CMIT SUMMER PROJECT 2020 (CHEN/ALPERS): TOMOGRAPHIC RECONSTRUCTION OF POLYGONS USING CONVOLUTIONAL NEURAL NETWORKS

SUPERVISOR: DR. A. ALPERS

ABSTRACT. I would like you to do something very similar as in [2] (i.e., implement and play around with a convolutional neural network), but apply this to a data set that I have from a rather different application [1]. This is in some sense real research since nobody has ever done this before in this context, and I actually don't know what will come out of it.

1. Scientific Background

The motivation for looking at this problem originates from my paper [1]. There we show that it is possible to reconstruct nanowires from electron microscopy images acquired from a small number of angles (which cuts down the data acquisition time quite dramatically). We propose there several algorithms, and what we consider in this project is something similar to the algorithm UFBP (and in some sense $2n$-GON) in [1]. The deep learning approach would be a new approach to this, and it would be interesting to see how well it performs. At the same time it seems a good testbed to get some hands-on-experience with convolutional neural networks.

2. Problem Statement

We consider 2-dimensional binary images (say, the object consists of the pixels with value 1 denoting the color white, while 0 denotes the background color black). Considering 2D is no real restriction, because 3D data sets are usually viewed slice-by-slice in tomographic applications.

Actually, what we know is that the binary object is approximately a (not necessarily regular) hexagon. But we don't know how the object really looks like; we want to reconstruct it from *tomographic data*. The tomographic data is given as a *sinogram*. Instead of defining it, I show you an example of how to compute it.

2.1. **Sinogram.** Consider the 5×5 pixel image shown in Fig. 1(a). (Please ignore that the object is a box and no hexagon.) Now, we count vertically the number of white pixels along each vertical line. The values are shown in the top row of Fig. 1(b). One can also view these as gray values, the corresponding colors are shown in the bottom row of Fig. 1(b). So, these values (or if you want, these gray values) are the tomographic information that we obtain from X-ray measurements taken from the object rotated by 0°. Now, rotate the object (you can obtain this by calculating the pixel coordinates of the object multiplied with the corresponding rotation matrix and by inserting them - rounded - into a new blank image). For this new angle, we get again 5 numerical values (or gray values). Suppose, we do this now for all angles from 0° to 179° in steps of 1°. What we obtain is a 180×10 matrix (containing the numerical values or gray levels). This matrix is the sinogram. And from this matrix it can be possible to reconstruct the object.

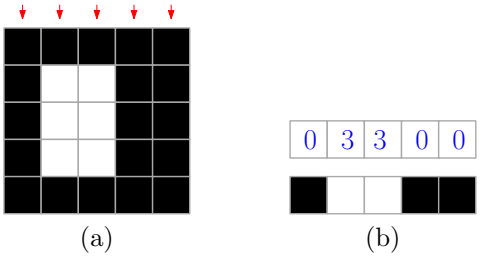

FIGURE 1

E-mail address: andreas.alpers@liverpool.ac.uk.
Date: July 24, 2020.

2.2. **Reconstruction.** We deal here with polygons (hexagons, but for testing it might be better to start by considering boxes). A very straightforward algorithm for reconstructing polygons from sinograms is the following (it is called UFBP in [1] and this is the one I would recommend to take, if that works and you are interested you might later explore other methods, but I guess there won't be time for it): Start with a black 5×5 image. Then consider the sinogram for $0°$. Set all pixels in the blank 5×5 image to 1 which are on a vertical line where the sinogram gave a non-zero value. Rotate the image by $-1°$, and add ones along the complete vertical strip were the $1°$ sinogram had non-zero values. Proceed in the same way for all other angles. Now, at the very end your polygon should consist in the pixels that have been in all strips, so your reconstruction should be the binary image consisting of the pixels that have value 180 (or, to be a bit noise-robust, have values, say, larger than 170).

If you play around with this (and it should be easily implementable in Python, but if you prefer you can do this also in other languages such as matlab), you will notice that the reconstruction is possible without taking all the 180 angles into account. It would suffice to take the angles such that your vertical lines are parallel to the edges of the convex object (so for hexagons 3 angles should be sufficient).

But how do you find out these angles? (One approach is the $2n$-GON algorithm in [1].) In fact, this is what we want to find out using the convolutional neural network.

3. Suggested Work Plan

I would suggest to do the following. First, we need to create a training data set. Instead of hexagons, I would suggest to first play around with boxes. Generate boxes of varying size and orientation in a random way and compute sinograms for this (later you can also perturb them by some random noise). The 'good' angles for these boxes (and later your hexagons) are known. They are the angles such that your vertical line is parallel to the edges. So for your boxes you will have them $90°$ apart, say degree α and $\alpha + 90$. These are the two labels for this sinogram (or, in other words, angles α and $\alpha + 90$ have a high value, the others have value 0). And you train now your CNN with many of such labeled sinograms.

After training you present some new sinogram to the CNN. The hope is that the CNN gives you as output the angles (in the box-case the 2 angles, in the hexagon case, the 3 angles) which are close to the good angles for reconstruction. As mentioned in [2], maybe the CNN is giving you more than this small number of angles. Then, you can experiment of what to do with this. In [2] they apply a k-means algorithm (this is easily implementable) selecting the centroids of the $k = 2$ (or $k = 3$ in the hexagon case) cluster of points representing the angles with the highest value (here one would also need to experiment where to set the threshold).

Then, reconstruct it with the above described reconstruction algorithm and compare with the ground truth (the original object for that sinogram). How well does the CNN perform if you do this a large number of times?

4. Comments

(1) So, in some sense it is very similar to what is described in [2]. However, there they take arbitrary shapes, consider more general reconstruction algorithms, and there they also need a way to find the best angles (we know them, because we consider polygons; so their algorithm SFS does not need to be implemented and considered in our context). We have a more special situation here, and it could be possible that we get better or similar results.

(2) What image size should one take? Here I also suggest to play around with what is computationally feasible for your computers. If only small sizes work, that would also be fine. We could also decrease the solution of the original data, so that all images are small (for instance 32×32 or 64×64).

(3) I think it would be good to play around with data sets that you generate and where you have complete control. For instance, start with boxes. If that works, go to hexagons. (And possibly noisy sinograms.) If that works, you can go to the real data set that I would then provide.

(4) Generating the sinograms and getting the correct reconstruction is probably the best first step in this project. If this works, go for the CNN. I would suggest to take the same or similar parameters as in [2]. If you like you can play around with other parameters and architectures. Implementing the CNN can also be done independently, I think. You can just make up some abitrary 'good' angles for a some arbitrary data set that you create. And then you can test whether your CNN recovers these good parameters. If that works, then you only need to concentrate on the first part (the sinograms and reconstructions).

(5) I think it would be best if you 3 work together. So, having one program and short report at the end. If you like you can all have your individual implementation. But it would be good if you communicate (say over skype) to help each other. I would be also available sometimes over skype or email. But as I said, try to do this as group work (communicating first among yourself).

(6) Of course, it would be an ideal outcome if everything works very nicely. However, I would also count it as an success if you play around with this type of problem, getting (and documenting getting) some experiences with CNNs.

References

1. A. Alpers, R. J. Gardner, S. König, R. S. Pennington, C. B. Boothroyd, L. Houben, R. E. Dunin-Borkowski, and K. J. Batenburg, *Geometric reconstruction methods for electron tomography*, Ultramicroscopy **128** (2013), no. C, 42–54.
2. G. Pap, G. Lékó, and T. Grósz, *A reconstruction-free projection selection procedure for binary tomography using convolutional neural networks*, Karray F., Campilho A., Yu A. (eds) ICIAR 2019: Image Analysis and Recognition, Springer, Cham, 2019, pp. 228–236.