Automated Cropping of Large Image Filesets with Python

Thomas O'Brien

School of Engineering and Innovation, The Open University, UK

Abstract

Autocropping of digital images offer an opportunity to both eliminate waste outside the region of interest and also to save on storage space for data. While work is ongoing on utilising machine learning techniques to autocrop images, the accuracy and repeatability of these techniques continues to pose challenges. This technical note describes a simple algorithm that can be used to autocrop large image filesets according to a pre-defined cropping criteria and is then is programmed using Python. The program is tested on two differing filesets and results are quantified in terms of storage space reduction. The results show that significant improvements in terms of the region of interest and in storage requirements can be attained with a simple approach.

Keywords: Digital image processing, autocropping, Python

1. Introduction

Digital imaging systems have transformed the method by which individuals and organisations document visual data. The vast majority of visual recording is now done using digital cameras and they are widely used throughout science and engineering to record data for further analysis. However, due to the ease and speed of data capture, increasingly large filesets are being produced and these filesets are driving a need for improved automation methods to enable the extraction of useful data for analysis. A frequent problem encountered with the recorded data is that the recorded image contains a particular region of interest (ROI) as a subset, therefore, a significant percentage of the recorded information is not required. A further problem that this gives rise to is the need for increased data storage to retain the acquired data. Various recent studies have found the annual growth rate of data is increasing by up to 50% per year (Buko et al. 2023) and much of the data being stored is wasteful offering little or no purpose.

Digital image libraries are produced using a wide range of experimental and empirical techniques across various scientific and engineering disciplines. Medical imaging, flow visualisation, structural analysis, photography and microscopy all utilise digital image acquisition to gather and store data. Different image digitalisation approaches, such as CCD or CMOS, may be used but the final recorded data is always a digitalised image of the subject matter.

The digitalisation of images enables computational methods for their analysis. Extensive software resources are now available that enable researchers to analyse both single and multiple image filesets. However, while software available for image analysis is improving, they are lacking certain tools and automation features in standard software. In addition to this, due to the wide range of applications that digital images can be used for, there is frequently a need to develop bespoke programs that are capable of performing specific tasks to deliver certain objectives.

The cropping of a digital image is one such method that has long been available and which allows the user to reduce the size of the image file by reducing the area of interest being recorded (Zhang et al. 2005). Single image cropping is widely implemented throughout commercial software and is a standard tool that all digital image processors would expect. However, auto-cropping of a digital image fileset where a series of images exist all of which contain undesired or excess data is not yet a standard feature within most commercial business software.

Researchers are investigating the implementation of machine learning algorithms to deliver auto-cropping in two significant areas of interest: photography and medical imaging (Samii et al. 2014, Patansis et al. 2023). Due to the mass growth of consumer digital photography, there is a clear need for tools to enable the general public to autocrop multiple photographs. However, though the ROI is generally subjective and is not easily specified nor quantified (Bhattacharya et al. 2010), processing necessity dictates that quantification is obligatory and so approaches need to be specified in order to enable autocropping. In medical imaging, the ROI described as the area of interest in a medical imaging scan is the area within a digital image that must be recorded, but, it is generally recorded along with undesired regions. Application of machine learning algorithms to image training datasets may be considered acceptable for general public photographs, however, the inherent inaccuracies within machine learning algorithms means that applying them to medical images for auto-cropping purposes may result in a loss of data that must be quantified and which could be deemed unacceptable (Zia et al. 2018).

Simple autocropping approaches are available although these approaches have not been standardized and results can vary (Liu, 2018). The need for accurate and powerful auto-cropping tools is clear but their development has not been standardized and their availability is limited.

The objective of this technical note is to demonstrate a simple yet effective method of autocropping a large image dataset with a view to reducing the size of the images by recording only the ROI. The approach may have application in managing large libraries where the storage cost of the dataset is to be minimised. It may also prove beneficial to filesets that require further processing where processing time is related to file size. The algorithm that underpins the approach is presented along with test results on two sample filesets. One fileset is derived from a product lifecycle management application while the other fileset is derived from a medical imaging application. Each dataset consists of 10 individual image files with each file stored within the same format and filesize. The programmed algorithm is then used to autocrop the dataset. The approach is demonstrated using an algorithm developed for implementation by Python 3.8.4.

2. Algorithm

The principle of the algorithm is to write each pixel (greyscale) value into a list or array that reads sequentially through the x, y co-ordinates of the input raster image. This assignment is repeated for all images in the fileset. The algorithm then checks each pixel value against a criteria which can be based on a threshold value, a rate of change in the pixel value, or, some other selection criteria. If the pixel is found to represent the change criteria, then, the pixel value is recorded as being a point to which the image may be cropped. The preceeding pixels on the previous line are erased and the image is cropped to that point. The image is then rotated 90 degrees and the cropping steps are repeated. The complete process is then repeated for the next image in the fileset.

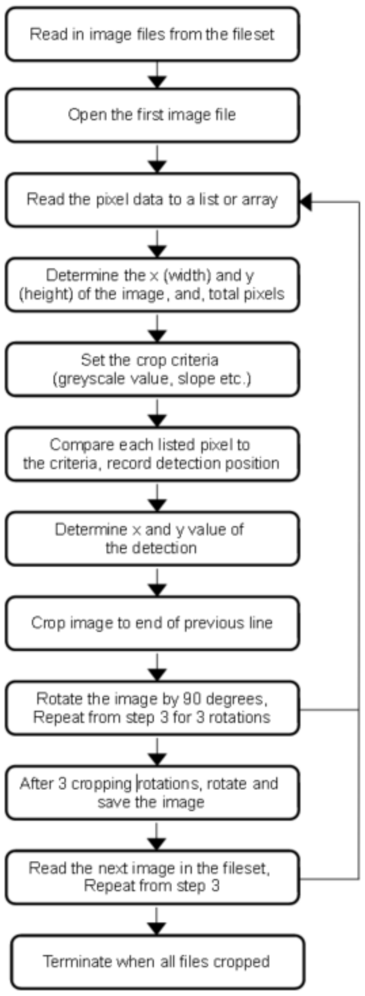
Read in image files from the fileset
Open the first image file
Read the pixel data to a list or array
Determine the x (width) and y (height) of the image, and, total pixels
Set the crop criteria (greyscale value, slope etc.)
Compare each listed pixel to the criteria, record detection position
Determine x and y value of the detection
Crop image to end of previous line
Rotate the image by 90 degrees, Repeat from step 3 for 3 rotations
After 3 cropping rotations, rotate and save the image
Read the next image in the fileset, Repeat from step 3
Terminate when all files cropped

Fig. 1. Computer algorithm used to autocrop the image filesets.

Libraries

A number of additional Python libraries are imported to enable programming of the algorithm. *PILLOW (Python Imaging Library)* is used to import the *Image* class which allows conversion and rotation of images. *NumPy* is a library used for numerical analysis and array processing while *Imageio* is the Python library that is used to read and write various types of image data. Lastly, the *OS* library can be used to read the image files from the directories to a list.

3. Method

The general method is illustrated by the following steps.

Step 1. Read in the image files:

file_list = os.listdir('C:\\python\\imagefiles')

Read in the output names of cropped files:

partlist=open("C:\\python\\partlist.txt")

Step 2. The following approach is used to read the pixel values (RGBA) into a list and to store the image dimensions. A loop is initialised and each image is processed for the total number of images:

im = Image.open(files[i], 'r')

Step 3. The pixel values are then read into a list:

pix_val = list(im.getdata())

Step 4. The variables width and height are assigned to the image dimensions:

width, height = im.size

The total number of pixels for the image is then determined. This number is then assigned as a range.

Step 5. The criteria for the crop condition is then set. As an example, a simple crop condition that will crop white from the image (RGB 255,255,255) is set.

Step 6. A *for* loop is then used to iterate through the *pix_val* list. For each entry in the list, the value is compared to the cropping criteria. As an example of a simple black/white comparison, the following condition can be checked. In this example, white pixel lines, with an RBG of 255,255,255 is to be cropped out of the image. If a non-white pixel is detected, the count of the first non-white pixel is noted.

```
value=(int(pix_val[x]))
if value != 255:
        nonwhite.append(count)
```

Step 7. Following detection of the first pixel to crop to, it is possible using the width and height information to determine the number of complete lines preceeding the line of the first instance of the condition. This then allows calculation of the number of lines to crop from the first pixel in the list.

Step 8. The image can then be cropped for the number of lines preceding the first instance of the condition.

croppedIm=im.crop((left, top, right, bottom)) # (x-start, y-start, x-end, y-end)

Step 9. The image may then be rotated 90 degrees and the cropping process repeated. Rotation of the image for three further crops result in the entire image being cropped.

im2=croppedIm.rotate(90, expand=True)

Step 10. Save the cropped image following a final 90 degree rotation to the original position. The image can then be written as a new file according to the file name list and type. In this example, the file is saved as a PNG although other file type choices are possible using the Python module.

imageio.imwrite(partlist[i]+'.png', imag[:, :]

Step 11. Read the next image in the dataset.

Step 12. Terminate the program when all files have been cropped.

4. Image Set Results

Two sample image sets were investigated using the programmed algorithm. The image sets are typical of large image sets used in technical and scientific work. The first image set represented general technical product specification information stored in the form of isometric drawings while the second image set represented a more demanding example of where detailed three dimensional geometric information is contained within a sequence of medical images (Cabibihan & Abinahed, 2018).

The objective in processing the first image set was to demonstrate the approach whereby pixels that do not contain useful information are cropped. While a sample of 10 images were selected for the purposes of this test, the program could also be used to crop image sets numbering into the thousands or tens of thousands of files. The images were stored as *.png files as 8 bit colour images with dimension 500 x 500 pixels and were reduced in pixel dimensions by the program.

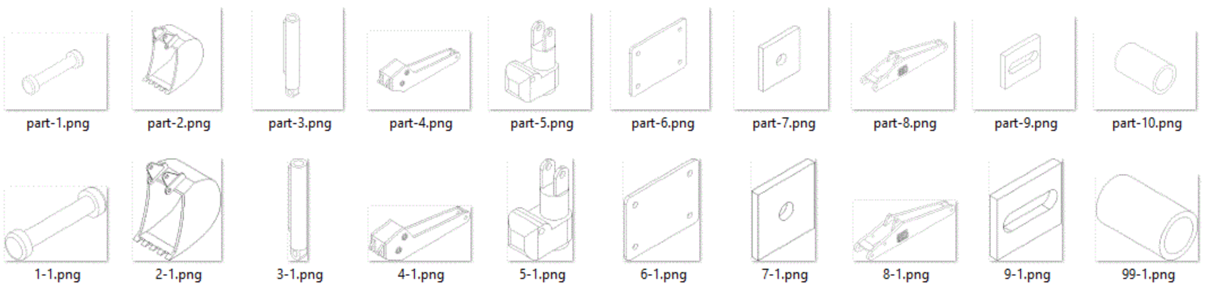

Fig 2. The first set of images cropped using the program.

A second image set was processed in order to demonstrate the program on a set of medical images. The images were stored as *.png files as 16 bit colour images a with dimension 512 x 512 pixels. The images were sourced from the Harvard Dataverse online repository of medical images (Cabibihan & Abinahed, 2018).

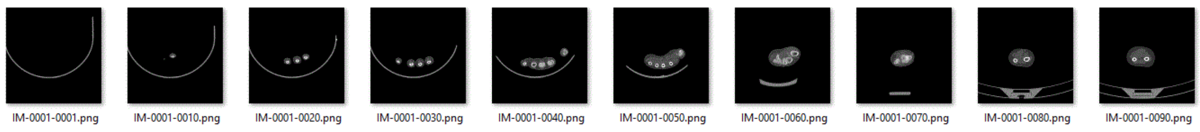

Fig. 3. The medical images as sourced from the online repository.

Images are batch inverted to show regions of interest with greater clarity and cropped using the program:

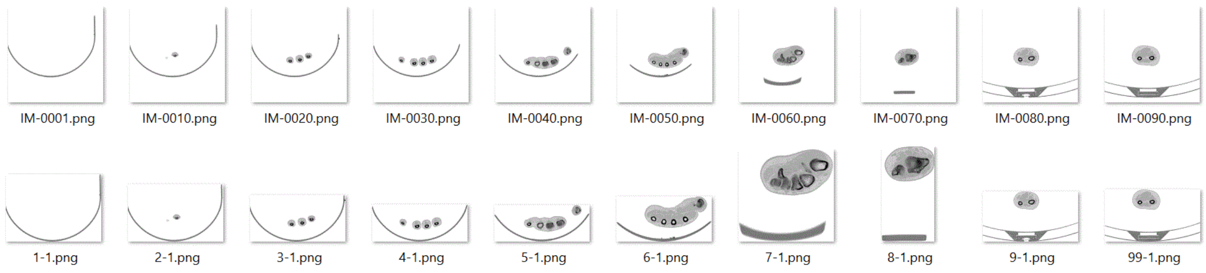

Fig. 4. The second set of images cropped using the program.

In both tests, a significant reduction of memory required to stored the cropped images was noted. The amount of storage memory saved is a function of the amount of waste data stored in each image file. In the medical images, storage space was reduced from 145kB down to 107kB, a 26% reduction. In the product part fileset, the storage space was reduced from 38kB to 31kB, an 18% reduction. It is therefore demonstrated that autocropping large image filesets can result in considerable savings in computer storage particularly in filesets containing significantly large areas outside the ROI.

5. Discussion

Manual cropping of image libraries remains an option where concerns surrounding accuracy and image resolution remain (Juneja et al. 2023). Deep learning has been used to autocrop images in medical applications, however, these approaches require large training, validation and testing libraries together with advanced algorithms and expertise (Juneja et al. 2023). Despite the computation effort required, cropping errors can occur and valuable data can be lost. Semi-automated approaches are also used (Zia et al. 2017) but also report challenges with accuracy and effort required.

Deep learning techniques, such as convolutional neural networks, have been used to automatically crop multiple medical images in order to detect regions of interest for certain medical conditions. (Liu et al. 2021, Patsanis et al. 2023). However, the training of deep learning models require large, annotated libraries. In addition to this, deep learning models are typically trained on specific image sizes and require retraining when image size (m x n) changes. Preprocessing of images, when done manually, requires cropping of the image to capture a certain ROI and this preprocessing stage is time consuming (Patsanis et al. 2023). In addition to this, automated cropping approaches, when applied, frequently rely on simply defined cropping approaches such as centre-cropping to a defined window for all images processed. This can result in a loss of data when the ROI falls outside the defined window.

A simple algorithm designed to reduce the size of image filesets has been demonstrated whereby significant savings in storage has been attained. The crop criteria selected was based on a simple pixel greyscale value but it is clear that other crop criteria can be programmed as required.

Conclusions

This paper presents a method that can be used to automatically crop large libraries containing multiple raster image files. An algorithm has been described and programmed in Python that enables a set of images to be cropped according to a predefined cropping function. The program has been used to automatically crop two different image libraries typical of visual libraries found in mechanical engineering and medical imaging. The program resulted in rapid automatic cropping of multiple images to record only regions of interest and to delete regions lacking relevant information. This method could be adapted to a wide variety of image interpretation problems and would help conserve data storage.

References

Bhattacharya, S., Sukthankar, R., & Shah, M. (2010). A framework for photo-quality assessment and enhancement based on visual aesthetics. Proceedings of the 18th ACM international conference on Multimedia. https://dl.acm.org/doi/10.1145/1873951.1873990. (Accessed 4 May 2024)

Buko, T., Tuczko, N., Ishikawa, T. (2023). DNA Data Storage. BioTech (Basel). 2023;12(2):44. Published 2023 Jun 1. doi:10.3390/biotech12020044. (Accessed 4 May 2024)

Cabibihan, J.J., Abinahed, J. (2018). CT Data for Upper Limb Prostheses and Lifelike Robotic Arms. https://doi.org/10.7910/DVN/TJTSCM. (Accessed 4 May 2024)

Juneja, M., Singh, G., Chanana, C., Verma, R., Thakur, N. & Jindal, P. (2023): Region-based Convolutional Neural Network (RCNN) architecture for auto-cropping of pancreatic computed tomography. The Imaging Science Journal. https://doi.org/10.1080/13682199.2023.2226413. (Accessed 4 May 2024)

Liu, M., 2018. Stackoverflow. https://stackoverflow.com/questions/51158106/how-do-i-crop-multiple-images-using-pil?rq=3. (Accessed 4 May 2024).

Liu, K., Zhu, S., & Zhang, J. (2021). A Lightweight Auto-Crop Based on Deep Reinforcement Learning. International Symposium on Advanced Technologies and Applications in the Internet of Things. https://ceur-ws.org/Vol-3131/paper2.pdf. (Accessed 4 May 2024)

Patsanis, A., Sunoqrot, M.R.S., Bathen, T.F., Elschot, M. (2023). CROPro: a tool for automated cropping of prostate magnetic resonance images. Journal of Medical Imaging . Vol. 10(2). https://doi.org/10.1117/1.jmi.10.2.024004. (Accessed 4 May 2024)

Samii, A., Mech, R., & Z. Lin, Z., (2014). Data-Driven Automatic Cropping Using Semantic Composition Search. Computer Graphics Forum. Volume 34 (2015), number 1 pp. 141–151. https://doi.org/10.1111/cgf.12465. (Accessed 4 May 2024)

Zhang, M., Zhang, L., Sun, Y., Feng, L. & Ma, W., (2005). Auto cropping for digital photographs. 2005 IEEE International Conference on Multimedia and Expo, Amsterdam, 2005, pp. 4 pp. doi: 10.1109/ICME.2005.1521454. (Accessed 4 May 2024)

Zia, R., Akhtar, P., & Aziz, A. (2018). A new rectangular window based image cropping method for generalization of brain neoplasm classification systems. International Journal of Imaging Systems and Technology, 28, 153 - 162. https://onlinelibrary.wiley.com/doi/10.1002/ima.22266. (Accessed 4 May 2024)