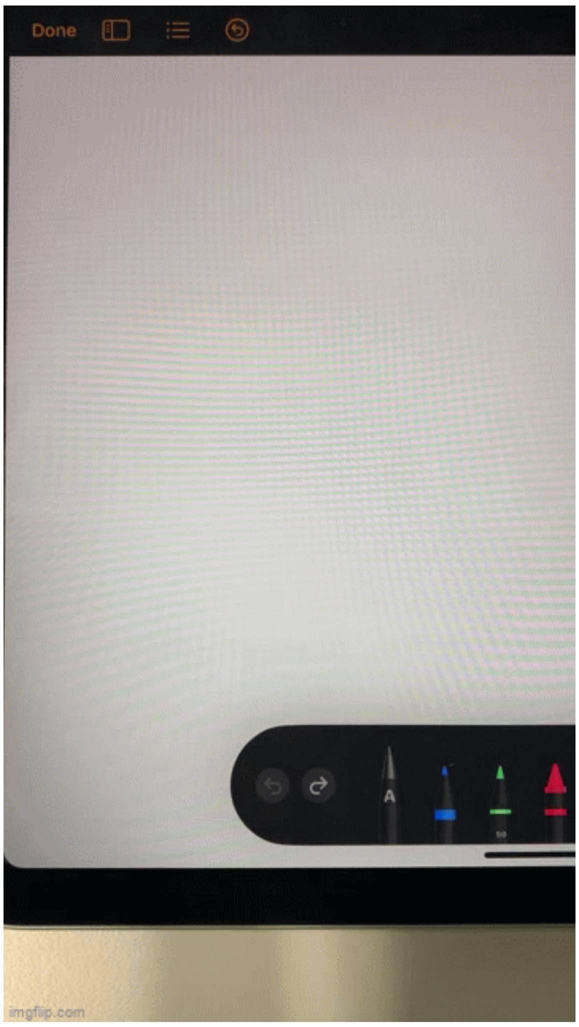
An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

This project aims to provide students with a hands-on experience in image generation by implementing a recursive algorithm that generates new images from an existing one by flipping pixel values under certain constraints.

Background

In W9 Workshop, you have used the `predict_number` function that is capable of predicting the number in a given image. The video that is provided below demonstrates how the `predict_number` function (or its equivalent) is used inside a tablet to predict the number in a hand-drawn image.



Let us also remind ourselves how to use the `predict_number` function.

```
from ai import predict_number

image = [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]

number = predict_number(image)
print("The predicted number is", number)
# The predicted number is 4
```

In the example provided above, the `predict_number` function will output `4` as the number it predicts in the input `image`.

Overall Task

The project consists of the following two main tasks.

1. Design a recursive program that inputs an image of a number (i.e., predicted by the `predict_number` function), and outputs all the possible new unique image(s) of the same number (i.e., also predicted by the `predict_number` function) that can be generated by flipping the pixels of the input image from `0` to `1` under the following two constraints:
    - the number of pixels that can be flipped per new image is limited by a budget, and
    - a pixel can only be flipped if it has an adjacent pixel in the input image with the value of `1`.
2. Verify the correctness of the designed program using unit testing.

ME (A FIT1045 ENJOYER) WHEN I REALISE
THAT A3 IS ACTUALLY ABOUT THE WORKSHOPS
imgflip.com

Task 1: Flatten and unflatten images

The images we are working with are represented by a 2D list (i.e., a list of lists of zeros and ones) of 28x28 dimensions. In order to work with these lists, we need to be able to i) flatten the 2D representation and ii) unflatten the flattened representation.

For example, consider a smaller 5x5 2D list (i.e., `image`) that depicts a cross:

```
image = [
    [1, 0, 0, 0, 1],
    [0, 1, 0, 1, 0],
    [0, 0, 1, 0, 0],
    [0, 1, 0, 1, 0],
    [1, 0, 0, 0, 1]
]
```

The flattened version of this list (i.e., `flattened_image`) would be:

```
flattened_image = [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]
```

The function `flatten_image(image)` has been provided for you as part of the assignment scaffold.

Task 1:

Create a function `unflatten_image(flat_image)` that takes in a 1D list of any size (assuming it is a valid flattened representation) and returns the 2D representation.

Hint: Given you are not allowed to import and use the `math` module, you can write `x**0.5` instead of `math.sqrt(x)` in order to take the square root of `x`.

Please note that you must not assume the size of input images, only that they will be *n* x *n* where *n* is a positive integer.

Task 2: Checking for an adjacent 1

The generation of new image(s) from an input image of the same number will be done by flipping pixels from 0s to 1s under certain constraints. One of those constraints is that a pixel can only be flipped from 0 to 1, and such a flip is only allowed if there is another adjacent pixel in the input image with the value of 1. In order to do this, we need a function that checks if a pixel has an adjacent pixel with the value of 1 in the input image. Here, adjacency is defined as the four orthogonal directions (i.e., up, down, left, right) which does not include the diagonals.

Consider the example `image` below that creates an "L" shape, with a focus on two pixels of the value `0` that have been highlighted in bold:

```
image = [
    [1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1]
]
```

The pixel with the indices (1,3) of `image` does not have any pixels with the value of 1 adjacent to it whereas the pixel with the indices (3,2) does.

Note that in the flattened representation of `image` (i.e., `flat_image`), the two pixels with indices (1,3) and (3,2) correspond to the indices 8 and 17, respectively:

```
flat_image = [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

Task 2:

Create a function `check_adjacent_for_one(flat_image, flat_pixel)` that checks if a given pixel in a flattened image has an adjacent pixel with the value of 1.

Remember to make use of your previously written functions `flatten_image` and/or `unflatten_image` to achieve this task.

⚠ Please note that you must not assume the size of input lists, only that they will be *n* x *n* where *n* is a positive integer.

End of Week 10 Checkpoint

We recommend you complete at least Tasks 1 and 2 by the end of week 10.

Your TA will check the progress of your team during your applied class.

Task 3: Recursively flipping pixels

In order to generate new image(s), we need a function that can generate all possible new unique images from the input image by the means of flipping pixels from 0s to 1s under the following two constraints:

1. a pixel can only be flipped from 0 to 1, and such a flip is only allowed if there is another adjacent pixel in the input image with the value of 1, and
2. the number of pixels that can be flipped is limited by a budget.

For example, consider the 3x3 example `image` below depicting an "L" shape:

```
image = [
    [1, 0, 0],
    [1, 0, 0],
    [1, 1, 1]
]
```

One possible `image` that satisfies with our adjacency constraint is:

```
[1, 0, 0]
[1, 0, 1]
[1, 1, 1]
```

Only one pixel is flipped in this example. Note that even if our budget was 2, as long as the number of pixels flipped is less than or equal to the budget, it is considered a valid possibility.

The below possible `image` contains two flips:

```
[1, 0, 0]
[1, 1, 1]
[1, 1, 1]
```

Note that in this example, the top right pixel should never be flipped as it was not adjacent to any pixel of value 1 in the original input image.

Finally, please note that you **must** use recursion to accomplish this task for full marks. If you do not use recursion (i.e., implement a solution that is not based on recursion), you can achieve up to half of the marks.

Task 3:

Create a function `pixel_flip(lst, orig_lst, budget, results, i=0)` that uses recursion to generate all possible new unique images from the input `orig_lst`, following these rules:

- The input `lst` is the current list being processed. Initially, this will be the same as `orig_lst` which is the original flattened image.
- The input `budget` represents the number of pixels that can still be flipped. When the budget reaches 0, no more pixels can be flipped.
- The input `results` is a list of resulting flattened images with flipped pixels. Initially, this will be an empty list.
- The input `i` represents the index of the pixel being processed, by default set to `0`, which is used to drive the recursive function towards its base case (i.e., initially starting from `i=0`).

At termination of the function, the argument `results` should contain all possibilities of the input `orig_lst` by only flipping pixels from 0 to 1 under both the budget and the adjacency constraints.

Please note that you must not assume the size of input images, only that they will be *n* x *n* where *n* is a positive integer.

Automated tests for this task will take some time to execute as they will also test your solution on a 28x28 image. Please simply re-run the marker if you receive the message 'Your program ran for too long'.

You can use `read_image` from the `ai` module provided along with the example images to try `pixel_flip` on a full size 28x28 image.

Task 4: Writing images to a file

Now that we have the capability of generating new images, we can begin working on storing the generated images as text files. In order to do so, we need the capability of writing the newly generated images to a file with a marker indicating the flipped pixel(s).

Reading images can be done using the function `read_image` within the provided `ai` module.

Task 4:

Create a function `write_image(orig_image, new_image, file_name)` that takes a 2D list of integers representing the original image (i.e., `orig_image`), a 2D list of integers representing a newly generated image (i.e., `new_image`), and a string representing the filename for the output file (i.e., `file_name`). The function should write the new image into the output file, marking the modified pixels with `'X'` to indicate the difference(s) from the original image.

Please note that you must not assume the size of input images, only that they will be *n* x *n* where *n* is a positive integer.

End of Week 11 Checkpoint

We recommend you complete at least Tasks 3 and 4 by the end of week 11.

Your TA will check the progress of your team during your applied class.

Task 5: Generating new images

In this task, you will make use of the functions you have previously created in order to define a new function which will generate all possible new unique images from the input image with the same predicted number (i.e., based on the prediction of the `predict_number` function) that satisfy the previously specified budget and adjacency constraints.

The provided `ai.py` module contains the `predict_number` function that accepts as an argument a 2D list of integers representing an image and returns an integer for the number it predicts the image represents.

Task 5:

Create a function `generate_new_images(image, budget)` that takes a 2D list of integers representing an image (i.e., `image`), and an integer representing the number of pixels that can be flipped (i.e., `budget`).

The function should input a 2D list and generate all possible new unique images from the input image under the previously specified `budget` and adjacency constraints (i.e., using the previously implemented `pixel_flip` and `check_adjacent_for_one` functions) while ensuring that the number predicted in the generated images are the same as that of the original input image (i.e., based on the prediction of the `predict_number` function).

The function should return the final list of generated images.

Remember to use `predict_number` from `ai` to test the prediction for each generated image.

Remember to flatten/unflatten images when needed.

Please note that this task does not have any automated test cases, you will get a chance to test your solution during the final task.

Task 6: Testing generated images

For the final task, we are going to create a unit test to verify the correctness of both our solutions and the generated images, and ensure that they comply with our adjacency and budget constraints. We will do this by implementing each test case as a method in our unit test in the file named `test_generative.py`.

Task 6:

Using assertions and the `unittest` module, please create a test case for each of the testing criteria that is presented below:

- Verify output of `flatten_image` for at least three different sizes of images.
- Verify output of `unflatten_image` for at least three different sizes of flattened images.
- Verify output of `check_adjacent_for_one` for three different pixel indexes of an image representing different scenarios (such as in the middle of an image, on the edge and in a corner).
- Verify output of `pixel_flip` for a 5x5 image with a budget of 2. Ensure that the image has at least 8 possible locations where a 0 can be flipped to a 1.
- Verify output of `generate_new_images` with the input `image.txt` such that each output image satisfies:
    - image is of size 28x28,
    - all values in the generated image are either 0s or 1s,
    - the number of pixels flipped from the original input image is within the specified budget,
    - all pixels flipped from the original image had an adjacent value of 1 in the input image.

Each test case's name must begin with `test_` in order for it to be executed under `unittest`.

Given the purpose of this task is to test the correctness of your code using unit testing, there are no automarkers for this task. Please note that the unit test can take a long time (e.g., 90 seconds or more).

End of Week 12 Checkpoint

We recommend you complete Tasks 1 to 6 by the end of week 12.

Your TA will check the progress of your team during your applied class.

Marking Rubric

1 Automatic Zoom

Changelog (5 changes)

08/05/2023 - change 1

Fixed the function signature of Task 5.

09/05/2023 - change 2

Added additional explanation to the argument i in Task 3.

09/05/2023 - change 3

Provided the additional assumption that n has to be a positive integer.

09/05/2023 - change 4

`flatten_image` function is now provided as part of scaffold (i.e., instead of being a part of task 1). The scaffold provided in the Overview and the grading rubric are also updated accordingly.

14/05/2023 - change 5

requirements.txt is removed from and image.txt is added to 'Submission Requirements'.