Maturaarbeit

Lars Hoesli

December 2023

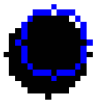
Circle: [25, 13, 9, 4, 17, 18]
Triangle: [18, 12, 17, 9, 12, 9]

Circle: [10, 13, 2, 11, 0, 0]
Rectangle: [6, 12, 9, 11, 1, 3]

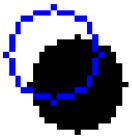
Circle: [7, 23, 17, 17, 21, 9]
Triangle: [14, 17, 17, 15, 12, 17]

Circle: [11, 19, 1, 11, 1, 0]
Line: [14, 12, 11, 4, 3, 0]

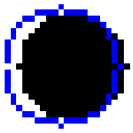
Circle: [8, 17, 7, 0, 0, 0]
Circle: [9, 15, 6, 0, -1, 0]



Circle: [19, 17, 8, 0, 0, 0]
Circle: [15, 12, 8, 0, -1, -2]



Circle: [11, 19, 9, 0, 0, 0]
Circle: [9, 19, 10, 1, -1, 2]

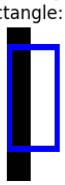


Circle: [8, 8, 12, 25, 1, 0]
Line: [9, 8, 18, 17, 0, 0]

Circle: [9, 10, 3, 16, 0, 0]
Rectangle: [6, 11, 8, 9, 0, 0]

Circle: [5, 31, 22, 19, 1, 0]
Line: [19, 17, 27, 20, 2, 0]

Circle: [4, 0, 1, 25, 0, 0]
Rectangle: [1, 3, 8, 17, -1, 3]



Circle: [12, 10, 5, 18, 0, 0]
Rectangle: [9, 7, 10, 14, 0, 0]

Circle: [17, 8, 2, 0, 0, 0]
Circle: [15, 9, 0, -2, -3, -1]

Circle: [13, 28, 6, 3, 3, 6]
Triangle: [11, 16, 9, 17, 7, 16]

Circle: [11, 17, 2, 0, 0, 0]
Circle: [11, 16, 5, -1, 0, 0]

Circle: [24, 20, 24, 28, 12, 11]
Triangle: [22, 24, 24, 18, 14, 17]

Abstract

This paper examines a particular approach for converting raster images with basic shapes into a vector representation. It is demonstrated how a neural network can learn to extract the necessary data through training with on-the-fly generated data.

Chapter 1

Introduction

Images have become an important part of everyday live, most of which are stored digitally, which is making the search for effective storage of images an essential, well researched aspect of computer science. Many different formats and compression have emerged, the most influential of which can be categorized into two main categories - vector and raster formats.

1.1 Vector and raster graphics

Vector and raster graphics are two fundamentally different approaches on how to represent the content of an image. Both have advantages in representing a certain kind of image, and are less appropriate in other situations. While raster images store the color values of small parts of a given picture - often called pixels - to approximate what the image looks like, vector formats rather store a specification for what shapes can be seen, similar to how humans describe images.

Both methods have pros and cons, and are more appropriate for certain situations than others. But vector formats have many advantages for storing images that are easily describable in shapes, especially shapes made up of one-colored areas or easily describable gradients. In such cases, vector graphics can represent those shapes more precise, with infinite resolution, while being less storage intensive at the same time. Raster formats in turn are better suited for images without easily distinguishable shapes, such as portraits or landcape images.

While format conversions among raster or vector formats and from vector to raster graphics can be done with a multitude of programs, the conversion from a raster image to a vector representation proves more challenging, especially because shapes have to be recognized.

Therefore a way to convert raster images into a vector format can be beneficial, and is not yet a solved problem.

1.2 Existing approaches

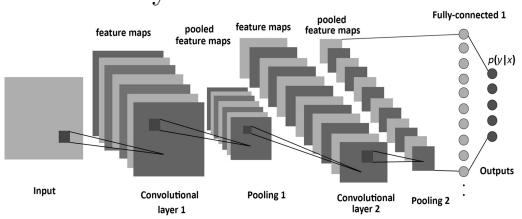
Raster to vector conversion is a well researched field, and there are many different approaches on how to do it. Apart from algorithm based strategies, machine learning and particularly neural networks are promising and in very active

1.3 Architecture of neural networks

Deep neural networks can be structured in different ways, leading to different kinds of traits that are beneficial in certain situations over others. For raster to vector conversion following architectures are interesting.

1.3.1 Convolutional neural network

A convolutional neural network (CNN) is a type of neural network that uses a convolutional layer to extract features from an input image. It is useful to reduce the information of the single pixels to a sequence of single features, that fully connected layers then can learn to connect. In order to extract the relevant features, it uses different *kernels*, which are moved through the images using a certain *stride value*, and applied to the pixel values. The *kernel* and *stride* values can be used to reduce the information for the following layers directly. Alternatively after each convolution, a certain function can be applied, which reduces the processed features. A common architecture consists of repeating convolution and pooling layers, and finally fully connected neuronal layers.



CNNs have the advantage of being able to automatically identify relevant features in images. As such, they are used for computer vision, object detection as well as classification tasks.

1.3.2 Recurrent neural network

1.3.3 Reinforcement learning

Chapter 2

Method

2.1 Overview

The data

2.2 Framework choice

2.3 Language choice

2.4 Model architecture

The ability to automatically learn how to classify but also extract important features from images is what makes the CNN architecture suitable for this project.

2.5 Data generation

The data that the model uses for training consists of raster images, with one shape in each of them. The images are internally represented as numpy arrays, where each entry represents the color values of a pixel. The background is white (i. e. RGB values set to (1, 1, 1)), and the pixels that fall within the shape are set to (0, 0, 0), thus appearing black, so that the contrast between shape and background is maximized.

All data generation is implemented in rtov/data/ and its subdirectories. A class, LazyDataset, which inherits from the torch. utils .data.Dataset, provides an interface, which a torch. utils .data.Dataloader object can use later to get the next image. Therefore, the method __getitem__(self , i: int) is

provided, which loads a numpy array, draws a random shape on it and transforms it into a vector.

Interesting is that the memory that holds the numpy array which represents the pixel values is being reused, such that garbage collection is less often invoked. Since garbage collection can be a performance bottleneck , this can in some situation lead to significant performce improvements. This is achieved by using `self.image[:] = np.full(shape, ..)`. This would usually not be possible in Python, since it does not officially support manual memory management and objects like arrays are usually passed by pointer, and by reassigning to a variable, this pointer is overwritten instead, leaving the old value in memory, until the next garbage collection is invoked and frees the memory after finding out that no references to it exist anymore.

In the case of numpy arrays though, `[:]` can though be used to dereference the numpy array.

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a[:])
```

2.6 Model architecture

[TODO]

2.7 Optimizing mechanisms

[TODO]

Chapter 3

Results

Chapter 4

Discussion

4.1 Opportunities of generated training data for vectorization

4.2 Limits

4.3 Proposed model architectures

4.4 Future work

4.5 Conclusion

Chapter 5

Sources and references