Journal Final Paper 2023

Monday 8. May

- I started a repository for the journal and the written work
- I brainstormed first and then picked three propositions which I formulated in more detail

Thursday 11. May

- Started journal
- Experimented with tensorflow
- Wrote basic image recognition code (initial commit on Vec2Ras)

Saturday 13. May

- Researched about Markup and Styling languages to find out what is available out there
- Responded on teams answer with more explanations about two propositions
- First experiments with Tensorflow.py

Sunday 14. May

- More experiments with Tensorflow > I found myself trying too much. After starting with the basic example model, I tried to directly move on to build the final image generator and training the model on its output > There I ran into a problem when I found out that the model doesn't accept the input. I tried to reshape it, change the model etc, and because I thought it *should* be possible, I lost much time with that. > I then stepped back, moved the whole code into a folder named stage2. > I began with the basics again (now stage0, copied them into stage1 and began to chop the problem into smaller pieces.

Monday 15. May

- work on stage1: Shape recognision with generated input data

Tuesday 16. May

- Made a model that works with data with comparable structure (stage1_1)
- Tried to train that model on images that vary in brightness instead in shapes stage1_2
    - Even after adapting the models and loss function, I couldn't make the accuracy rise above 0.5
    - The problem must be in the code or the loss function, because the loss is minimized quite well

On and before June 17th

I couldn't work that much on the project anymore, since I'm in a phase of many exams right now. Though I could do some more experimentation and research during the last weeks. This is where I'm at right now:

The problem with the brightness could not be easily solved, it may not be possible with the same model. I'm planning to take a closer look at multitask learning, and maybe use a model with shared layers but different branches to detect several features of the same object.

I've now been researching about existing approaches to tackle the problems I encountered. After searching on google scholar, I found several papers and works that could potentially be helpful: - Vectorization of Raster Manga by Deep Reinforcement Learning - Successful (very precise) raster to vector converter for mangas - A subtask of what I try to accomplish - interesting pipeline and *model* - Can't solve the shape-color problem though - Im2Vec - A try to synthesizing vector graphics without vector supervision - *Could maybe also be useful for generating train/test vector images*

Summary June 17th - September 11th

During the summer break I had dedicated one week for the matura work. I used this time, as well as the weekends that followed, to experiment more with the different tooling that is available for the task I'm trying to accomplish. Below I made a short overview of what I tried during the time since the last entry in the journal:

1. Tried alternative programming languages

I wasn't directly content with the language I had chosen. I decided to go with Python, since it is a common option for machine learning, especially for research in deep learning. It had all the necessary features that I was looking for: - A rich ecosystem for deep learing and working with neural networks with keras, tensorflow and pytorch - Easy to use libraries for working with raster and pixel graphics (opencv2) - Including a svg to png converter (cairosvg) - Additionally I expected to make quick progress, since I was already comfortable with the language and libraries and Python focuses on being simple and increasing developement speed

But after experimenting with tensorflow for a bit, I soon realized that Python might not have been an optimal choice. Especially the last aspect wasn't always an advantage. Python tries to increase developement speed of software through taking responsibility from the programmer, for example in that it automatically figures the type of a variable at runtime (dynamically typed. This did accelerate the developement of the image generation part, but as I went on and tried to train the model on the data I had generated, it hindered more than it helped. I essentially had to guess the types and dimensions that the functions accept, and didn't know when something was wrong until I ran the program and got a runtime exception. Expecially when intensively working with numpy arrays, it was an awful experience, which often led to many hours(!) debugging one otherwise simple dimensional dismatch error. Often the error would occur hiddeng deep in the tensorflow source code and be raised by Python, when an inpossible opperation was tried.

Since this whole kind of errors would not occur in any statically typed language, I tried different options: - Cython: - A superset of Python that lies between CPython (the normal Python implementation) and C - C and C++ - Tensorflow and pytorch are written in those languages, and they provide an API in at least one of these languages - Rust - Can use any Python library with a C-API (tch-rs, tensorflow) - Or any C library - As well as the brand new libraries that are written in Rust (burn) - Would be closer to Python syntax wise and more secure compared to C and C++

I quickly figured that with Cython I'd still have to use either the CPython libraries, which wouldn't solve my type problem, or C or C++ libraries or APIs, which would be interesting, since it would allow to freely switch between Python and C or C++. So I started to learn how to use them, but when I wanted to start using tensorflow, I found out that neither the tensorflow nor the pytorch (libtorch) APIs were well documented.

2. Tried alternative libraries

Namely pytorch, which I might be using in the final work, the C API of pytorch (libtorch) and the C++ API of tensorflow.

4. Writing the first module

After settling again on Python, this time using pytorch, I started from new with the most fundamental part of my work, the image data generation. I made it as flexible as possible, but focused on creating only fundamental shapes like lines, triangles and rectangles.

This module is finished for now and hosted on this colab notebook.

5. Writing a clearly defined concept This is the part I'm currently at. In a meeting with Beat Temperli, he poposed this idea. So before I continue much longer writing code I won't need, I intend to clearly make up my mind about what exactly I want to do, the scope of my work and a schedule. I also want to evaluate which language and deep learning library I'm going to use.

September 28th

Finished the concept.

Circle detection workes to a point where it is acceptable (see find_circle_center.png Pytorch seems like a good option, it is more intuitive than tensorflow.

October 5th

The circle detection is now done (center + radius) for now. I will extend it later to support color as well. I really began to have dynamic typing during this project. There have been so many nasty bugs and incomprehensible errors that would not even be a thing if the language just knew what types the variables are it's using. I have made screenshots of some of the worst error messages.

October 7th

I could extend the model quite easily to support the necessary shapes, there haven't been many difficulties. The restructuring certainly helped, and as soon as the model is accepting the matricies I'd like to feed it with, it becomes much easier.

October 9th

I'm again restructuring and rewriting the code. This time I don't focus on making it work though, but rather prepare for the submission. I try to keep a modular structure, separating training and testing/evaluation, but not I work with modules and functions rather than classes for the most part. It seems to fit for now, and as long as I don't see a need for OOP, I'll keep it that way.

October 28th

I've finished the training, as well as testing modules by now. A certain optimization, that already had caused many problems in another place has been removed completely now. I had written an explanation for it in the paper, but since it is not really relevant for the work itself, I think it's better in here:

Memory Management Pitfal

Interesting is that the memory that holds the numpy array which represents the pixel values could be reused, which could result in garbage collection being less often invoked. Since garbage collection often has to freeze the execution of the program, it can become a performance bottleneck if it is often invoked. Decreasing the memory usage of a program thus can lead to significant performance improvements in some situations.

Such a reusing of memory was tried to be achieved by using following code: `self.image[:] = np.full(shape, ..)`. Manual memory management is usually not possible in Python, since it does not officially support it. Objects like arrays are usually passed by pointer, and by reassigning to a variable, this pointer is overwritten, leaving the old value in memory until the next garbage collection is invoked. Then the garbage collector frees the memory after finding out that no references to it exist anymore. In the case of numpy arrays though, `[:]` can be used to explicitly dereference the numpy array.

So `a[:] = x` in Python, where `a` is a numpy array is the equivalent to do `*b = x;` in C in terms of memory access.

Interesting is, that after implementing this improvement, no significant performance improvement was being observed. Consequently, a micro benchmark has been written to be able to examine the effect of the change. The results are rather surprising: In a situation comparable to the one in the usage in the demonstration, overwriting the memory does introduce more overhead and is slower in that specific case than simply produce a new array. This behaviour might have to do with optimizing mechanisms of numpy and the possibility that the new array still is created by the `np.full` function, which subsequently had to be copied into the old memory spot.

Benchmarks and a demonstration have been conducted and stored under https://github.com/lrshsl/matura-doc/tree/main/preparation/benchmarks

From now on, I'll just have to write the paper and occasionally make small changes to the code.