SAIPS Home Exercise - Defects Detection
Nadav Carmel
July 2, 2021

1. Executive summary:

Defects detection approach and its results are covered here.
While performance is not satisfactory on this dataset given the implemented approach, few improvements are suggested (I had no time to implement them as well), which potentially can gain more impressive results.
Also, I hope this report can be an opportunity for the company to get the impression on my technical and analytical skills.
Code is attached and can also be cloned from: https://github.com/NadavCarmel/defects_detection

2. Introduction:

We have pairs of images (inspected, reference), where two pairs containing some defects and one pair does not.
It is assumed that inspected and reference images are highly similar, except of:
- Small relative translation
- The defects that exist in one but not in the other.

We are also given with labels: the (i, j) coordinates of the (center I assume?) of each defect.

We want to construct an algo which gets such pair of images as an input and return the binary images of the detected defects in the inspected image.

Practically for this exercise, this means 2 main things:
- Align the reference image such that it's as matched as possible to the inspected one.
- Subtract the aligned reference image from the inspected image, and by that, transform the problem into an (weekly supervised) anomaly detection problem, where each pixel is either anomalous (of a defect) or not.

3. Methodology:

3.1. Step 1 – Image Alignment:

There are many methods for image alignment with available implementations on opencv etc.

If our images had also relative scale / rotation to calibrate, we could benefit from (for example) landmark-based approaches (for example SIFT for finding the landmarks with following RANSAC for the matching step).

In our relatively simple case, I chose to use 2 approaches 'out of the box', and implement one of my own for this alignment step:

3.1.1.1. Phase-Correlate

: this opencv implementation maximizes the correlation between the 2 images, by transforming them into Fourier-space. The idea is that:
- the multiplication in Fourier space is equivalent for convolution in the original space
- the Fourier transform of an inverted function (f(-x)) is the complex conjugate of the Fourier of the function
- a convolution of f(x) with g(x) is the correlation of f(x) with g(-x)

Thus, by applying the below formula, we get the correlation between the 2 images:

$$corr(I_{ins}, I_{ref}) = IFFT(FFT(I_{ins}) \cdot FFT(I_{ref})^*)$$

Where I_{ins} is the inspected image and I_{ref} is the reference image, and by looking for the *argmax* of this resulting array we can estimate the relative translation.

3.1.1.2. Enhanced correlation coefficient maximization (EEC): this opencv implementation (*Georgios et al.*) basically should be very similar to the Phase-Correlate method. It minimizes the L2-norm of the normalized inspected and reference images, thus maximizing the correlation: $\left(\frac{x}{|x|} - \frac{y}{|y|}\right)^2 = 2 - 2\,corr\left(\frac{x}{|x|}, \frac{y}{|y|}\right)$

But I tried it anyway and got slightly better results than the Phase-Correlate, due to the additional preprocessing of the images (gaussian blurring).

The displayed results in this report are based on this alignment.

3.1.1.3. Convolutional filter matching: this approach, which I thought of and implemented in the code (see: ./src/align_by_convolutional_filter.py), works by running a k-by-k convolutional filter, centered around pixel (i, j), on the reference image, and minimizing the squared distance to the corresponding pixel (i, j) in the inspected image.

Say the reference image is translated by +2 pixels in x direction and +3 in y direction w.r.t. the inspected image, the learned filter will be all zeros except of '1' -2 pixels in x direction and -3 pixels in y direction from its center.

In practice, this is formulated as a liner system of equations (#strides equations, k^2 variables), and is solved straightforward (using pseudo-inversion).

- The downside is that we cannot calibrate a translation larger than $k/2$.

If translation is too significant, and we set k to some large number, we'll be getting too many degrees of freedom in our system of equations and probably ill-conditioned matrix.

I'll cover a possible improvement (which was not implemented) for this issue in Recommendations section.

3.1.2. Step 2 – Anomaly Detection:

Now, that we have our reference images all aligned (to a reasonable degree), comes the second part of classifying the pixels.

The idea I had implemented here was to subtract the inspected image from the aligned reference image and get a new 'error' image (I_{error}), from which classification can arise. Intuitively, a high-valued pixel in this new image can imply a defect (since the 2 original images are assumed to be very similar aside of the defects), but we need to somehow use our labels to optimize the classification threshold.

By this representation, the problem is transformed into a weekly supervised anomaly detection problem.

The approach I took to find the threshold using I_{error} was:

a. Calculate the mean and variance of the non-defected pixels
b. Calculate the mean and variance of the defected pixels
c. Assume each pixel is drawn from a normal distribution with either defected / non-defected distribution params.
d. Repeat those steps over the 2nd pair of images ('case 2')
e. Average those 4 statistics ($\mu_{defected}, \sigma_{defected}, \mu_{non-defected}, \sigma_{non-defected}$) over the 2 sets.

f. On inference (say ‘case 3’ pair), construct the ‘error’ matrix, and compare the likelihood of the defected and non-defected distributions per each of the pixels.

See: ./src/estimate_defects_model.py and ./src/inference.py for the implementation.

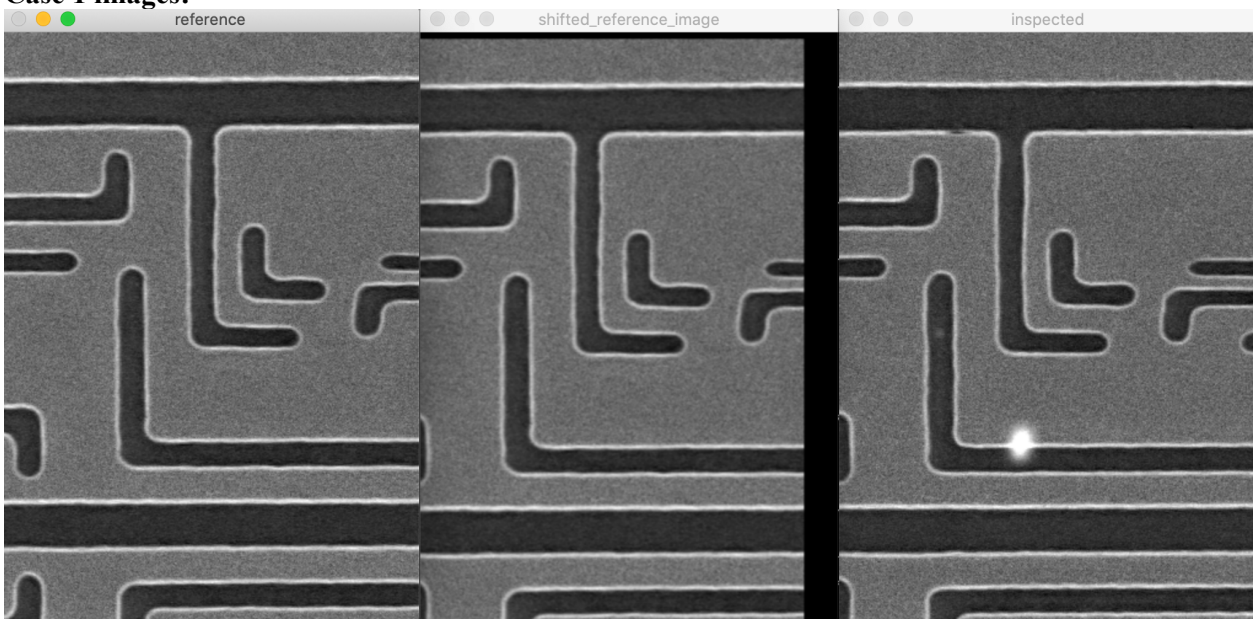
3.2. Results:

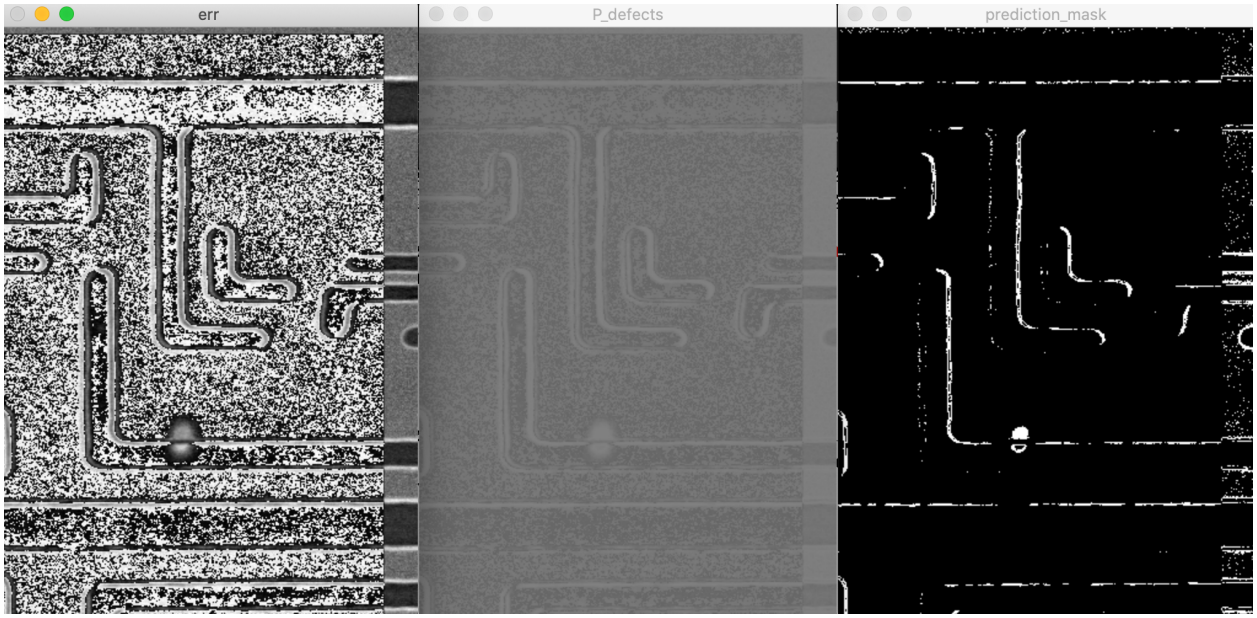
In this section I let the images speak for themselves, and do not get into the commonly used metrics (TPR, FPR, Accuracy, precision etc.).
The main reason is that these metrics will not be accurate, as the defects are not properly labeled (not all pixels of the defects are marked in the given data, and the number of defected pixels is in-fact much larger).
As a bottom line, it seems the results are not good enough, with many pixels (especially on some patterns-line-edges) detected as false-positives, and some defected pixels (especially on small defects) which are not detected.
I will cover some suggested ideas to enhance performance in the Recommendations section, which I had no time to implement.

Per each case, there are 6 images attached:

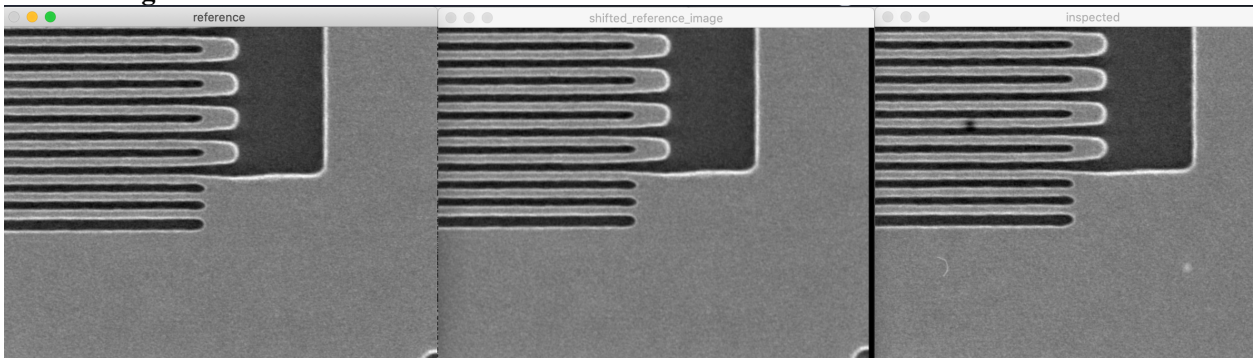
1. Inspected image
2. Reference image
3. Shifted reference image
4. Diff between the inspected and the shifted reference images (I_{error})
5. Probability of each pixel to be considered as defect according to the model ($P_{defects}$)
6. Defects prediction binary image (prediction_mask – this is what we mainly want to see)

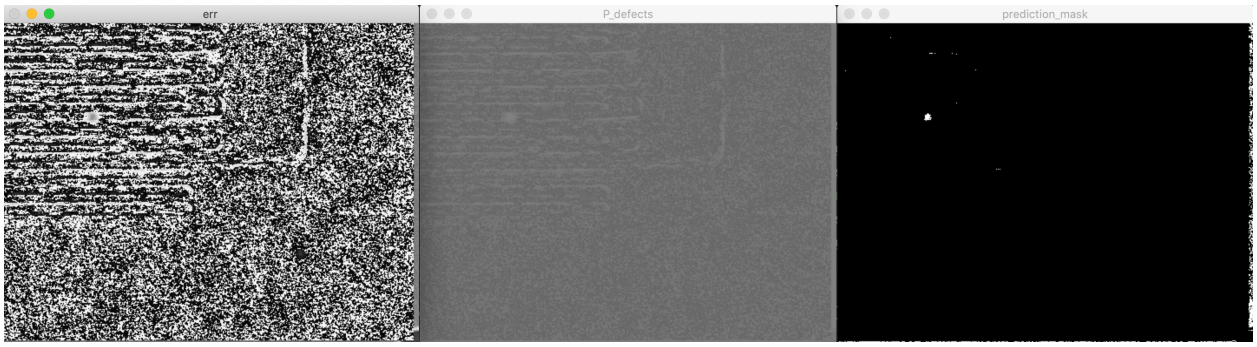
- **Case 1 images:**



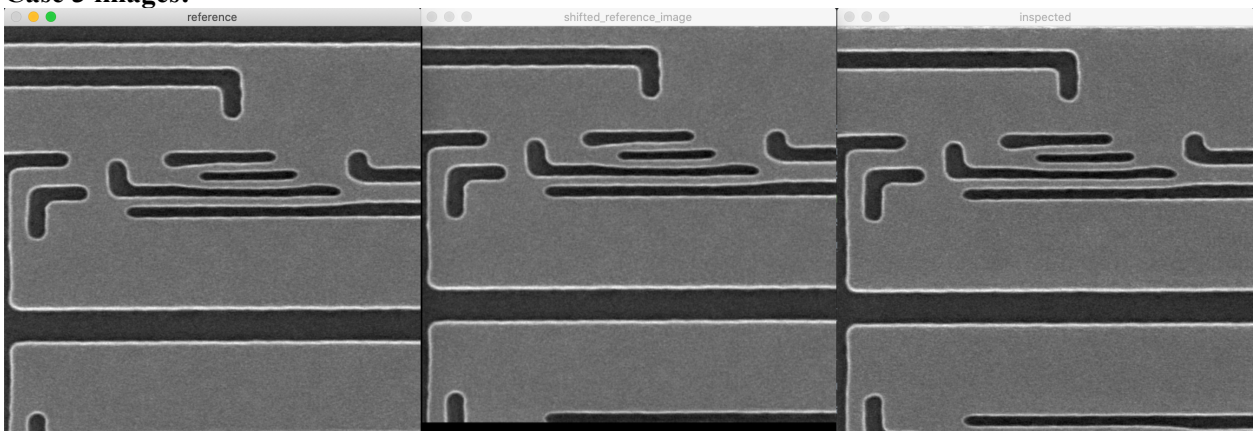


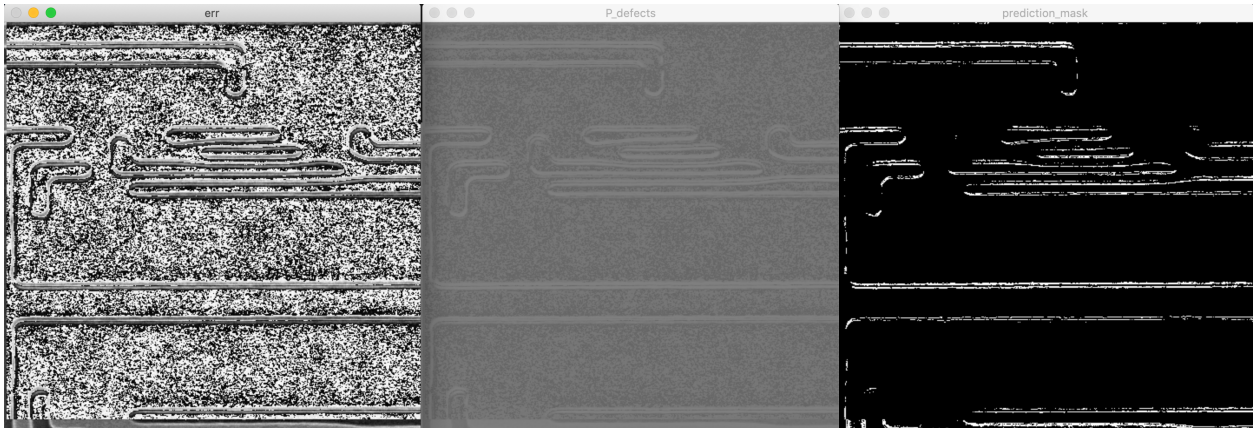
- **Case 2 images:**





- **Case 3 images:**





3.3. Conclusions:

We have demonstrated a way of locating defects in images pairs. Since we did not get the results we wish for, additional efforts should be applied on the improvement of the algorithmic flow, or possibly use a different algorithm.

3.4. Recommendations:

There are several steps that can be taken to improve classifier performance on this problem:

a. Improve labeling of the 'training data' – apply some heuristic to account for all defected pixels, and not just the center of it. This will dramatically increase the number of positive observations (pixels), resulting in the option of utilizing a more powerful model while maintaining generalization, and also, reducing noise of the negative observations (currently the pixels surrounding a defect-marked pixel are labeled as negative, although they are truly positive as well, which is practically a noise in the data).

b. Giving lower a-priory defect-probability to the edges (a difference in 2 edge pixels valued at ~255 will be much higher than 2 'mid-region' pixels valued at say ~100).

c. Improve convolutional matching method (method at 3.1.1.3, if we wish using it) by iteratively applying a decreasing-effect blurring followed by convolutional filter estimation then corresponding translation. This should converge to the optimum translation even if it is initially exceeding k/2 (half the filter size).

d. Interesting but slightly complex approach: since we are very limited in the number of defected pixels, we can try artificially enhancing their count by generating pairs of:

(defect, defect), (defect, non-defect), (non-defect, non-defect).
The scheme is as follows:

1. Construct the 'error' matrix from the subtraction of the inspected and aligned-reference images
2. Generating X, y observation by running a window over the matrix (say with (1, 1) stride), where each window will be some observation X_i in X and each label $y_i \in \{-1, +1\}$ is determined by the pixel at the center of the window.
3. Construct a new enriched dataset by composing the pairs mentioned above, where the m^{th}observation is the concatenate of 2 original observations: $X_{enriched_m} = concat(X_i, X_j)$
4. Artificially assign a value per each class, say:
   y(non-defect, non-defect) = 0
   y(defect, non-defect) = 1
   y(defect, defect) = 2
5. Train a Siamese kernel filter that will receive $X_{enriched_m}$ and solve a regression problem to predict y.
6. Use some anchor original observations for prediction of any new window on inference time.

Thank you for reading this report.