Converting past to present: Turning historical hand-drawn architectural designs into vectors using autoencoder and corner detection algorithms

Mark Jeremy G. Narag
University of the Philippines
Diliman
Quezon City, Philippines
mnarag@nip.upd.edu.ph

ABSTRACT

I vectorized heavily stained historical architectural designs dating back to 1938. The process involves initially extracting the lines through binarization. Due to the deep-seated stains accumulated over time on the original linen drawings, traditional thresholding techniques failed to binarize them. Therefore, I developed several autoencoders to automate the binarization of all 26 drawings in the archive. The approach involves training the model to clean one drawing first, then applying the learned model to clean the rest of the drawings. Overall, I created four pairs of models (deep and shallow) with varying input sizes: 32×32, 64×64, 128×128, and 256×256. After binarization, the lines of the drawings are traced to produce vectors. This is achieved by extracting the corners and endpoints of the lines using various corner detection algorithms. The coordinates of the extracted corners are then utilized to draw the vector lines. My results demonstrate success in both binarizing and vectoring the architectural designs. Remarkably, all the models effectively binarized the drawings by successfully separating the pronounced stains from the drawn lines, achieving up to a 97.7% F1 score and a 95.5% IOU score. For vectorization, template matching yielded the best results in accurately detecting the corners and endpoints of the lines compared to the Harris operator and Shi-Tomasi algorithm. Overall, I have successfully transformed heavily stained hand-drawn architectural designs into vectors, a task never undertaken before in the context of historical drawings.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

historical, architectural design, drawings, autoencoder, corner detection

1. INTRODUCTION

Cultural heritage buildings serve as profound symbols of a community's rich history and collective identity, making their conservation a matter of utmost importance. To preserve them, adaptive reuse (AR) is a recognized strategy [8]. AR is the repurposing of old buildings into contemporary usage, ensuring their preservation [4]. Not only does it retain their heritage attributes and values [3, 9], but it can also reduce the production of demolition waste [15] and construction costs [2, 3, 11, 15].

In the Philippines, AR was employed to transform the historical 1938 Agriculture and Commerce Building into the current National Museum of Natural History. This neoclassical building, designed by Filipino Architect Antonio Toledo in 1938, was originally constructed in 1940, just before the onset of the Second World War. Unfortunately, it was destroyed during the Battle of Manila in 1945, but the building was reconstructed based on the original plans after the war. It served as the Department of Tourism office until 2015. In 2012, the National Museum (NM) began planning the conversion of the building into the National Museum of Natural History [1]. NM architects retrieved the original 26 architectural drawings from the Department of Public Works and Highways in 2012. In the same year, our group digitally scanned them using NIJI - a digital scanner from Kyoto University. Because the plans had been sketched by hand on linen canvas and subsequently stored rolled up, the canvas accumulated dirt and stains.

These architectural drawings are crucial in guiding the renovation process. For architects and engineers, it is essential that they are digitized and converted to vector format. However, historical documents are susceptible to degradation due to physical and chemical weathering, biodeterioration, or neglect. The stains can sometimes be as dark as the lines of the drawing. Thus, it is hard to directly convert them into a vector format. The architectural drawings should be digitally "cleaned" first such as through image binarization.

In fact, back in 2015, our group was able to provide promising results using difference-of-Gaussians (DoG) filter in comparison to traditional edge detection and image segmentation techniques [6]. However, DoG requires finding the optimal parameters per drawing. After 10 years since scanning

it, in this study, I revisited this same data set in the hopes of finding a universal solution.

A work similar in aim to mine is a paper of Suh et al. in 2022 [12], where they binarized low-resolution, hand-drawn floor plans from National Archives of Korea. They extracted 24 features per pixel and fed them to a random forest to classify whether the pixel is black or white. However, while floor plans and architectural designs are both forms of line art, architectural designs tend to be more intricate and detailed than floor plans. Hence, their method may not be directly applicable to my work, as architectural designs often include exterior, front, side, and isometric views, whereas floor plans typically only provide a top-down view of interior spaces. Although it was not their primary focus to binarize an image, Tabia et al. in 2017 [14] generated a 3D version from architectural designs, in which, one of their primary steps is binarizing the design first through traditional thresholding technique. Aside from floor plans and architectural designs, several studies have also attempted to binarize degraded text documents both in traditional techniques [7, 10] and deep learning approach [16, 13].

In this paper, I propose to vectorize historical architectural drawings by first binarizing it using convolutional autoencoder models. Once the "clean" version is extracted, I will perform Shi-Tomasi corner detection operator and template matching algorithm to detect all the relevant end points on the whole image that represents a line. Finally, the coordinates of these end points will then be used to draw the vector lines that traces the original image.

1.1 Novelty

To the best of my knowledge, no work has attempted to vectorize hand-drawn architectural designs. While there have been extensive efforts to binarize stained historical documents of various types, including musical notes, palm leaves, and texts, none have specifically addressed architectural designs. Additionally, all of which focus on low-resolution images. The closest example is the work of Suh [12] on floor plans. In this paper, I present a novel approach to binarization applied to high-resolution images of architectural designs. This paper is also the first of its kind to undertake the further vectorization of historical designs.

2. METHODOLOGY

This paper is divided into two parts. The first part involves binarization of the architectural designs in order to extract the "clean" version. From this, the second part involves vectorization through corner detection.

2.1 Binarization

The 26 architectural drawings are scanned at a high resolution of 470 dpi. The actual linen size is approximately 100 by 80 cm. To train the autoencoder model for image binarization, we need both the input (dirty images) and the desired output (binarized images) data. Since I only had access to the dirty images, I had to create my own clean and binarized version. To do this, I initially binarized the dirty image using a traditional thresholding technique as a starting point, avoiding the need to begin from scratch. The output was still dirty, as expected, with the patterns of the linen and the outline of stains still apparent. As a result, I resorted to manually removing or cleaning them using GIMP software. The cleaning process took 21 hours to complete. Shown in Figure 1 is the final clean and binarized version of a dirty image which I will use as the output in our model. We chose quadrant 3 of drawing 1 to for the training since it contained the largest stain, and therefore was the dirtiest section of the image. The reason for this is to train the model with many dirty-to-binarized pairs to ensure that it can generalize well to other dirty images.

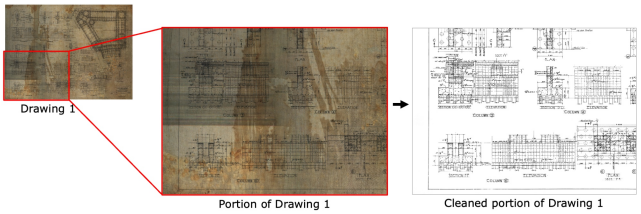

Figure 1: Creating the desired output of the model by manually binarizing a portion of drawing 1.

Given the high resolution of the images, I performed binarization locally by dividing the input and output pairs into small, overlapping patches of either 32×32, 64×64, 128×128, 256×256, depending on the model. The total extracted patches ranges from 22, 484 to 22,644. For training and validation, I applied a 70-30 split. Table 1 provides a summary of the architecture of each model which shows the image dimensions after each convolution and deconvolution operations. As we can see, odd numbered models are deep networks while even numbered models are shallow networks with only one hidden layer.

Table 1: Details of the autoencoders used for binarization.

Model	Architecture
1	32×32 → 16×16 → 8×8 → 16×16 → 32×32
2	32×32 → 16×16 → 32×32
3	64×64 → 32×32 → 16×16 → 8×8 → 16×16 → 32×32 → 64×64
4	64×64 → 32×32 → 64×64
5	128×128 → 64×64 → 32×32 → 16×16 → 8×8 → 16×16 → 32×32 → 64×64 → 128×128
6	128×128 → 64×64 → 128×128
7	256×256 → 128×128 → 64×64 → 32×32 → 16×16 → 8×8 → 16×16 → 32×32 → 64×64 →128×128 → 256×256

A kernel size of 3×3 with a stride of 2 was used for each convolution and deconvolution operators with rectified linear unit (relu) as the activation function. A stride of 2 was chosen to avoid using max pooling operation in order to reduce the computational cost of training the models. I also used adam solver for optimization with default learning rate of 0.01. The models were built using the keras-tensorflow framework in python scripts. I ran the models for 100 epochs on either Google Colab (models 1 to 4) or Google Colab Pro (models 5 to 8). Google Colab offers 12.7GB RAM and 15GB GPU as part of its free software, whereas Google Colab Pro, requiring a subscription fee of 10 dollars per month, provides greater computational power with 83.5GB RAM

and 40GB GPU. The choice of using these two software platforms stems from the fact that the complexity of the model directly correlates with the computational power required. The free version of Google Colab can not handle models 5 to 8.

2.2 Vectorization

The vectorization can only proceed after removing the heavy stains on the architectural design. Without binarization, the stains can be detected as corners in this next step. The vectorization process has two parts - corner detection and line tracing. I consider two algorithms for corner detection, that is, Shi-Tomasi corner detector and template matching.

2.2.1 Corner detection using Shi-Tomasi

The Shi-Tomasi corner detection algorithm builds upon the Harris corner detector, introducing a refined corner response function to enhance its performance. The cornerstone of the algorithm lies in the computation of the corner response function for each pixel.

Given an image with intensity values represented by the matrix I, the gradients I_x and I_y are computed using, for example, Sobel operators:

$$I_x = \frac{\partial I}{\partial x}$$

$$I_y = \frac{\partial I}{\partial y}$$

These gradients are then used to calculate the structure tensor M at each pixel:

$$M = \begin{bmatrix} \sum_{\text{neighborhood}} I_x^2 & \sum_{\text{neighborhood}} I_x I_y \\ \sum_{\text{neighborhood}} I_x I_y & \sum_{\text{neighborhood}} I_y^2 \end{bmatrix}$$

The eigenvalues λ_1 and λ_2 of M are then computed using standard eigenvalue solvers. The corner response function R is defined as the minimum eigenvalue:

$$R = \min(\lambda_1, \lambda_2)$$

This modification makes the Shi-Tomasi algorithm less sensitive to noise, enhancing its performance in corner detection compared to the original Harris detector.

Figure 2 illustrates a sample result of the Harris and Shi-Tomasi operators for various cases of line images. Initially, I tested them on two synthetic images to discern their efficacy and limitations. The first row of the figure features uniformly thin lines, while the second row exhibits varying thickness. It is evident that both Harris and Shi-Tomasi successfully detected the corners in the thin lines. However, for thick lines, both operators erroneously identified four corners instead of two, indicating a tendency to treat thick lines as rectangles. Thus, it becomes apparent that, in the context of corner detection in line images, the lines should be thin for both operators to perform optimally.

The third row displays the actual portion of the binarized drawing, where the lines are thick and rough, leading to failure in corner detection by both operators. Although corners were detected, numerous in-between corners inside the lines were also erroneously identified. To address this issue, I preprocessed the drawing using various transformations such as resizing, blurring, dilation, and erosion. Upon investigation, the most effective transformation involved two dilations followed by five erosions, and then a Gaussian blur of size 3×3. The fourth row presents the result, demonstrating considerable success for Shi-Tomasi in detecting the desired corners with fewer in-between corners compared to when the original image is used. Given that the lines in the original architectural designs are nearly uniform in thickness, it is reasonable to assume that this pre-processing step will also be effective for the rest of the drawing.

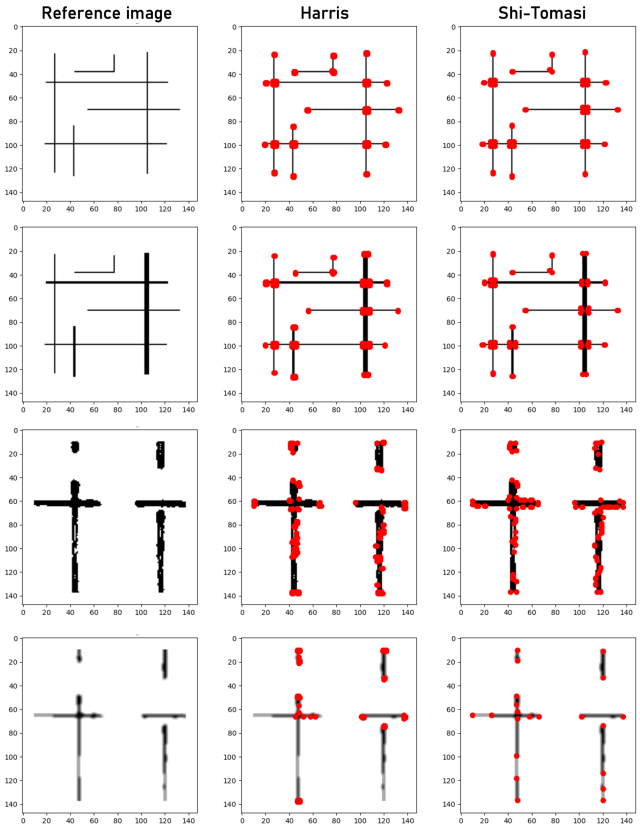

Figure 2: Corner detection for synthetic image with uniform thickness (1st row), synthetic image with varying thickness (2nd row), actual portion of the drawing (3rd row), and preprocessed actual portion of the drawing with twice erosion, five times dilation, then 3 by 3 Gaussian blur (4th row).

2.2.2 Corner detection using template matching

Template matching is a technique used for identifying instances of a specific pattern, known as the template, within a larger image. In the context of object detection, the template represents the target object, and the algorithm seeks to locate occurrences of this object within a reference image. Shown in figure 3 are the templates used which represents the possible scenarios of a section of an image being the end of a line. There are 20 templates in total with varying

thickness.

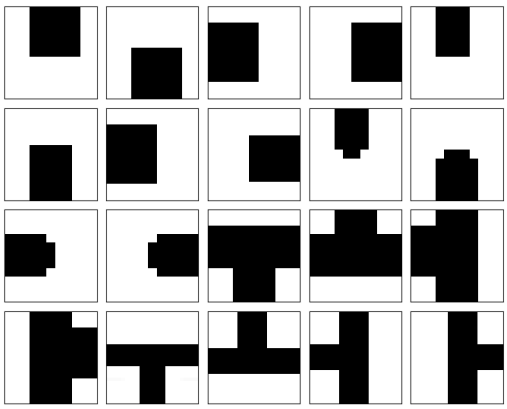

Figure 3: 20 templates used for the template matching. They have a size 11×11 except for the last four with 25×25.

Let I be the reference image and T be the template image (mask). The goal is to find the position (x, y) in the reference image where the template best matches. The measure of similarity between the template and a region in the reference image is often computed using the normalized cross-correlation:

$$\text{Corr}(x,y) = \frac{\sum_i \sum_j [I(i,j) \cdot T(i-x, j-y)]}{\sqrt{\sum_i \sum_j I(i,j)^2 \cdot \sum_i \sum_j T(i-x, j-y)^2}} \tag{1}$$

where $I(i,j)$ and $T(i-x, j-y)$ represent pixel values at positions (i,j) in the reference image and template, respectively.

The template matching algorithm slides the template over the reference image, calculating the correlation at each position. The result is a correlation map, denoted as R, where each element $R(x,y)$ represents the similarity measure at the corresponding position.

To identify potential matches, a threshold is applied to the correlation map. Positions where $R(x,y)$ exceeds a predefined threshold are considered as potential matches:

$$\text{Potential Match}(x,y) = \begin{cases} 1 & \text{if } R(x,y) \geq \text{Threshold} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

To avoid redundancy in the detected matches, I implemented a post-processing step in filtering closely located centers. A distance threshold is defined, and only one representative center within this threshold is retained. The distance between two centers (x_1, y_1) and (x_2, y_2) is calculated as:

$$\text{D} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3}$$

Centers are considered duplicates if D $\leq$ D thresh.To illustrate, Figure 4 shows a sample process in template matching. Similar to the Harris and Shi-Tomasi methods, the image undergoes a pre-processing stage, with the optimal transformation identified as a single erosion followed by two dilations and a Gaussian blur with a kernel size of 3×3. In the first image of the figure, multiple corners are initially detected, with the large red circles representing clusters of duplicate corners. The second image in the middle, showcases the outcome after applying post-processing. The blue dots now serve as representatives for each clustered point. Lastly, the third image displays the final corners of the image.

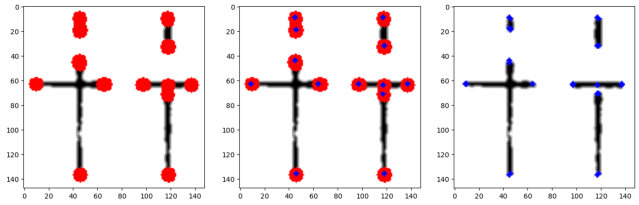

Figure 4: Multiple detected corners using template matching (left). Filtered corners, in blue dots, from the cluster of multiple detected corners (middle). Final detected corner (right).

2.2.3 Line coordinates tracing

Once the coordinates of the edges and corners of the lines are obtained, the subsequent challenge is to accurately determine which corners should be connected to trace the lines aligning with the original image. In other words, given that the end of a line is defined by coordinates $[(x_i, y_i), (x_j, y_j)]$, the task is to identify these coordinates from the extracted collection of N coordinates $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$. A potential solution involves drawing lines from all possible coordinate pairs. For instance, starting with (x_1, y_1) as a reference point, connections will be established to all N coordinates, excluding itself. The drawn lines will function as masks applied on top of the original drawing. For each pair, the mean pixel intensity values within the region of interest (mask) will be computed. The line with the lowest value will be identified as the correct pair for (x_1, y_1). Figure 5 visually illustrates this process for clarity.

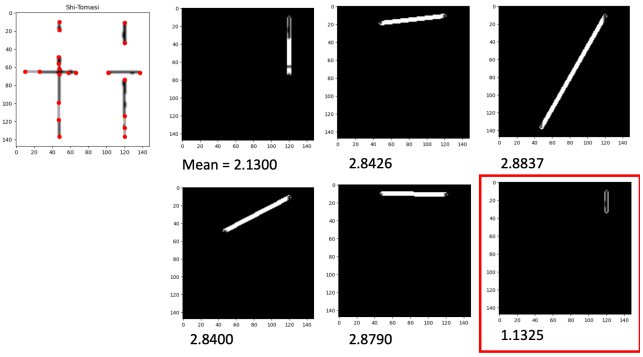

Figure 5: Process for correctly identifying the line that matches the original image. The red box has the lowest mean pixel intensity value which matches the original image.

Finally, after identifying the correct pair of coordinates, these

will be used to draw the vector lines.

2.3 Evaluation metrics

For the binarization, I quantify the resulting output of each model using three metrics - F1 score, intersection-over-union (IOU), and peak signal-to-noise ratio (PSNR). F1 score is the harmonic mean of precision and recall. Precision measures the ability of the model not to label negative instances as positive while recall is the ability of the model to identify all positive instances. Both are obtained from the confusion matrix where each pixel is classified either black (0) or white (1). F1-score is given by

$$F1score = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

where precision and recall are denoted as follows

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

where TP, FP, and FN are denoted as true-positives, false positives, and false negatives of the confusion matrix. On the other hand, IOU is the ratio of the logical AND and logical OR of the ground truth and the output image of the model. It is given by

$$IOU(f, g) = \frac{|f \cap g|}{|f \cup g|} \tag{7}$$

Lastly, PSNR quantifies the pixel differences of the ground truth and the binarized output image of the model. Given the ground turh image f and the binarized version g of the model, both of size M×N, the PSNR between f and g is given by:

$$\text{PSNR}(f, g) = 10 \log_{10}(255^2 / MSE(f, g)) \tag{8}$$

where

$$\text{MSE}(f, g) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (f_{ij} - g_{ij})^2 \tag{9}$$

The PSNR value approaches infinity as the MSE approaches zero [5]. The higher the value of the PSNR, the closer the two images in terms of pixel difference.

3. RESULTS AND DISCUSSION

3.1 Binarization

3.1.1 Traditional thresholding

I set aside quadrant two of drawing 1 as our test image, which has never been seen by the models before. This portion of the drawing was also subjected to manual cleaning. I will compare the results of our autoencoder models with that of a traditional binarization technique. Specifically, I will use the thresholding technique where a fixed intensity value is selected as a threshold. It categorizes pixels as black if their intensity value is lower than the chosen threshold, and as white if their intensity value is higher. Since pixel values ranges from 0 to 255, this technique can yield 256 different results. Thus, I binarized the test image across all these potential threshold values and determined the threshold value that will yield the highest metric score. Figure 6 shows the binarized outputs for various threshold values. A lower threshold value tends to underdraw the image, while a higher threshold value tends to overdraw it. I identified that a threshold value of 48 yielded the highest metric scores: an F1 score of 0.9510, an IOU of 0.9083, and a PSNR of 22.07.

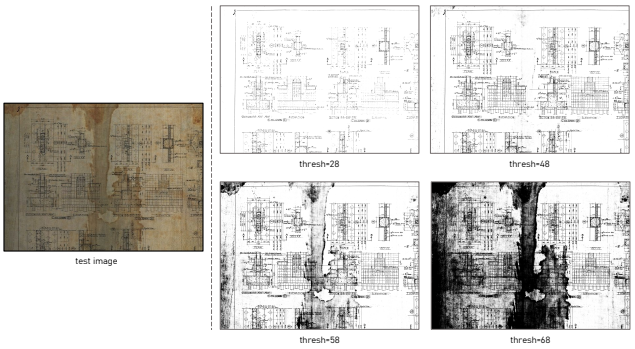

Figure 6: Traditional thresholding technique for different threshold values.

3.1.2 Autoencoders

Now, let us compare the results of our approach with the traditional thresholding technique. The performance scores of the different models for the test image are presented in Table 2. In Model 7, the metric scores are blank due to overfitting, which prevented the model from properly binarizing the test image. However, looking at the overall picture, the table clearly shows that all eight models have outperformed the traditional thresholding technique, achieving higher scores across all metrics. Notably, the metric scores of the models are consistently close to each other.

Table 2: Metric scores for the eight (8) different models and the traditional thresholding technique. Odd models - 1, 3, 5, 7 are deep autoencoders while even models - 2, 4, 6, 8 are shallow autoencoders. Highlighted in bold is the highest F1 score, IOU, and PSNR for each set. For the traditional thresholding technique, the highest score all of the threshold values is presented.

Input size	Technique	F1 score	IOU	PSNR
32×32	model 1	0.9683	0.9385	23.99
	model 2	0.9745	0.9502	24.93
64×64	model 3	0.9612	0.9253	23.11
	model 4	0.9750	0.9511	25.02
128×128	model 5	0.9654	0.9331	23.62
	model 6	0.9770	0.9551	25.4
256×256	model 7	0.9616	0.9260	23.13
	model 8	**0.9768**	**0.9547**	**25.37**
whole image	traditional	0.9510	0.9083	22.07

In the context of using a shallow or deep models, remarkably, models with shallow architectures - model 2, 4, 6, and 8, yielded higher metric scores than their deep model counterparts. This observation proves that shallow architectures are already sufficient for this binarization task. Shallow autoencoders model reduces the computational complexity and running time of a deep model, highlighting the effectiveness and, importantly, the efficiency of employing shallow autoencoders.

Now, let us identify the optimal model that strikes a balance between effectiveness and efficiency. As highlighted in Table 2, model 8 emerges with the highest metric score among all the models. However, it is important to reiterate that all models yielded metric scores in close proximity and produced nearly indistinguishable outputs. Remember that Model 8 was executed using Google Colab Pro due to its increased RAM requirement. Since shallow autoencoders have proven to be effective, we can eliminate their deep counterparts from consideration. This leaves us with model 2 and 4.

Considering that model 4 yielded higher metric scores in comparison to model 2, we can identify model 4 as the optimal choice for this binarization task. It is a shallow autoencoder with an input size of 64×64. Notably, the execution of the model is feasible on the free version of Google Colab, rendering it the ideal solution that effectively balances both efficacy and efficiency for this particular binarization task.

To provide further visual context of our best model, as shown in Figure 7, I present sample sections of the binarized outputs using Model 4, in comparison to the optimal output achieved through the traditional thresholding technique. A clear distinction between the two techniques is evident, with our approach being notably better. For all sections, we observe that the weave patterns of the linen are present when using the traditional thresholding technique, whereas our method successfully avoids detecting them. Particularly noteworthy is the third row, where a dark stain overpowered the letters, rendering them unreadable. It is possible to suggest that this can be solved by changing the threshold value, but the presented output is already the optimal value with the highest metric score. Note that lower threshold values are sensitive to detect the stains while higher values tend to underdraw the lines. When it comes to our proposed model, it is very apparent that the output of our model is nearly indistinguishable from the ground truth. This proves the effectiveness and efficiency of using just shallow networks to binarize the dirty input.

3.1.3 Vectorization

We start first with a small portion of the drawing of size 128 by 128. Note that a padding of 10 pixels for all the corners are added in order to still detect the corners for the lines along the edges of the image. Shown in Figure 8 is the resulting vector image for different samples of images. As we can see, using template matching as corner detection provides better vector image than Shi-Tomasi. Finally, the results for a larger image of size 512 by 512 in shown in Figure 9. It is consistent here that template matching provides cleaner vectorization.

4. CONCLUSIONS

I successfully binarized the historical architectural drawings of the Agriculture and Commerce Building using shallow convolutional autoencoders. These models were trained exclusively on a segment of a single drawing, resulting in the creation of eight models. All eight models outperformed the traditional thresholding technique in both quantitative and qualitative assessments. Model 4 emerged as the optimal choice within this set, featuring a single hidden layer, an

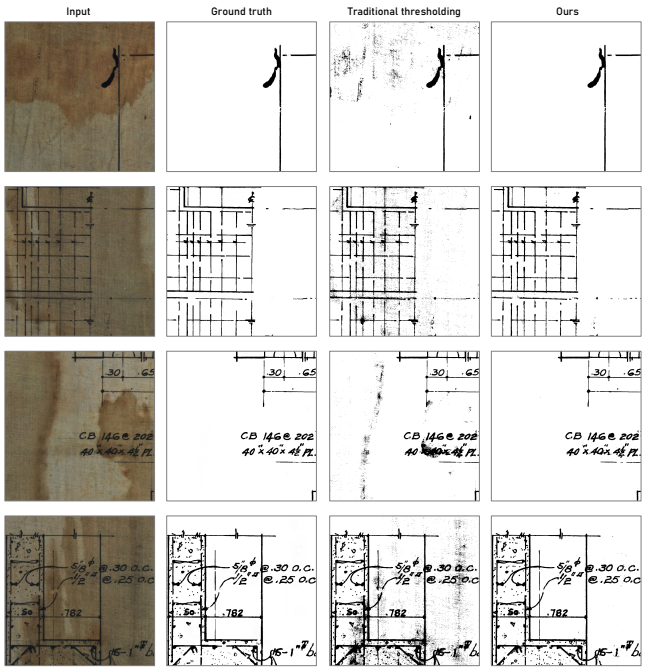

Figure 7: Binarization output using traditional thresholding versus Model 4.

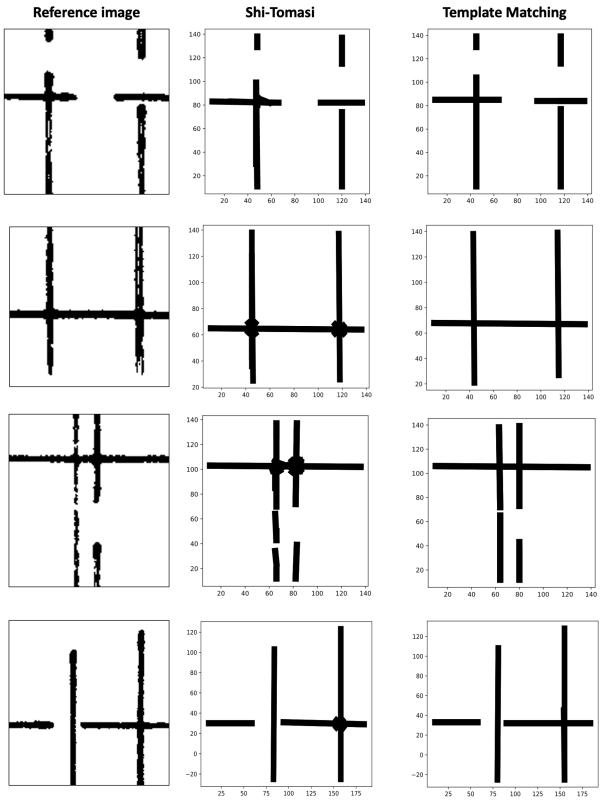

Figure 8: Resulting vector format for different reference images.

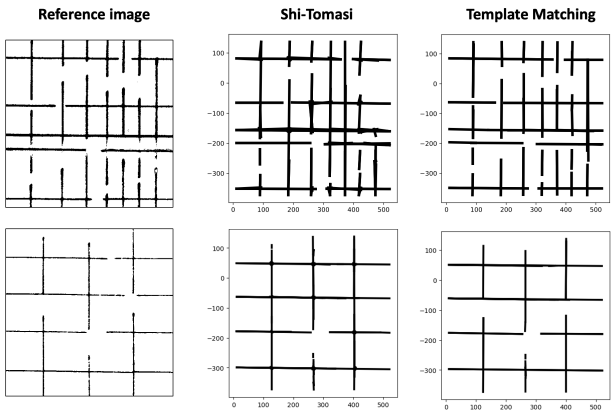

Figure 9: Resulting vector format for larger reference image dimensions.

input size of 64×64, and being executable within the free version of Google Colab.

Moreover, I achieved successful vectorization of the resulting binarized image using template matching for corner detection. Currently, my vectorization process is limited to straight lines. For the curved lines present in the architectural design, I recommend utilizing a Hough transform in future applications. Additionally, I suggest employing RANSAC during the line-tracing process, which involves pairing all the coordinates.

In summary, transforming heavily stained hand-drawn architectural drawings into vector format is a challenging task that had never been attempted before. This study marks the first endeavor to do so.

5. ACKNOWLEDGMENTS

I acknowledge the Department of Works and Highways and the National Museum of the Philippines for the architectural designs used in this study.

6. ADDITIONAL AUTHORS

7. REFERENCES

[1]

[2] M. D. Alba-Rodríguez, R. Machete, M. Glória Gomes, A. Paula Falcão, and M. Marrero. Holistic model for the assessment of restoration projects of heritage housing. case studies in lisbon. *Sustainable Cities and Society*, 67:102742, 2021.

[3] P. A. Bullen and P. E. Love. Adaptive reuse of heritage buildings. *Structural survey*, 29(5):411–421, 2011.

[4] B. L. Fatemeh Hedieh Arfa, Hielkje Zijlstra and W. Quist. Adaptive reuse of heritage buildings: From a literature review to a model of practice. *The Historic Environment: Policy & Practice*, 13(2):148–170, 2022.

[5] A. Horé and D. Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[6] B. E. Masil and M. N. Soriano. Line drawing enhancement of historical architectural plan using difference-of-gaussians filter. In *2015 International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, pages 1–5, 2015.

[7] W. Niblack. *An introduction to digital image processing*. Strandberg Publishing Company, 1985.

[8] N. Pintossi, D. Ikiz Kaya, P. van Wesemael, and A. Pereira Roders. Challenges of cultural heritage adaptive reuse: A stakeholders-based comparative study in three european cities. *Habitat International*, 136:102807, 2023.

[9] H. RemØy. Preserving cultural and heritage value. In *Sustainable building adaptation: Innovations in decision-making*, pages 159–182. Blackwell, 2014.

[10] J. Sauvola and M. Pietik§inen. Adaptive document image binarization. *Pattern Recognition*, 33(2):225–236, 2000.

[11] R. Shipley, S. Utz, and M. Parsons. Does adaptive reuse pay? a study of the business of building renovation in ontario, canada. *International journal of heritage studies*, 12(6):505–520, 2006.

[12] H. Suh, H. Kim, and K. Yu. *Machine learning-based binarization technique of hand-drawn floor plans*, 2022.

[13] S. Suh, J. Kim, P. Lukowicz, and Y. O. Lee. Two-stage generative adversarial networks for binarization of color document images. *Pattern Recognition*, 130:108810, 2022.

[14] H. Tabia, C. Riedinger, and M. Jordan. Automatic reconstruction of heritage monuments from old architecture documents. *Journal of Electronic Imaging*, 26(1):011006, 2016.

[15] E. H. Yung and E. H. Chan. Implementation challenges to the adaptive reuse of heritage buildings: Towards the goals of sustainable, low carbon cities. *Habitat International*, 36(3):352–361, 2012.

[16] J. Zhao, C. Shi, F. Jia, Y. Wang, and B. Xiao. Document image binarization with cascaded generators of conditional generative adversarial networks. *Pattern Recognition*, 96:106968, 2019.