Image Processing Report

Michal Pluta
April 30 2024
733 words

1 - Introduction

This report aims to outline the thought process behind developing an autonomous knowledge extraction pipeline (Figure 1) for X-ray imagery (Figure 2).

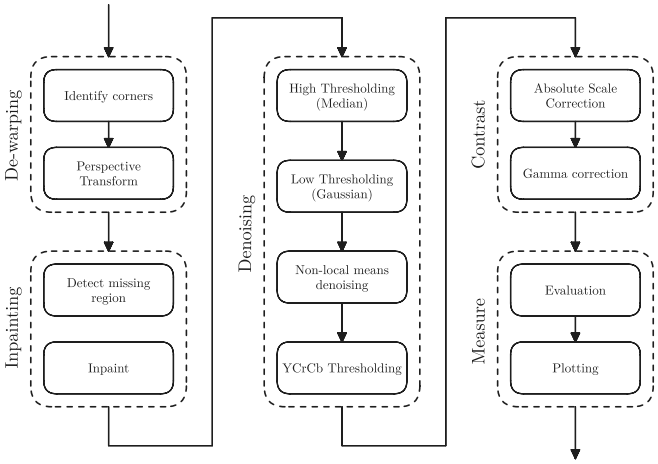


Figure 1: Implemented Image Processing Pipeline

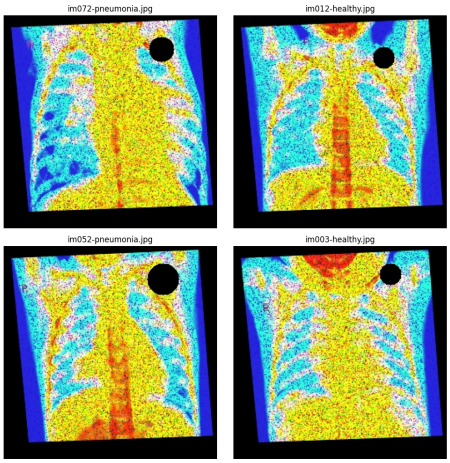


Figure 2: Sample noisy images

As evident from Figure 2, the input images exhibit a variety of different noise, imbalance and adjustments i.e.

Type of issue	Description
Warp	The images have been perspective warped and downscaled within the boundaries of the original image.
Missing Region	There is a randomly sized and positioned circle cut out of each image.
Impulse Noise	In the form of salt and pepper noise varying in the range [0, 255], scattered around the image.
Gaussian Noise	Present in the form of red and green mis-aligned patches.
Brightness	The images are too bright and lack contrast.
R Overlay	There is an R pattern overlayed on the images. However it is hard to tell whether this is part of the image or noise.
Color Imbalance	Currently too much yellow but it may be infeasible to restore the original shade of green/teal.

Table 1: Summary of issues with the input images

1.1 - Performance Metrics

Accuracy alone can be misleading as an increase in accuracy does not necessarily mean the model is more confident in its predictions. As a result, I chose to iterate my algorithm using accuracy, RMSE and a plot of the model's outputs as metrics.

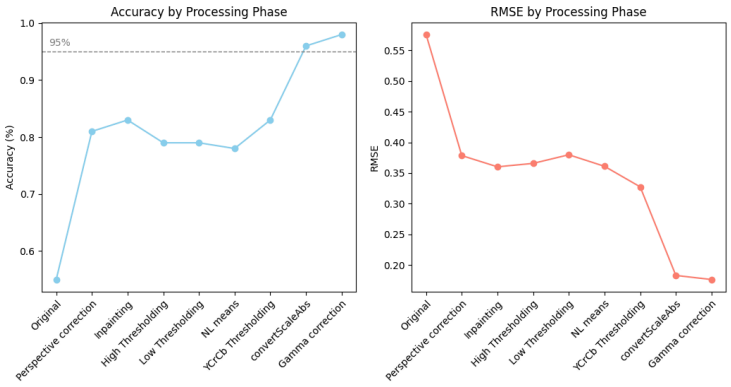


Figure 3: Graphs of Accuracy and RMSE at different stages of the image processing pipeline

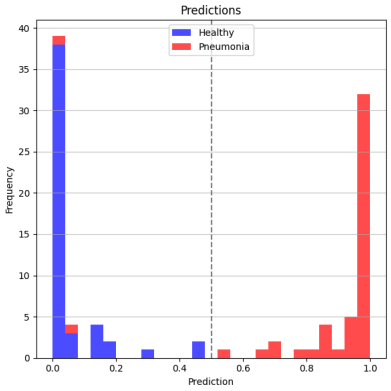


Figure 4: Stacked distribution of model predictions

While experimenting, it became very clear that small changes to the images could result in drastically different outputs from the model. This suggested that the model was very overfitted, which prompted me to divert my attention from model accuracy to image quality instead.

I considered using metrics such as PSNR, SSIM and Entropy to establish an empirical measure of noise removal, however, without access to the original images this was infeasible and hence, I judged image quality entirely subjectively.

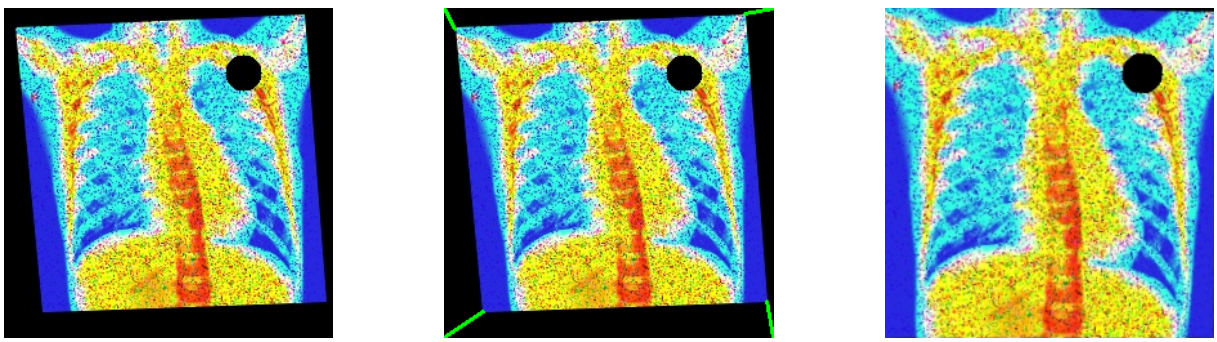
2 - Methodology

2.1 - De-warping

The most prominent issue with the images is the perspective warp. I considered two methods of finding corners, needed to reverse the warp.

- Slowly inscribing the image in a box until corners are reached
- Using contour detection and then aproximating a polygon

Overall, both methods worked on all images, however contour detection often found better corners since it is less sensitive to noise in the image.



Original | Warp Stretch illustration | De-warped

Figure 5: Images showing the de-warping process

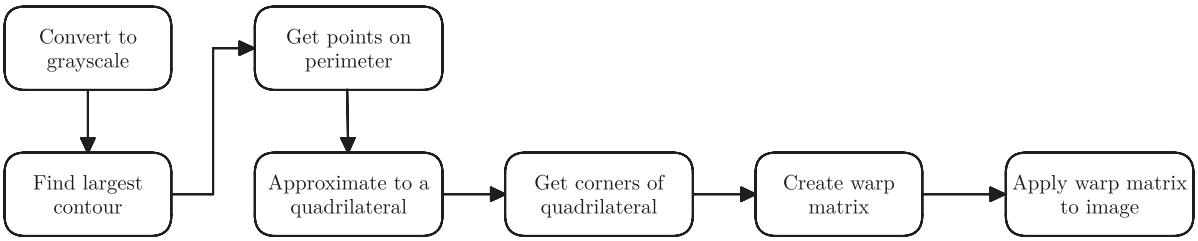


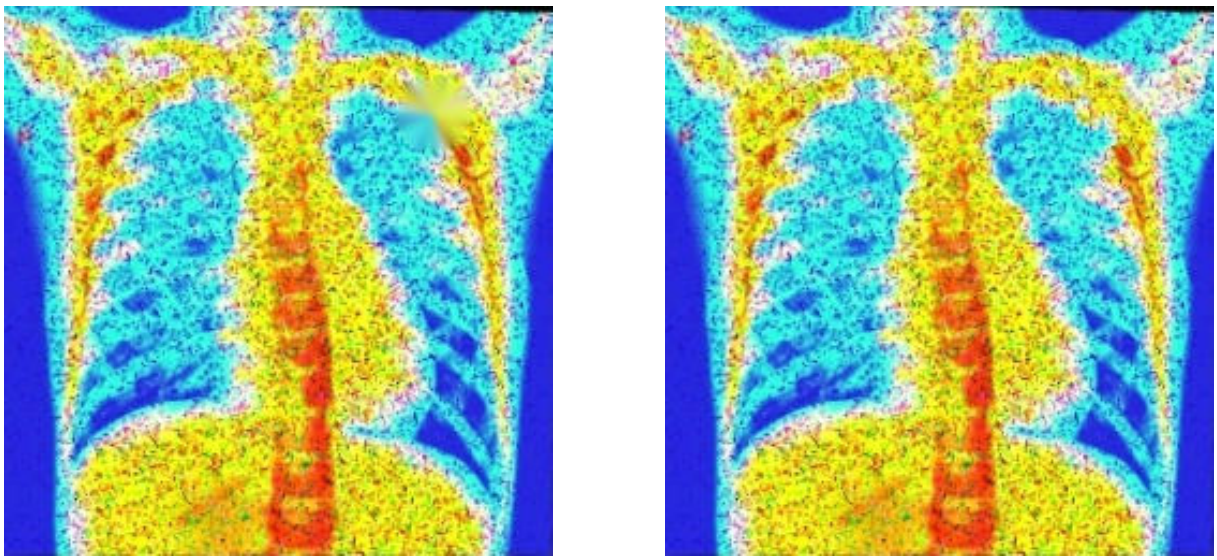
Figure 6: Pipeline for de-warping an image

As can be seen in Figure 5 and Figure 6, Once the source corners are found, a transformation matrix is calculated and applied to the image.

2.2 - Inpainting

Another big issue with the images is the missing region. By converting the image to greyscale and thresholding the dark areas (<40) I obtain the mask for the circle.

Looking at the inpainting results of the basic `inpaint()` method in Figure 7, it is clear that it doesn't look good enough.



OpenCV `inpaint()` Exemplar-based inpainting

Figure 7: Comparison of inpainting algorithms

To improve upon this, I adapted an existing implementation [1] of a well-known Exemplar-based inpainting algorithm developed by Criminisi et al [2].

Exemplar-based inpainting techniques work by filling in missing or damaged areas of an image with plausible data from surrounding 'patches'. They leverage existing structures and textures of images to create coherent, inpainted images.

This algorithm in particular starts by computing the fillfront, the pixels on the boundary of the region to be inpainted. The algorithm then determines which point along this boundary to address first, prioritizing those with the highest confidence and best structural and textural match to their surroundings. Once a target point is selected, the algorithm iterates through all possible source patches and chooses the one which minimises the sum squared difference (SSD) and the patch variance. This patch is then copied onto the target point and the process is repeated until the image is repaired.

2.3 - Denoising

The next big issue is noise, and in particular pepper and coloured gaussian noise. From my experimentation, and due to the fact that pneuomia is characterised by white pixels, I chose not to handle any of the salt noise, as I felt it may unintentionally harm the model's performance.

A common way of handling pepper noise is through the use of local filters such as Median or Gaussian blurs, or even Non-local filters such as a Bilateral Filter. The biggest downside to this is loss of sharpness and loss of detail in images. I tried to avoid this entirely by using filters such as Improved Adaptive Weighted Mean Filter [3], [4], but ultimately the filter wasnt very effective.

My current solution to this is by utilising thresholding to selectively pick out the noise from the image.

2.3.1 - Median Thresholding

- Identify strong pepper noise
- Replace pepper noise with median image

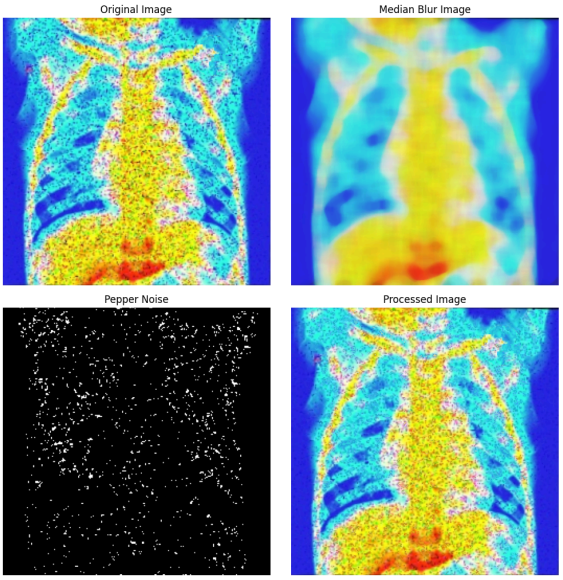


Figure 8: Phase 1 of noise removal

2.3.2 - Gaussian Thresholding

- Identify weak pepper noise
- Replace pepper noise with Gaussian image

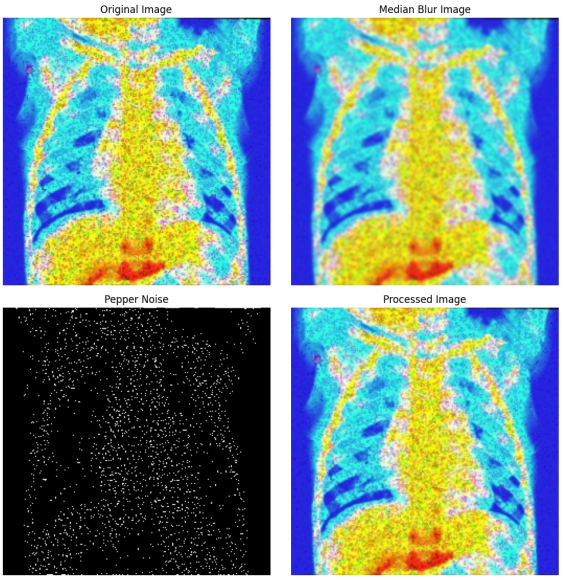


Figure 9: Phase 2 of noise removal

2.3.3 - Non-local Means Denoising

- Apply a Non-local means Denoising filter to all channels with varying strength
- Red is denoised most

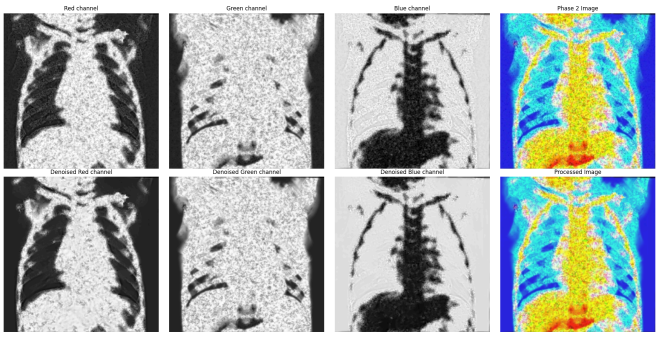


Figure 10: Phase 3 of noise removal

2.3.4 - YCrCb Thresholding

- Identify strong pepper noise within the Y channel in the YCrCb color space
- Create mask from the noise
- Mask areas in the image are replaced with mask areas from the Median image of the Y channel

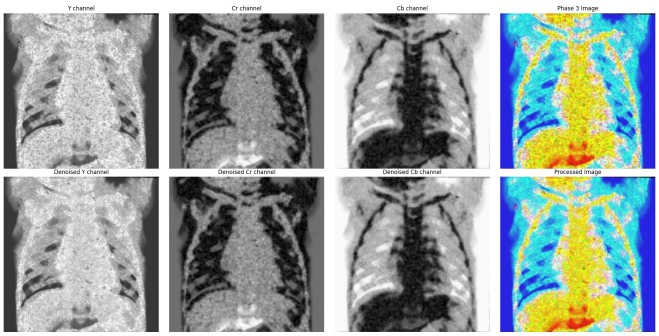


Figure 11: Phase 4 of noise removal

2.4 - Performance (Accuracy & Efficiency)

Importantly, the base inpainting algorithm [1] is very unoptimised:

Inpainting Method	Average runtime (s)
OpenCV Inpaint	1.24×10^{-3}
Original Exemplar-based inpainting	214.73
Vectorised Exemplar-based inpainting	48.60
Reduced Exemplar-based inpainting	9.55

Table 2: Summary table comparing run-times of various inpainting implementations

By utilising `numpy`'s vectorised operations, the efficiency of inpainting can be greatly improved, which is especially important when you consider that inpainting accounts for the majority of the total processing time (See Figure 12).

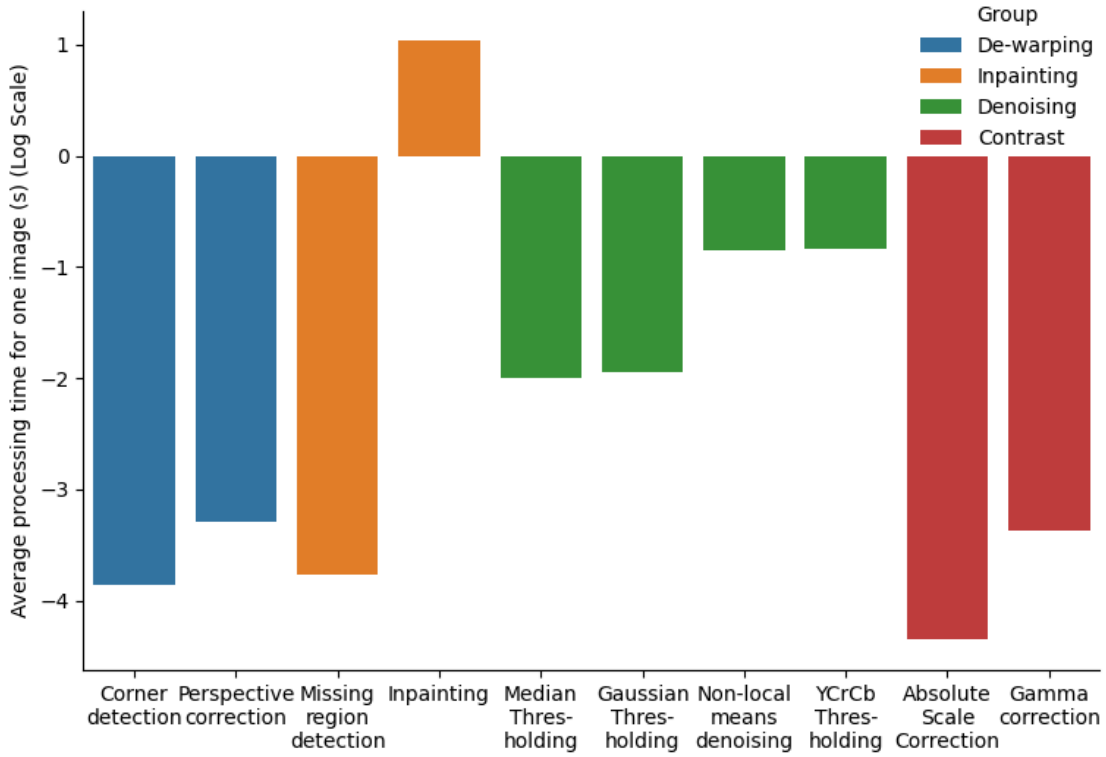


Figure 12: Comparison of average runtime of each processing phase

In terms of accuracy, the model performs exceptionally well, however, it returns 2 false negatives Table 3. Looking back at Figure 4, it is clear that the model is wrongly over-confident on these images, which suggests the image processing was either too strong or the images were initially faulty.

		Actual	
		Pneumonia	Healthy
Predicted	*Pneumonia*	48	0
	Healthy	2	50

Table 3: Confusion matrix of image processing algorithm, with Type 1 error shown in orange and Type 2 error shown in red.

Bibliography

[1] NazminJuli, "Nazminjuli/criminisi-inpainting: Python reimplementation of well known exemplar-based image inpainting." [Online]. Available: https://github.com/NazminJuli/Criminisi-Inpainting

[2] A. Criminisi, P. Perez, and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," *IEEE Transactions on Image Processing*, vol. 13, no. 9, pp. 1200–1212, 2004, doi: 10.1109/TIP.2004.833105.

[3] U. Erkan, D. N. H. Thanh, S. Enginoğlu, and S. Memiş, "Improved Adaptive Weighted Mean Filter for Salt-and-Pepper Noise Removal," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 2020, pp. 1–5. doi: 10.1109/ICECCE49384.2020.9179351.

[4] N. N. Hien, D. N. H. Thanh, U. Erkan, and J. M. R. S. Tavares, "Image Noise Removal Method Based on Thresholding and Regularization Techniques," *IEEE Access*, vol. 10, no. , pp. 71584–71597, 2022, doi: 10.1109/ACCESS.2022.3188315.