Enhancing Chest X-ray Images for Improved Classification

tdkq34 —— Adam Wood

Introduction

In medical image analysis, the quality of the input images plays an important role in the accuracy of diagnostic algorithms, thus creating a need for pre-processing to enhance image quality. So, to achieve the best classifier performance, I used a wide range of techniques. After all of the steps that I will outline below, the classifier achieved an accuracy of 95%.

Method & Implementation

I started off by trying to tackle de-noising. I did so by using the fast NLM de-noising function in cv2, I tried adding Gaussian Blur as well, but it only seemed to decrease accuracy for little to no visual gain. Without any de-noising techniques, accuracy drops to 91%.

Subsequently, I fixed the image warping by aligning corner coordinates to fit a standard 256x256 size. This transformation proved to be essential, as without it, accuracy drops to 63%!

To in-paint the black circles in the images, I dynamically identified their areas and applied a threshold to create a binary image. By converting the warped image to grey-scale and applying the threshold, I highlighted the missing regions - the black circles. Then, using contours, I identified the outlines of these regions and generated corresponding masks. Finally, I utilized the **cv2.inpaint** function with a radius of 30 to fill these regions, resulting in enhanced visual appeal and classifier accuracy. Without in-painting, the accuracy decreased to 94%.

I then moved on to adjusting the contrast and brightness of the image using **convertScaleAbs**. I wasn't sure what the best values of brightness and contrast were, so I created a few python programs outlined on the next page, to be able to run a wide range of α (contrast) and β (brightness) values, and subsequently graph it on Desmos, where:

$$x = \alpha, \quad y = \beta, \quad \text{and} \quad z = \text{accuracy of classifier}$$

so that then I could try and figure out if there were any local maxima that I could use for my beta and alpha values.

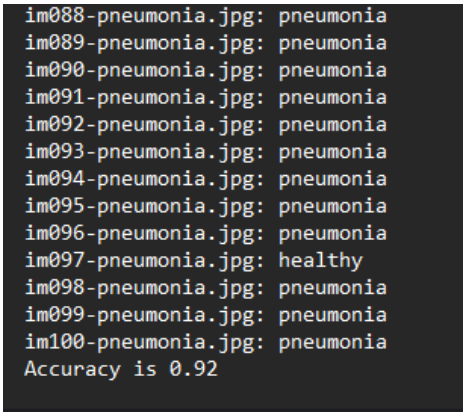


Figure 1: **changeAlphaBeta.py** - Output of classifier.py on a certain value of α and β

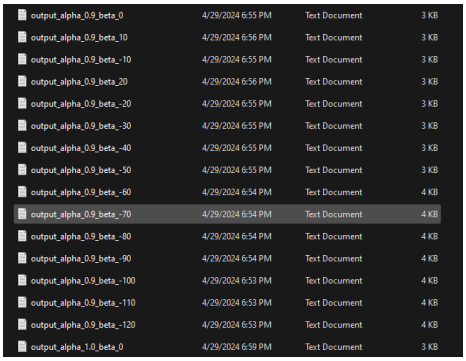


Figure 2: **alphaBeta.py** - changeAlphaBeta.py ran on all specified values of α and β

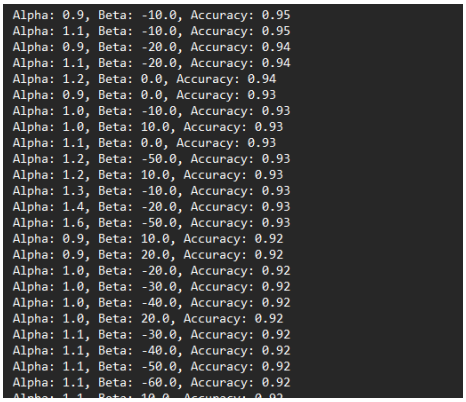


Figure 3: **readAccuracy.py** - All α and β in descending order of accuracy, saved in Accuracy.txt

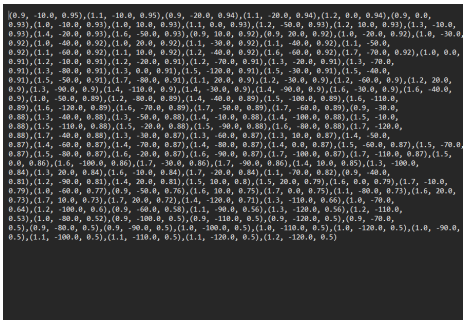
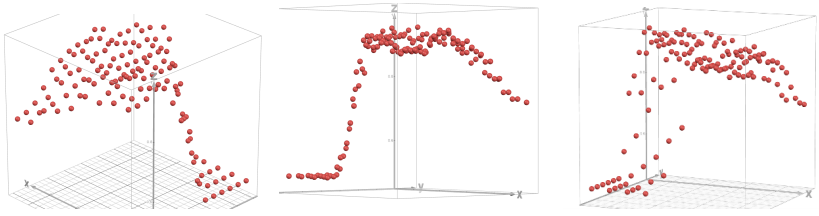


Figure 4: **accuracyDesmos.py** - All the values in Accuracy.txt in a format that can be inputted directly into Desmos to produce the 3D graph, where x = α, y = β and z = Accuracy



(a) Top head-on angle (b) Side angle (c) Top back angle

Figure 5: 3D graph of contrast and brightness

I initially checked a very large range of values of alpha and beta, and managed to narrow it down to

$$\alpha \in \{0.9, 1.0, 1.1, \ldots, 1.8\}$$

$$\beta \in \{-120, -110, -100 \ldots, 20\}$$

After running it on these values, I got the following shape (Shown in Figure 5): `https://www.desmos.com/3d/ui1jopyy16`. This gave me quite a few maxima to examine, which all yielded the final 95% accuracy. Namely, 0.9, -10, 1.1, -10, 0.9, -20. After examining all of these by eye, I concluded that alpha=1.1 and beta=-10 yielded the most appealing results. Without contrasting and brightness adjustments, accuracy drops to 91% . Finally, I increased colour saturation. This was done using **cv2.cvtColor**, and the entire function effectively just converts the image into an HSV colour space, increases the saturation channels values by 1.5, while clipping them to the range [0,255] and then converts the modified image back to the BGR colour space. Without saturation, accuracy is also reduced to 91%.

Evaluation

Quantitative evidence of performance improvement is presented through showcasing the pre-processing steps impact on the overall accuracy of the classifier. It can be easily deduced that Warping the image had the biggest effect on performance, and in-painting actually added the least to the images.

Qualitative evidence can be seen in direct comparison of the original images, and the images produced from my image processing. As you can see, the original image is angled and warped, has a large amount of noise and colour distortion, and ofcourse, contains the black circle. In the processed image, all noise has been removed, it's been fit to fill the 256x256 dimensions, its in-painted the black circle, and had the colour saturation and contrast increased significantly. It's fair to say that the processed image is a lot easier to view, not just from a classifier perspective, but from the human eye as well.

Finally, the exploration of contrast and brightness adjustments showcases heuristic-driven optimization, which is crucial in image enhancement. Using a wide range of python programs to inspect the parameter space effeciently, the solution identifies optimal values based on accuracy maxima found in the Desmos 3D Graph, and in textbfAccuracy.txt. This type of validation ensures both improved classifier accuracy and and enhanced visual quality of pre-processed images.

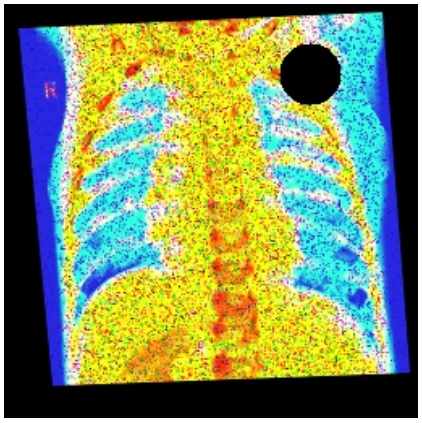


Figure 6: Original Image

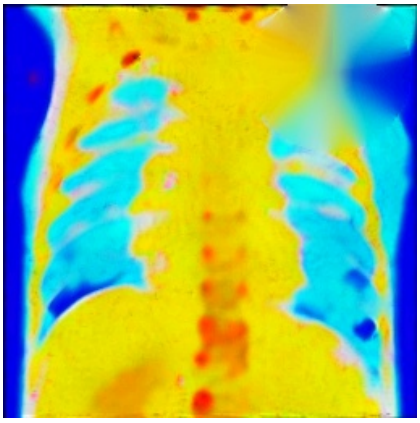


Figure 7: Enhanced Image