OCR using generic algorithms

Jakub Kučera
ČVUT–FIT
kucerj56@fit.cvut.cz

May 9, 2021

1 Introduction

This report is about a semester project that I worked on for courses BI-ZUM (Introduction to artificial intelligence) and BI-PYT (Programming in Python).

The objective of this project was to create a program which could conclusively differentiate between symbols in a given dataset, by finding the smallest possible pixel combinations. A part of this objective was to use traditional AI methods, therefore I decided to use genetic algorithms (from here onwards referenced as GA), instead of convolutional neural networks, which are mostly used for OCR these days.

2 Input data

Given dataset is provided as a folder containing set of images each representing a single symbol. These images can be in various rasterized formats and resolutions, but must be the same in a single dataset.

These images are processed by first converting into a 2D array of 8-bit numerical values and later into an array of Boolean values based on a given threshold value.

To later reduce the time complexity when using GA, we can extract the indexes only of those pixels, which don't have the exact same value in all of the symbols. For example by applying this method to a provided dataset of standard English alphabet, we can reduce the number of relevant pixels from 256 to 214, which might not look as much, but the number of combinations of these pixels reduces dramatically.

3 Used methods

First important thing to decide when using GA is how to represent chromosomes. In tasks based around TSP (travelling salesman problem) chromosomes include all the values and differ in the order of elements, basically the goal is to find the correct permutation. Such approach couldn't be used in this case, since we need to find the correct combination of pixels, meaning that the order of elements is not important and not all values are needed. Therefore I decided to represent chromosomes as an array of pixel indexes. As for why I decided to use array and not a set was so that I could use NumPy library for C-like performance while using still Python.

Since I used a different kind of chromosome representation, I also had to come up with a different way to calculate fitness. After a few iterations I ended with a variant, where it's calculated by multiplying the number of occurrences of each pixel value combination in all symbols on given pixels. Additionally it has to be subtracted by 1 and multiplied by -1. This ensures that higher value means better fitness and that the best possible fitness equals to 0.

For the selection operation I decide to use the tournament selection, where a small number of random individuals from a population is chosen and the one with the highest fitness is chosen into the next generation.

Commonly used crossover methods also rely on only permuting elements so my version of crossover method takes two neighbouring chromosomes and randomly picks elements from them.

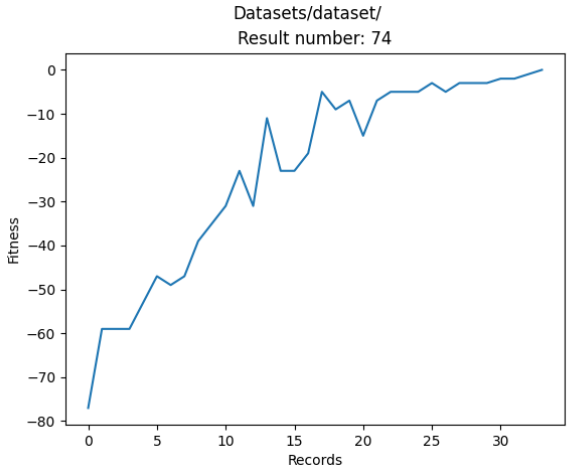

Figure 1: Plot showing best fitness in each generation

4 Results and observations

Throughout the development when I was testing this program, I realised that finding the the smallest

possible pixel combination to differentiate between all the symbols is going to be more problematic than I originally thought. In the English alphabet dataset, you need at least 5 pixels to find the correct solution. However since we want to find the smallest possible amount of pixels, we can only work with 5 pixels. There are more than 3 billion possible combinations for 5 pixels out of 214, however only very few are valid. I iterated through multiple versions of genetic algorithm improvements, but even when using the best ones, finding the solution for 5 pixels takes longer than expected. However I found out that for 6 pixel the solution is found almost instantaneously, which is why I decided so that the program with default configuration will try to find the solution for 5 pixels 2 times and then increase the number of pixels until the solution is found. This is of course all configurable by user.

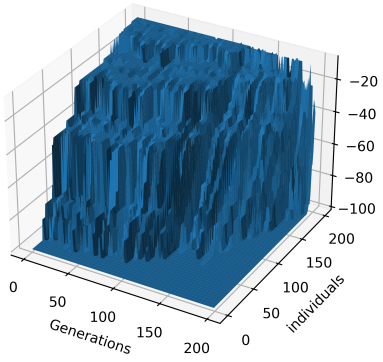

Figure 2: 3D plot showing fitness values across all generations and individuals

5 Conclusion

From observed behavior I came into the conclusion that using GA for OCR purposes is not ideal, I think that using more complex methods like already mentioned convolutional neural network will have more accurate and faster results.

References

[1] Jason Brownlee. Simple Genetic Algorithm From Scratch in Python. online, 2021. [cit. 2021–09–05] `https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/`.

[2] Ph.D. Doc. RNDr. Pavel Surynek and Ph.D. Mgr. Ing. Ladislava Smítková Janků. BI-ZUM course website. online, 2021. [cit. 2021–09–05] `https://courses.fit.cvut.cz/BI-ZUM`.

[3] Vojtěch Vančura. BI-PYT course website. online, 2021. [cit. 2021–09–05] `https://courses.fit.cvut.cz/BI-PYT`.