BRINGING GRAYSCALE TO GHOST TRANSLATION

An Undergraduate Research Scholars Thesis

by

ZACHARY BOTTORFF

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Alexei Sokolov

May 2023

Major: Physics – Computational Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Zachary Bottorff, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Faculty Research Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Bringing Grayscale to Ghost Translation

Zachary Bottorff
Department of Physics and Astronomy
Texas A&M University

Faculty Research Advisor: Dr. Alexei Sokolov
Department of Physics and Astronomy
Texas A&M University

In recent decades, physicists developed ghost imaging, which is an alternative technique to the conventional imaging used everywhere by cameras. Ghost imaging was given its name because none of the light that reached the camera-like detector ever interacted with the subject of the image, yet this technique was able to produce an image of that object. This method initially utilized two correlated beams of light; one beam interacted with the object and was collected by a bucket detector without spatial resolution, and the other was simply collected by a spatially-resolved detector. Then, a single-beam method was developed known as computational ghost imaging, where a device modulated the spatial pattern of the only beam. The propagation of the light could be computed, allowing the object to be imaged without a spatially-resolved detector at all. Deep learning techniques were then adopted from computer science to improve computational ghost imaging.

In the past few years, researchers have begun utilizing a type of deep learning network called a Transformer network, resulting in a regime known as ghost translation. This regime appears to be robust to noise and could enable major computational shortcuts when compared to ones that utilize other types of neural networks. However, ghost translation has only been developed to work on simple binary images, which do not resemble most applications found in real-life or lab-

oratory settings. I build upon this recent work, exploring the feasibility of extending this regime from binary images to grayscale ones. I take concrete steps toward creating a network that can use such images and identify promising new directions, suggesting that such a network is in the near future. Grayscale images are found in common imaging applications, and they are the step immediately preceding full-color images that are the hallmark of conventional imaging. Hence, improving this regime to give it compatibility with grayscale images opens the door to useful laboratory applications and promotes the discovery of further uses for computational ghost imaging. While a fully-capable grayscale Transformer network is not yet here, I bring it several steps closer in this work.

DEDICATION

To my peers and my family, who have provided invaluable support throughout my entire undergraduate career.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Alexei Sokolov, and postdoctoral researcher Dr. Tao Peng for their indispensable help and guidance throughout this research.

My thanks go to Dr. Zhenhuan Yi for his guidance regarding the use of campus computing resources.

I am very grateful to the Undergraduate Research staff for their assistance in the entire thesis writing process.

Thanks to Wenhan Ren, Xiaoyu Nie, Dr. Tao Peng, and Dr. Marlan O. Scully for developing the foundational code upon which this project was built. My sincere appreciation goes to Wenhan Ren for her patient help in interpreting the initial code and for providing me with a starting point.

Finally, thanks and love to my fiancée for her continual support and prayers.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Portions of this research were conducted with high performance research computing resources provided by Texas A&M University (https://hprc.tamu.edu). Otherwise, this research did not receive any direct funding.

NOMENCLATURE

GI	Ghost Imaging
CGI	Computational Ghost Imaging
CW	Continuous Wave
SLM	Spatial Light Modulator
DNN	Deep Neural Network
DMD	Digital Micromirror Device
HPRC	High Performance Research Computing
trg	Target (Transformer network desired output)
src	Source (Transformer network input)
MNIST	Modified National Institute of Standards and Technology
MSE	Mean square error
SNR	Signal-to-noise ratio

1. INTRODUCTION

For many decades, the accepted method for producing images has involved collecting photons after they reflect off of the object of study. When this method of conventional imaging was first used by humans as a tool, it involved the use of film that reacted when exposed to light, but it has since been adapted to an array of digital detectors. This is the basic principle behind the camera, and even behind human eyes. Photons bounce off of an object and into the lens of the camera. Inside, there is an array of detectors. When a photon is registered in one of these detectors, the location of the detector corresponds directly to where that signal belongs in the resulting image. By synthesizing the signals from many of these detectors, a two-dimensional image can be created. Similarly, in the eye, the location of the photon upon the retina directly informs the brain of where the light originated. This feature that a light detector, whether in a camera sensor or a retina, can collect images by using the location of incident light upon it is known as "spatial resolution." The layperson may wonder if it is even possible to collect images without any spatial resolution at all, only the intensity of light. However, spatial resolution is not the only way to generate images of objects. Ghost imaging (GI), for example, measures photons to generate an image using a detector lacking in any spatial resolution.

1.1 Physics Background

In recent years, new methods of optical detection and image recovery have been developed. Ghost imaging was one of these techniques, and it gave rise to computational ghost imaging (CGI).

1.1.1 Ghost Imaging

Ghost imaging is a method of generating images through correlations between photons. Its first discovery utilized sources that emitted entangled photon pairs, so its mechanism was understood to be a result of quantum mechanics [1]. However, later experiments suggested that GI was not exclusively the result of a quantum effect, instead indicating that a semiclassical interpretation could be applied. This was demonstrated when ghost imaging was conducted using pseudothermal

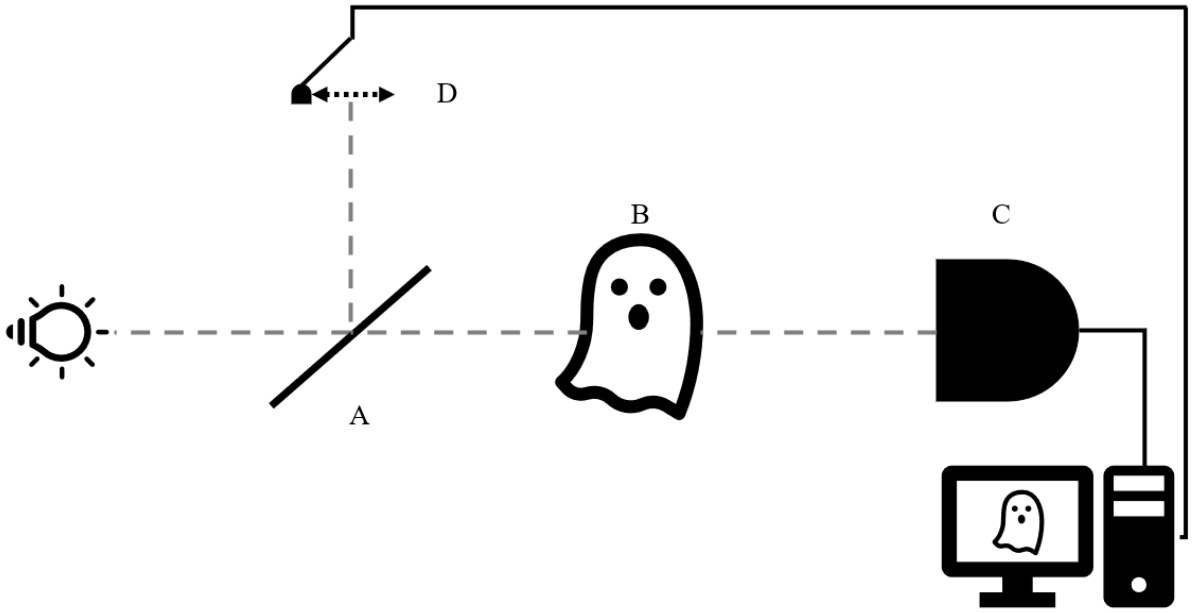

Figure 1: Example experimental setup for ghost imaging. Light is separated into two paths by beam splitter (A). One beam interacts with an object (B), and is collected by a bucket detector (C), which is not spatially resolved. The other beam is collected by a spatially-resolved pinhole detector (D).

light [1].

In such an experiment, a continuous wave (CW) laser was passed through a spatial light modulator (SLM), producing a speckle pattern of light and dark areas, or through a diffuser. This beam was passed through a beam splitter, such that half of the light passed through the object to be measured and into a bucket detector while the other half passed into a pinhole detector, as shown in Fig. 1. The bucket detector can only measure how many photons pass the object, and the pinhole detector can only measure the unperturbed speckle pattern made by the SLM or the distribution made by the diffuser [1]. Hence, unlike conventional imaging with a camera, there does not exist any image of the object with spatial resolution. However, by performing repeated measurements with different speckle patterns, the correlation between each pattern and the photon count in the bucket detector can be determined. From this correlation, the image can be constructed.

1.1.2 Computational Ghost Imaging

It is possible to simplify this setup. In a scheme known as computational ghost imaging, the beamsplitter and pinhole detector are absent. This is because the hypothetical detections of a

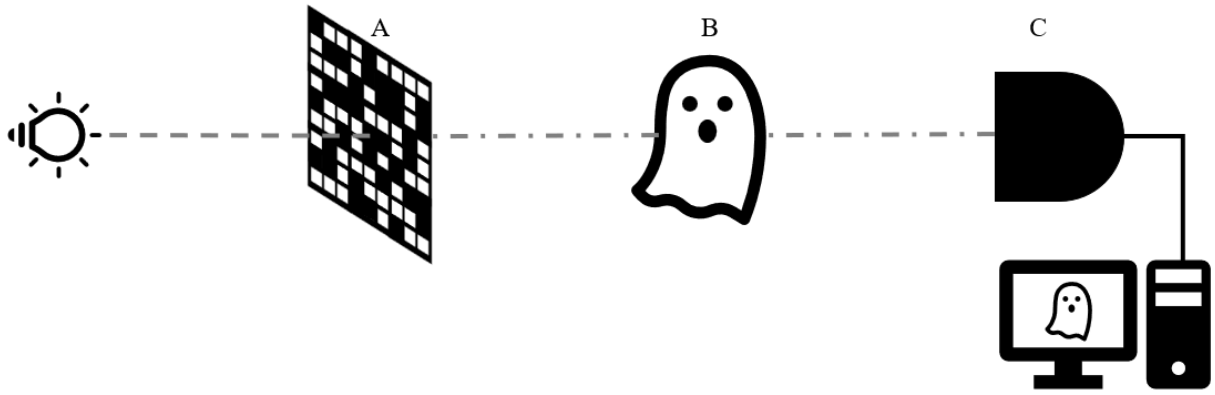

Figure 2: Example experimental setup for computational ghost imaging. Light is modulated into speckle patterns by an SLM (A), interacts with an object (B), and is collected by a bucket detector (C), which is not spatially resolved.

pinhole detector can be computed without measurement, given known, deterministic modulation on the part of the SLM [1]. Hence, the detection scheme has been simplified to a laser, an SLM, the object, and a simple bucket detector, as depicted in Fig. 2. With these components alone, it is possible to construct an image of the object with appreciable fidelity [1–5].

This scheme still does not require deep learning to function. However, deep learning can allow it to operate with a lower sampling rate (that is, a lower ratio of speckle patterns to pixels in the image) [2, 6]. It also enables the use of nothing but the bucket detector values as the input, given a standardized (but still unknown) speckle pattern set [6]. If the speckle patterns can be unknown, that would improve the use of this technique in which there is strong random noise that does not vary significantly with time.

One important discovery about CGI is that the set of speckle patterns is very influential on its efficacy. Pink speckle patterns, which are ones that have lower-frequency correlations, have been used extremely effectively in CGI, even at very low sampling rates. In fact, appreciable results have been retrieved with sampling rates as low as 0.8% [2].

CGI has certain advantages over conventional imaging. For example, it is robust to noise, meaning that it can still reconstruct images in the presence of interference [2, 3]. In fact, noisy environments can be simulated and trained, and if the noise matches the sorts of scattering that

would be encountered in real detections, it can improve the quality of the CGI images [4]. However, this is not necessary, and images have been retrieved with networks trained without noise when operating in situations with noise [6]. Additionally, if it is afforded more computing power, it even allows for 3-dimensional sectioning [1]. One other advantage to the CGI setup is that the neural network can be trained purely on simulation data, meaning that lengthy experiments are unnecessary to prepare the network [4, 7].

1.2 Computer Science Background

Computational ghost imaging in this technique requires intensive computing resources and a deep learning network.

1.2.1 Deep Learning Networks

Deep learning is a type of machine learning characterized by having many layers. The computer undergoes training over a large set of example inputs and outputs, and it adjusts mathematical parameters so that its output predictions better match the correct outputs. The most common types of neural networks have been the recurrent or convolutional neural networks [8]. These use an encoder and a decoder, the generation of hidden states, and some attention mechanisms to learn the correlations between the positions of certain symbols. They are well-designed to interpret images. However, their design restricts their use in parallel computing, which limits the speed at which they can execute their computations [8].

1.2.2 Transformer Network

A newer deep neural network (DNN) that has been developed is the Transformer network. This style of deep learning relies almost exclusively on attention mechanisms, which are often minor parts of recurrent networks [8]. Attention draws global dependencies between input and output, and it allows for parallel processing [8]. The Transformer is well-suited for such tasks as language translation.

In a Transformer network, the encoder is composed of a set of identical layers, each with two sublayers: a multi-headed self-attention sublayer and a position-wise feed-forward network [8]. The decoder is similar, with an additional sublayer containing multi-headed attention

directed toward the encoder output. Positions that would be inaccessible during a true image-gathering experiment are masked from the decoder so that it cannot effectively cheat by looking ahead [8].

Mathematically, a Transformer network performs repeated steps. When it performs an attention computation, it maps a query and key-value pairs to an output, while all 4 data types are expressed as vectors [8]. The output is a weighted sum of the values, where the weights are determined by a compatibility function of the query and key [8].

Self-attention, which is the primary mechanism of the Transformer network, has advantages over other methods of deep learning. It allows long-range relationships to be considered by the network, and it allows for parallel computing with the Transformer network [8].

1.3 Ghost Translation

Recently, Ren *et al.* developed an application for the Transformer network to CGI at Texas A&M University [7]. Convolutional neural networks had been the primary tool for CGI [7], but this new network shows substantial promise. In their experiment, Ren *et al.* illuminated an object using K speckle patterns and collected corresponding intensities from the bucket detector. The patterns were made with a digital micromirror device (DMD). The Transformer network was then used, and it was able to learn to reconstruct binary images (where each pixel is a 0 or 1) based on training, even when there was substantial extraneous light on the bucket detector [7].

The beauty of this technique is that, given a known set of 2D speckle patterns, it is able to translate a 1D series of intensities into a set of 2D images [7]. A network can be trained on these speckle patterns and fed their results without even knowing the patterns themselves. Rather, the only inputs to the network are the bucket detector intensities and the "true" object images to train the model. This means that, with a trained network, an object can be imaged given nothing more than the DMD and a simplistic bucket detector [7]

The purpose of this thesis is to improve upon the design of the Transformer-based CGI, demonstrating how the same mechanisms may be used in grayscale images. While the ghost translation regime performs well with binary images, the contexts in which that is useful are exceedingly

limited. Grayscale images are much more widely applicable and more closely resemble images that are made by conventional means. Therefore, extending this regime to grayscale images will open the door to many more applications. With the speed brought by parallelization, CGI could be a viable form of imaging in laboratory contexts. This is particularly true given the way in which it is tied to speckle patterns, which are being used to accomplish increasingly difficult feats, such as defeating the diffraction limit [9]. Once grayscale images are achievable in the ghost translation scheme, full-color RGB images ought to be a simple change, and CGI will be applicable to many more contexts. Perhaps, one day, it will be a commonplace technique to generating 3-dimensional images of things smaller than the diffraction limit.

2. METHODS

2.1 Commenting and Documentation

Computational ghost imaging falls between the disciplines of physics and computer science. As such, there is some discrepancies regarding how experts in either group approach this problem. The Transformer CGI code was generally lacking in comments, which are important in computer science. Enough comments were therefore added such that future physicists working on the code could comprehend it. Further, since the code uses the PyTorch package in Python, the logging package was used and implemented to track the code's behavior. This made the code much easier to understand and to change appropriately.

Additionally, import paths were altered to be more clear and less prone to conflicting definitions.

2.2 Identifying Alterations

The Transformer network was, from previous research, designed to handle binary images, saved in the form of NumPy arrays in .npy files. These images are 32×32 pixels, and each pixel is either a 0 or a 1. The images are taken from the Quick Draw smiley face dataset [7]. For an example, see Fig. A.1. Grayscale images can be saved as NumPy arrays where the values range from 0 to 255, so the images were made to have a value between 150 and 255 in place of the 1's. One of the results is shown in Fig. A.2. Compatibility was then tested using the Texas A&M University High-Performance Research Computing (HPRC) clusters, and extensive log files were generated. The code was then altered to remove runtime errors.

The precise method of the previous version of the network was to convert the binary images into a list of pixel indices. PyTorch uses objects called Tensors that function similarly to NumPy arrays. For a batch of ten 32×32 pixel images, rather than using a $10 \times 32 \times 32$ Tensor, the network first converted it into a $10 \times x_{nz}$ Tensor, where x_{nz} is the greatest number of pixels with a value of 1 from a single image. This could be separated into a one-dimensional Tensor of length x_{nz} for each of the 10 images in a batch. Each of these one-dimensional Tensors contained the indices of pixels

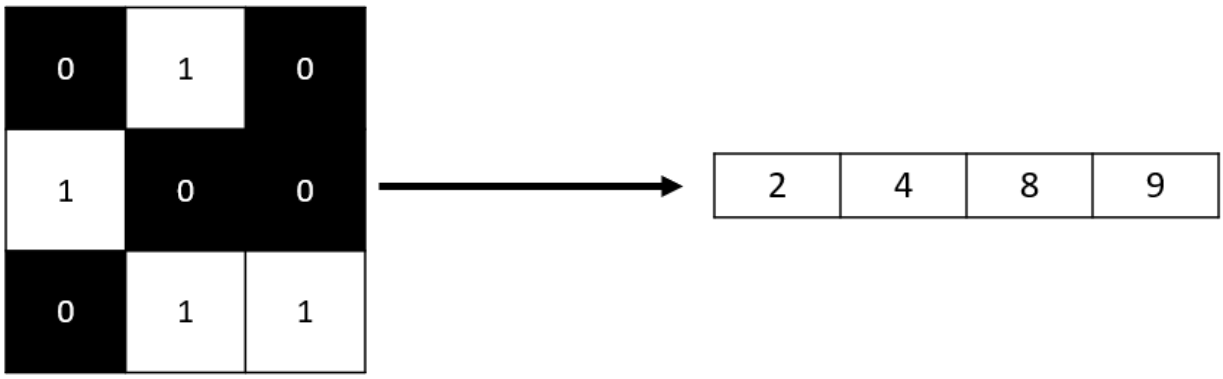

Figure 3: Previous method of image processing for binary images, applied to an example 3×3 image. The index of each pixel with a value of 1 is saved to a 1D Tensor.

with value 1 in a given image, as shown in Fig. 3. This technique was very resource-efficient, as it completely ignores all of the pixels with a value of 0 and makes the vocabulary size for the network only 1024 (the total pixel number in an image).

However, this approach left no room for values other than 1 in the active pixels, which are necessary to denote grayscale values. I conceived two potential approaches to implementing grayscale images. In the first, I would convert the Tensors to $10 \times x_{nz} \times 2$ Tensors, making an additional dimension of length 2. In this new dimension, one element would denote the pixel number, as before, and the other would denote the corresponding grayscale value. Just as before, these Tensors can be separated into ten individual Tensors, but these would have an additional dimension, as shown in Fig. 4. This would permit the network to use the same technique as before, finding associations between active pixels and associations between their values, while keeping the values closely tied to their pixel. In the second method, I would abandon the method of converting the images to lists of active pixels. Instead, I would employ the more straightforward method of inputting the entire images into the network. This would rely on the innate positional encoding designed into the network to ensure that it considers the position of pixels [8, 10, 11].

2.3 Method 1: Adding a New Dimension

In the first method, all references to the images needed a new dimension of length 2. This was accomplished through a technique as shown in Algorithm 1. In this technique, the .unsqueeze()

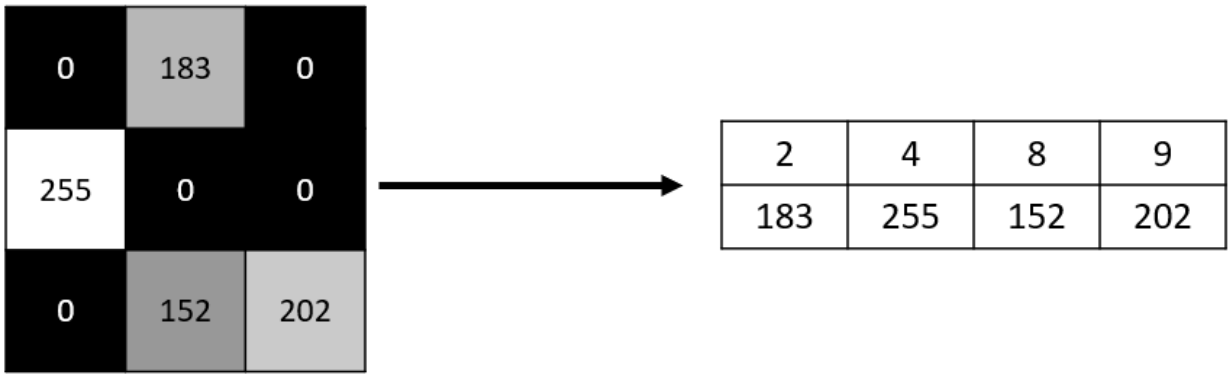

Figure 4: Method 1 for grayscale images applied to an example 3×3 image. Both the pixel index and the value of each nonzero pixel is stored in a 2D Tensor.

function was applied to a Tensor, introducing a new dimension of length 1. Then, that dimension was given a length of 2 through the function .expand(). However, this method causes both dimensions to refer to the same memory addresses. In other words, the second layer was only a shallow copy of the first. To allow these elements of the new dimension to be manipulated independently, the Tensor was made to be a clone of itself through the .clone() function.

Algorithm 1 Introducing a New Dimension

I $\leftarrow$ unsqeezed I with additional dimension
I $\leftarrow$ I with new dimension expanded to length 2
I $\leftarrow$ clone of I $\triangleright$ this creates a new object in a different memory location

This left the question of what to do with the input, namely the 10 one-dimensional Tensors that include the bucket detector photon counts. To allow algebraic compatibility with the image Tensors, a similar technique was applied, giving them an additional dimension with each element copied to the new dimension.

2.4 Method 2: Utilizing Innate Positional Encoding

In the second method, the image processing was simplified. Rather than being converted into a set of (for example) 10 one-dimensional Tensors of length x_{nz} containing pixel indices, the

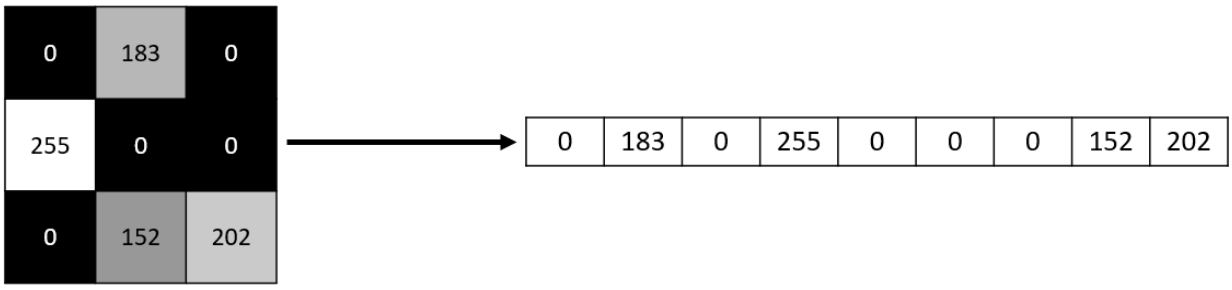

Figure 5: Method 2 for grayscale images applied to an example 3×3 image. The value of every pixel is stored in a 1D Tensor.

images were converted into 10 one-dimensional Tensors of length $32 \times 32 = 1024$, where each element was the value of a pixel, such as in Fig. 5. This conversion was accomplished with the .reshape() function, and it was done to make calculations more straightforward to track.

2.4.1 Positional Encoding

The Transformer model includes a PositionalEncoding layer designed to track the position of each token. This particular style of positional encoding uses vectors where each dimension corresponds to a sinusoid of a different frequency

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}), \tag{2}$$

where i denotes the dimension [8, 10]. This permits the network to transform any position into a vector sum.

2.4.2 Dimensionality

Because the previous network design involved frequently converting between ways of expressing the images, the code had to be almost entirely rewritten, except for the basic Transformer network design. The "correct" images that the network needs to use for training, frequently referred to as "trg" for "target," were straightforwardly reshaped to be 10×1024 Tensors. The bucket detector input ("src" for "source") was less straightforward but required fewer changes

from the previous version. The bucket detector input was shaped to be $10 \times x_{pat}$, where x_{pat} was the number of speckle patterns in the set.

One of the advantages to the ghost translation regime is that the bucket detector values can be simulated from known objects and speckle patterns rather than needing to be measured for every experiment [7]. Hence, that technique was adapted. In essence, the images were multiplied elementwise with each pattern, and all resulting elements were summed. Mathematically, this can be expressed as shown in Eq. 3, where B_i is the bucket signal corresponding to pattern P_i, and O is the object being imaged.

$$B_i = \sum_{x=0}^{31} \sum_{y=0}^{31} P_i(x, y) O(x, y) \tag{3}$$

The result was rescaled such that the lowest result for a given image was set to 0 intensity as a simple method to remove extraneous noise, and the highest result was set to $32 \cdot 32 \cdot 255 = 261120$. This allowed the bucket detector input to encompass every possible case from the image having pixels of only 0 intensity to pixels of only 255 (maximum) intensity. The pseudocode is given in Algorithm 2. Relatedly, the vocabulary size for "src" was set to be $32 \cdot 32 \cdot 256 = 262144$ so that each bucket intensity could be its own token, and the vocabulary size for "trg" was 256 to encompass each pixel intensity.

Algorithm 2 Simulating Bucket Detector Measurement

reshape patterns to $1 \times 51 \times 32 \times 32$ ▷ reshaping changes dimension lengths
reshape objectImages to $10 \times 1 \times 32 \times 32$
measuredImages $\leftarrow$ pattern $\times$ objectImages
bucket $\leftarrow$ sum of measuredImages over all pixels for each image-pattern combination
min $\leftarrow$ minimum element of bucket for each image
reshape min to be 10×1
bucket $\leftarrow$ bucket $-$ min
max $\leftarrow$ maximum element of bucket for each image
reshape max to be 10×1
bucket $\leftarrow$ bucket / max $\times$ 261120

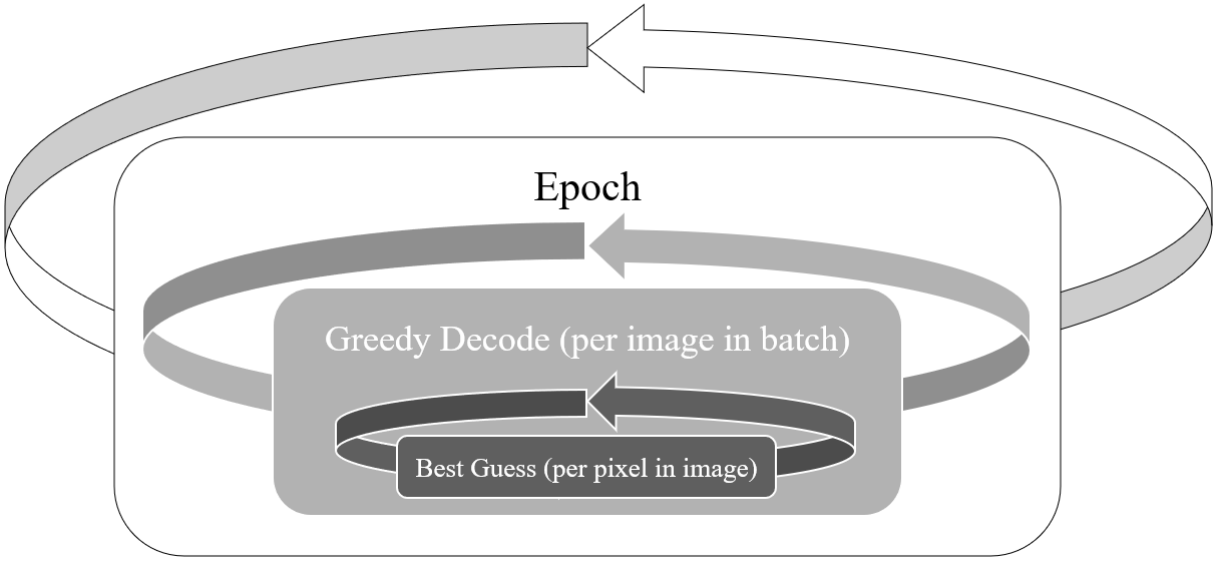

Figure 6: General outline of the Transformer network's repeating process. A large number of epochs are performed. In each epoch, each image in a batch is processed. For each image, the "best guess" is taken for each pixel.

2.4.3 General Architecture

The program for grayscale ghost translation adapts its general architecture from code used for binary ghost translation. Important objects, such as the model, are created first. The model is the object representing the Transformer network, and most other objects are components of the model. This model uses the Adam optimizer with a learning rate of 0.005. The model dimension is $d_{model} = 512$, there are $N = 6$ Encoder and Decoder layers, and there are $h = 8$ attention layers. The dropout rate is 0.1, and PositionwiseFeedForward layers have a dimension of 2048. Once these parameters and the model are made, the program begins a repetitive loop, as outlined in Fig. 6.

2.4.3.1 Epochs

The program contains an object that is its current "best guess" at re-creating the target images. This is updated many times, and each time that it is updated is called an "epoch." All other tasks are done within an epoch so that each layer of the Transformer network is run once per epoch. For this experiment, the network was run for 100 epochs and again for 433 epochs.

2.4.3.2 Greedy Decoding

In the interest of time efficiency, this network employs a form of greedy decoding. The version of greedy decoding has been adapted from works by Rush *et al.* and Ren *et al.*. Functionally, this is the process of forming a single guess as to the output based on the input. If the guess is better than the previous guess, then it is saved. To determine which guess is "better," their absolute error is used as their loss value, and the guess with the lower loss is used. Regardless of which guess is kept, the process repeats many times with the Encoder and Decoder layers being applied to the guess.

More specifically, the program constructs a guess one element at a time. Between each element, it processes the model through a set of Decoder layers, a Linear layer, and a Softmax layer with the previous elements as additional inputs. The model provides a probability for each pixel value for a given pixel, and the one with the highest probability is selected and appended to the Tensor with previous elements. This is done repeatedly, once for each pixel in each image per epoch.

2.5 Testing

In order to test the code's compatibility, it was run by the HPRC clusters after each significant change, which would reveal most of the errors encountered. Both a console output file and a logging output file were used. A batch of 10 Quick Draw smiley faces, each of which was 32×32 pixels, was used. A set of 51 "pink" speckle patterns were also used because they allow low sampling rates while retaining desirable results [2]. This number of patterns is approximately 5% of the sampling ratio [7], where 100% would cover the entire range of spatial relationships. The network was trained with 100 epochs, meaning that it processed each image 100 times. Then, it was trained again with 433 epochs.

A trial-and-error methodology informed the development of the program. As changes were made, attempting to run the code revealed errors as a result of the changes. This was either because a mistake was made or because another part of the code needed to be adapted to be compatible with the grayscale regime. The general development process, therefore, was as follows:

1. Make a change.

2. Run the code through the HPRC.

3. If there is a runtime error, correct the source of the error message. If necessary, consult the logging file produced by the code and the PyTorch documentation.

4. If more information is needed, insert more logging statements and rerun. Otherwise, make the appropriate change.

5. Repeat.

After the code was developed to the point of running with full compatibility, then the accuracy of its results would be examined. First, the grayscale ghost translation program would be run in training mode so that the model can learn the best parameters. Then, it would be run on different samples in evaluation mode to test its ability to reproduce accurate images. The fidelity of the resulting images will be evaluated using mean square error (MSE) and signal-to-noise ratio (SNR), which are some of the same quantitative measurements used in the original ghost translation regime [7]. Then, the quantitative results will be compared to the results achieved by Ren *et al.* in the binary ghost translation experiment.

3. RESULTS

3.1 Results of Method 1: Adding a New Dimension

The method in which the Tensors were given an extra dimension was plagued with runtime errors. Each time a Tensor was given this extra dimension, it was found to be incompatible with a different Tensor with regard to an operation that required specific dimensions. Eventually, it was found that the PositionalEncoding layers were attempting to perform an operation that was incompatible with the new dimensions. While it was feasible to alter the Transformer network's inner workings, that was decidedly outside of the scope of this project, and it would vastly increase the complexity of the issue; rather, the network should be used in its general form, applied to this specific situation. For this reason, this method was not pursued further.

3.2 Results of Method 2: Utilizing Innate Positional Encoding

The method in which the images were left as a full set of pixels showed more promise. However, the loss computation had repeated runtime errors. In particular, the use of the cross-entropy loss was not trivial to adapt. The version of the code that existed upon the beginning of this project did not actually use cross-entropy loss, but simply subtracted the pixel values of the best guess from the corresponding ones in the true object image. The absolute value of the total difference was the computed loss. I adapted this method for grayscale images.

3.2.1 Compatibility

As far as the basic compatibility, the bucket detector intensities and the object images were successfully integrated into the network as 10×1024 dimension Tensors. The neural network produced images that were correctly in grayscale form (with pixel values in the range from 0 to 255), as shown in Fig. A.3 through Fig. A.22, located in the appendix.

3.2.2 Qualitative Results

The images made with 100 epochs look nothing like the images from which they were generated. Their background color, rather than being black, is a dark gray, and they consist of multiple horizontal lines. They each have a similar structure with little differentiation, showing

that the network is treating them differently, but that it is not properly trained to handle the data set. Without substantial changes, these cannot be used for useful imaging.

The same process was done with 433 epochs. The number 433 was chosen for convenience in this particular run, but it is otherwise not meaningful. The resulting images are quite different from the 100-epoch images. They are lighter in their overall shade (having a higher average grayscale value). Rather than distinct horizontal lines, they exhibit a more evenly dispersed "static" effect. They also have greater diversity when compared to the other set of images. However, to a human observer, there is still nothing about them that resembles any of the original images. This indicates that there is still something interfering with the network's learning ability.

3.2.3 Quantitative Error

Two simple metrics to determine the error present in an image are the mean square error (MSE) and the signal-to-noise ratio (SNR). The equations for these are given in Eq. 4 and Eq. 5, where O_i represents the i-th pixel of the input object image, and O'_i represents the i-th pixel of the reconstructed image [7].

$$\mathrm{MSE} = \frac{1}{N_{pixel}} \sum_{i=1}^{N_{pixel}} (O'_i - O_i)^2 \tag{4}$$

$$\mathrm{SNR} = 10 \log_{10} \left(\frac{\sum_{i=1}^{N_{pixel}} O'_i}{\sum_{i=1}^{N_{pixel}} |O'_i - O_i|} \right) \tag{5}$$

For the set of 10 smiley faces used in the training and evaluation of this network, the error values after 100 epochs are shown in Table 1. The MSE values are quite large; a value of 10000 would mean that, on average, the pixel values differed from their true values by 100, where the maximum difference is 256. The binary version of ghost translation produced images with a MSE less than 0.05 using 100 epochs [7]. However, the range was only from 0 to 1 because the pixels could not be greater than 1. So, if this value is mapped linearly to the range of 0 to 256^2 for the grayscale case, similar results would have an MSE of 3276.8. My MSE values range from 188% to 392% of this value, meaning that my results have much greater mean square error than the binary

ghost translation regime, even when the error is properly scaled.

Regarding SNR, all of the results in the 100-epoch simulation are negative. The signal-to-noise ratio is a decibel value showing the relative strengths of the signal and of noise. A high, positive value is generally desired. The negative values in this simulation indicate that the noise has greater strength than the signal, rendering the original image all but impossible to retrieve. Binary ghost translation yielded an SNR of approximately 10 [7], which is far better than the grayscale results.

Table 1: Error calculations for each smiley face for 100 epochs.

Smiley face	**Mean square error**	**Signal-to-noise ratio**
1	11698	-3.3914
2	10496	-3.1195
3	10760	-2.8400
4	6896	-1.5464
5	6156	-1.4088
6	8804	-2.4513
7	11062	-3.1131
8	12845	-3.7399
9	11200	-3.0199
10	9424	-2.5795

For the test with 433 epochs, the results are tabulated in Table 2. Notably, the MSE values are all substantially higher than those from 100 epochs. This is unexpected because the network is set to prioritize reducing the absolute error, which should also reduce the MSE over time. This is because the absolute error, given in Eq. 6, is contained in the formula for the MSE. This high MSE is reflected in the lighter shades of the images, given that the background for the true object images is purely black.

$$\mathrm{AE} = \sum_{i=1}^{N_{pixel}} |O_i' - O_i| \tag{6}$$

The SNR for the 433-epoch results is much improved, however, when compared to the 100-

epoch results. These values are positive, indicating that the signal strength is above the strength of the noise. This development means that the general shape and the pattern of the smiley faces is more directly relevant to the output, which is likely tied to their increased variability. The input has greater influence on the output when there are more epochs, and the initial conditions of the network have less. This indicates the possibility that, given sufficiently many epochs, the image could come to resemble the correct object image. However, with this many epochs, the computational cost has already become high. It is much higher, for instance, than the 100 epochs needed for binary ghost translation [7].

Table 2: Error calculations for each smiley face for 433 epochs.

Smiley face	**Mean square error**	**Signal-to-noise ratio**
1	17949	0.73682
2	17639	0.72078
3	17041	0.75475
4	20192	0.42283
5	22622	0.28301
6	18936	0.48383
7	19588	0.85076
8	16713	0.94485
9	20522	0.87253
10	18237	0.75953

4. DISCUSSION

This new implementation needs more improvements before it is useful. However, it has illuminated likely issues that must be resolved in future research.

4.1 Loss Calculations

In this iteration of the ghost translation scheme, much of the calculation was done using both PyTorch Tensors and NumPy arrays. The NumPy arrays were included because they are extraordinarily easy to save to and load from files. Additionally, the Matplotlib module allows for them to be straightforwardly displayed as images. However, they were also used to carry over between epochs to allow easy checkpoint saving. Most significantly, they were used for the absolute loss computation during each training epoch, which was a technique carried over from previous versions of ghost translation. This implementation had detrimental results. Generally, a neural network tracks Tensors and operations through a directed acyclic graph, which allows it to traverse backwards and adjust parameters based on the changes to loss [12]. However, converting a Tensor to a NumPy array removes it from this computation graph, severing the graph and preventing backward propagation. This undermines the network's ability to fine-tune its parameters, which is its fundamental method of learning.

In future versions of this method, the use of NumPy arrays should be totally replaced with PyTorch Tensors. During all of the epochs, the same image Tensor should be used so that PyTorch loss functions, such as cross-entropy loss, can be used properly. The exception to this would be processing the initial image files and saving the final images as NumPy arrays in .npy files. Even saving checkpoints during long training sessions is possible with Tensors through saving the model state, rendering NumPy arrays unnecessary during the middle of the process.

4.2 Scaling

As the number of epochs increased, the overall shade of the images became lighter. One issue that may have arisen is that the scaling for src was poorly done. It was altered to have a

relative scale for each object. For each object, the bucket intensity for the pattern that was the lowest was set to be 0 to remove noise, and the bucket intensity for the pattern that was the highest was set to be $32 \cdot 32 \cdot 255 = 261120$. However, this assumes that there is a pattern that results in a completely black image and a pattern that results in a completely white image, which is not generally true. This technique was copied from previous research and might have been successful with binary images, but it may be poor practice that causes problems for grayscale images.

4.3 Dataset

Another source of error in this scheme is the limited nature of the training and testing sets. Because achieving mere grayscale compatibility was the primary focus of this experiment, a very small dataset was used. Namely, the set of 10 Quick Draw smiley faces shown in the inputs of Fig. A.3 through Fig. A.12 constituted the entire training set. However, the training set for a neural network is ideally much larger to give the network broader context to learn; 10000 handwritten digit images from the MNIST database have been used for computational ghost imaging [2]. One other practice that should be avoided in the future is using the same images in the training set as in the testing set. If the sets are different, then the ability of the neural network to work in general cases is properly tested. The MNIST dataset includes both a training set and a separate testing set. Therefore, in future work, the MNIST datasets or a similarly large pair of datasets should be utilized.

4.4 Epochs

The number of epochs also contributes to the accuracy of the results. Generally, the more epochs the computer is allowed to run, the more precise its convergence on the result will be. However, given the high degree of variability of the novel cases to which it can be applied, having more than a certain threshold of precision is usually not helpful. Because the binary ghost translation scheme required 100 epochs to achieve imaging with significant fidelity, it can be expected that the grayscale regime will require a greater number of epochs. This is primarily because this scheme requires that every pixel be represented, unless there is an additional optimization or batching added later. Additionally, the vocabulary that the network must learn only shrinks from the number of

pixels (for the pixel indices) to 256 (for the grayscale values). In this case, this is only a factor of 4. However, the dependency on the number of input values is likely more significant, so the runtime will be increased overall. It is likely that, given an improved version of this scheme, it will still require greater than 100 epochs to achieve satisfactory results.

4.5 Parallelization

One of the advantages of the Transformer network is its ability to perform many of its actions in parallel [8], saving on computing time. However, this potential was not fully realized in this scheme. With the use of NumPy arrays, some aspects of the Transformer network meant for this purpose, such as parallel loss computing [10], were not used. As part of transitioning to strictly Tensors, future work ought to use the full scope of parallelization available to the Transformer network to make the computation less time-intensive.

5. CONCLUSION

In the course of this research, methods of adapting the ghost translation regime to grayscale images were evaluated, and one was implemented with partial success. The methodology used for binary images was very efficient but entirely unsuited to grayscale images. While at first, expanding it with an additional dimension appears to be the most straightforward alteration, runtime errors suggest that it would require the inner workings of the Transformer network to be altered. A more successful approach involves converting the images into one-dimensional Tensors containing every pixel.

With this approach, the network was trained to output grayscale images given grayscale inputs. However, these images in no way resembled the desired images and had very large error values. Some component of the changes that were made had disrupted the ability of the neural network. This is likely due to the use of NumPy arrays in calculations, rather than exclusively PyTorch Tensors, and the use of an absolute loss calculation rather than the loss functions designed for PyTorch. Future research in this topic can implement these changes, use large datasets, and use parallelization to improve this method, potentially recreating the success that has been seen with binary images. While the implementation in this present work is not yet ready for laboratory application, it demonstrates several steps toward achieving grayscale ghost translation and identifies additional promising changes.

REFERENCES

[1] J. H. Shapiro, Phys. Rev. A **78**, 061802 (2008).

[2] H. Song, X. Nie, H. Su, H. Chen, Y. Zhou, X. Zhao, T. Peng, and M. O. Scully, Optics Communications **520**, (2022).

[3] Q. Gao, Y. Li, Y. Xia, and D. Duan, Turbulence-free computational ghost imaging (2022).

[4] Z. Gao, X. Cheng, K. Chen, A. Wang, Y. Hu, S. Zhang, and Q. Hao, IEEE Photonics Journal **12**, 1 (2020).

[5] X. Nie, F. Yang, X. Liu, X. Zhao, R. Nessler, T. Peng, M. S. Zubairy, and M. O. Scully, Phys. Rev. A **104**, 013513 (2021).

[6] F. Wang, H. Wang, H. Wang, G. Li, and G. Situ, Opt. Express **27**, 25560 (2019).

[7] W. Ren, X. Nie, T. Peng, and M. O. Scully, Optics Express **30** (2022).

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need (2017).

[9] N. Bender, M. Sun, H. Yilmaz, J. Bewersdorf, and H. Cao, Optica **8**, 122 (2021).

[10] A. M. Rush, V. Nguyen, and G. Klein, The annotated transformer (2018).

[11] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush, in *Proc. ACL* (2017).

[12] *A Gentle Introduction to torch.autograd* (2023).

APPENDIX: FIGURES

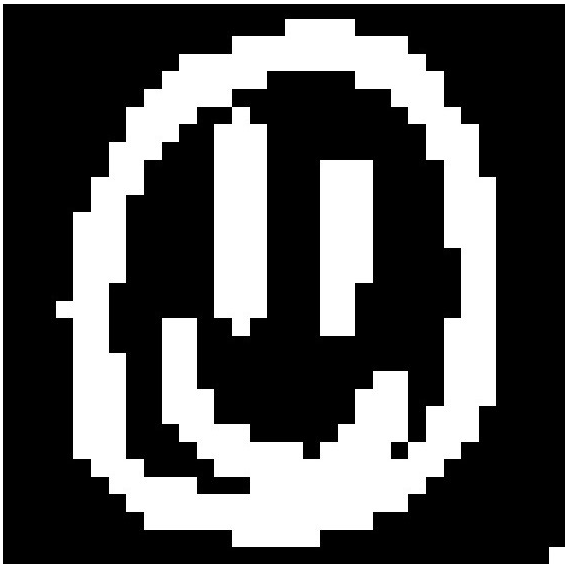

Figure A.1: An example of a binary training image.

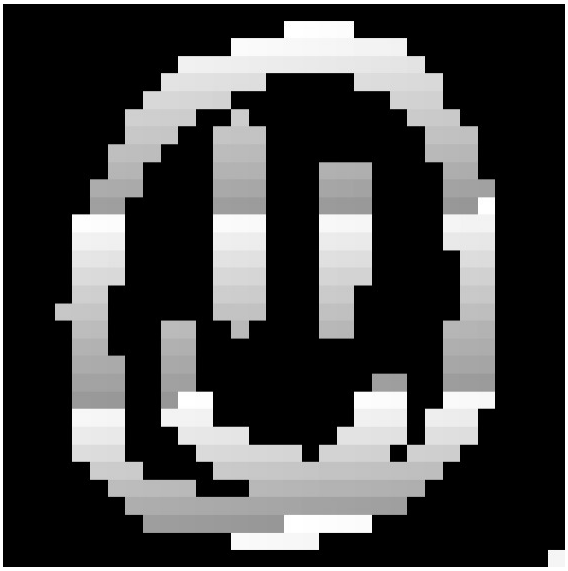

Figure A.2: An example of a grayscale training image, adapted from binary.

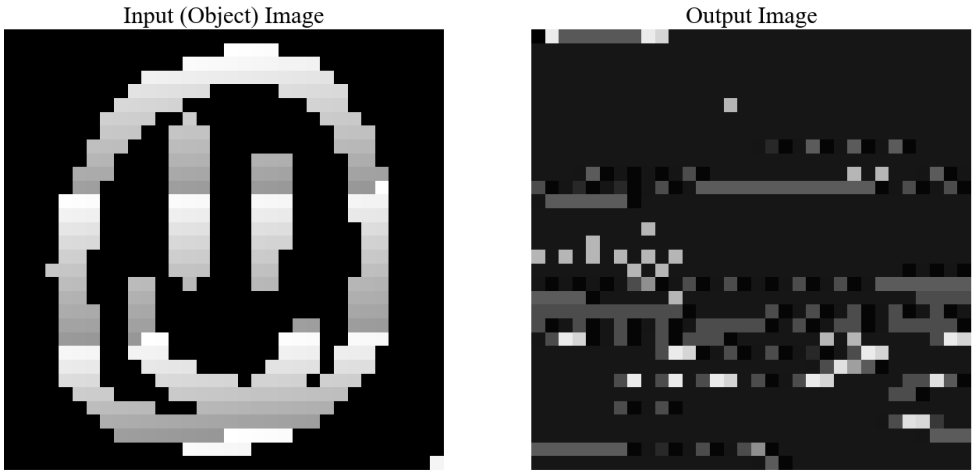

Figure A.3: Input smiley face 1 with its corresponding output after 100 epochs.

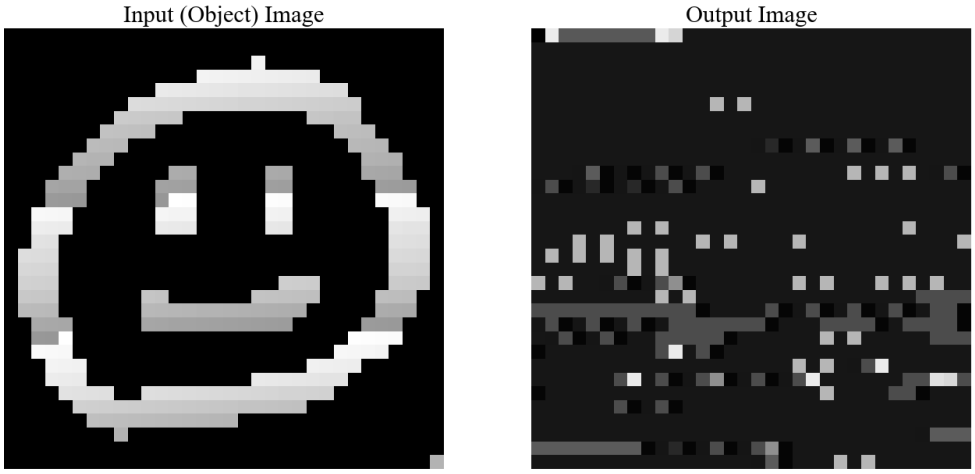

Figure A.4: Input smiley face 2 with its corresponding output after 100 epochs.

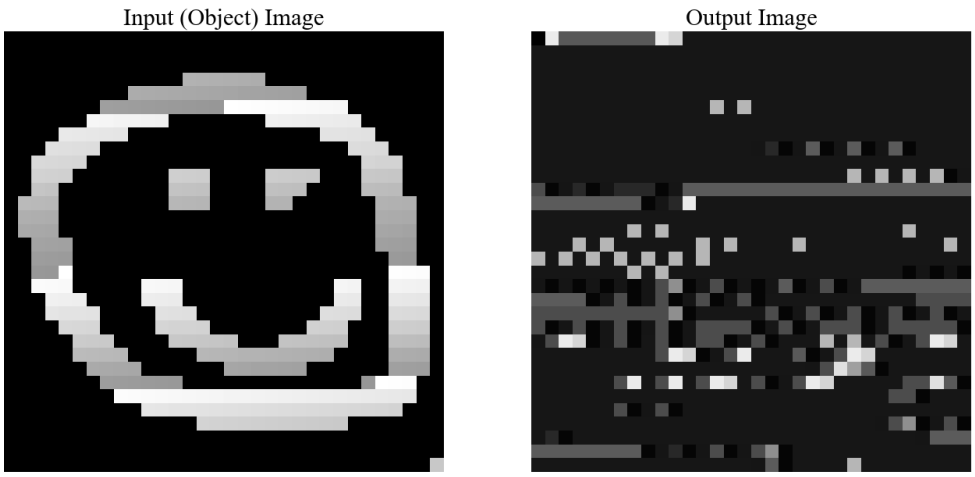

Figure A.5: Input smiley face 3 with its corresponding output after 100 epochs.

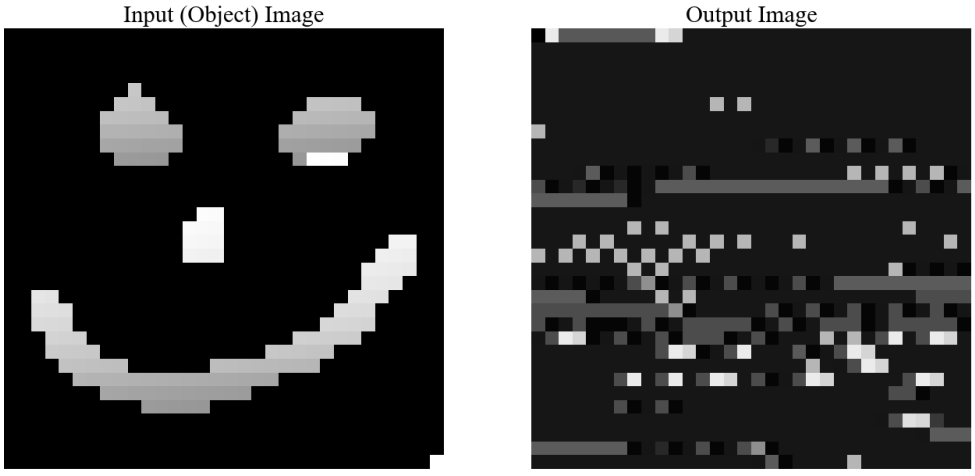

Figure A.6: Input smiley face 4 with its corresponding output after 100 epochs.

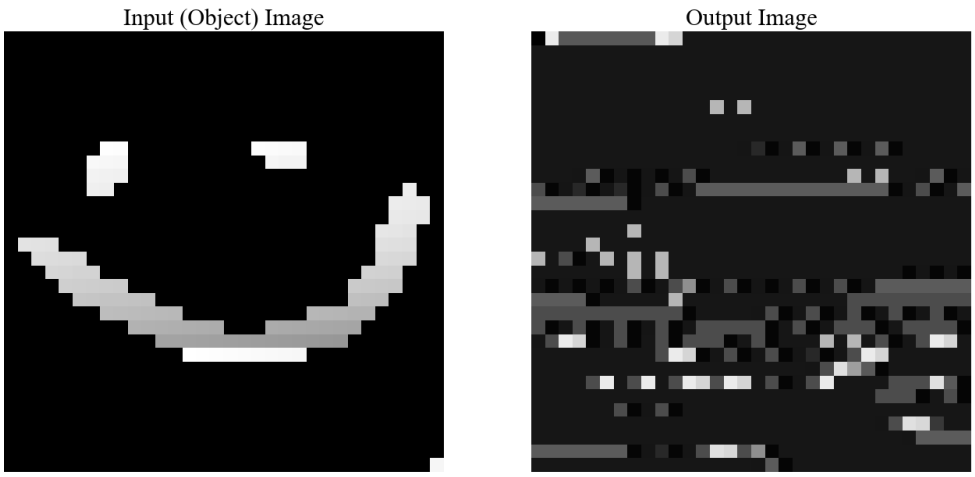

Figure A.7: Input smiley face 5 with its corresponding output after 100 epochs.

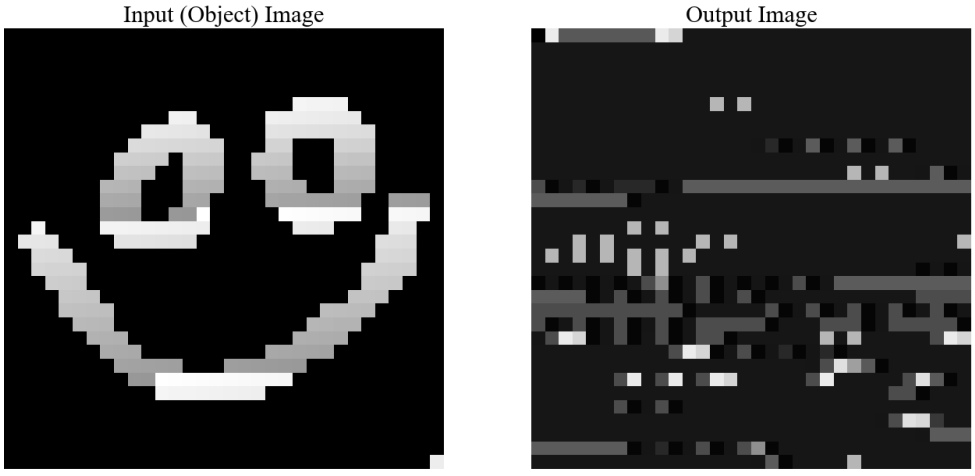

Figure A.8: Input smiley face 6 with its corresponding output after 100 epochs.

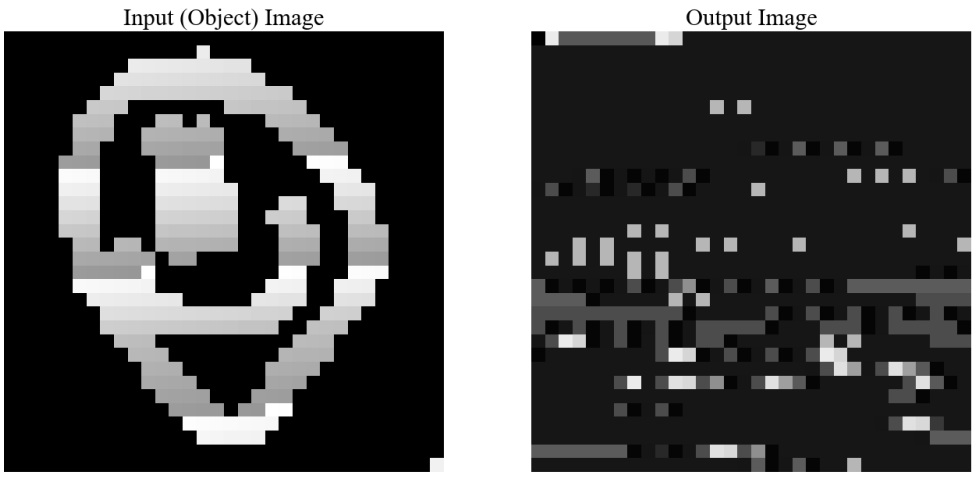

Figure A.9: Input smiley face 7 with its corresponding output after 100 epochs.

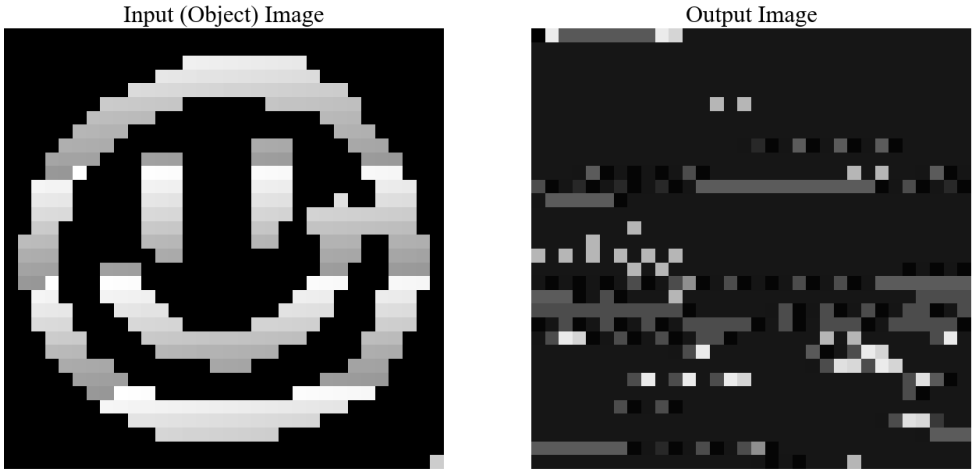

Figure A.10: Input smiley face 8 with its corresponding output after 100 epochs.

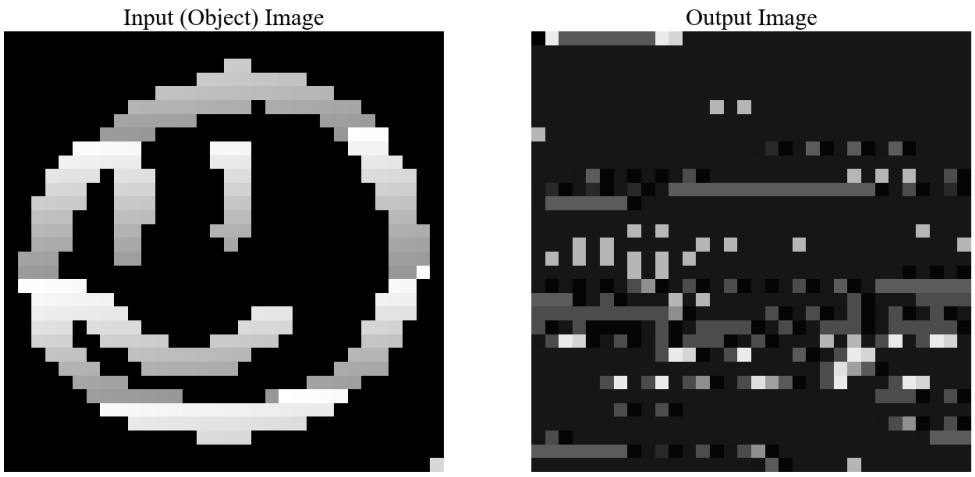

Figure A.11: Input smiley face 9 with its corresponding output after 100 epochs.

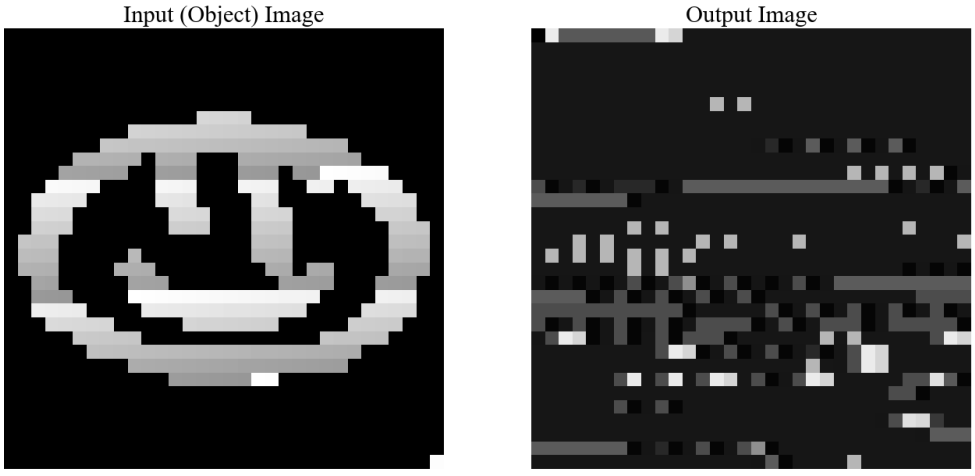

Figure A.12: Input smiley face 10 with its corresponding output after 100 epochs.

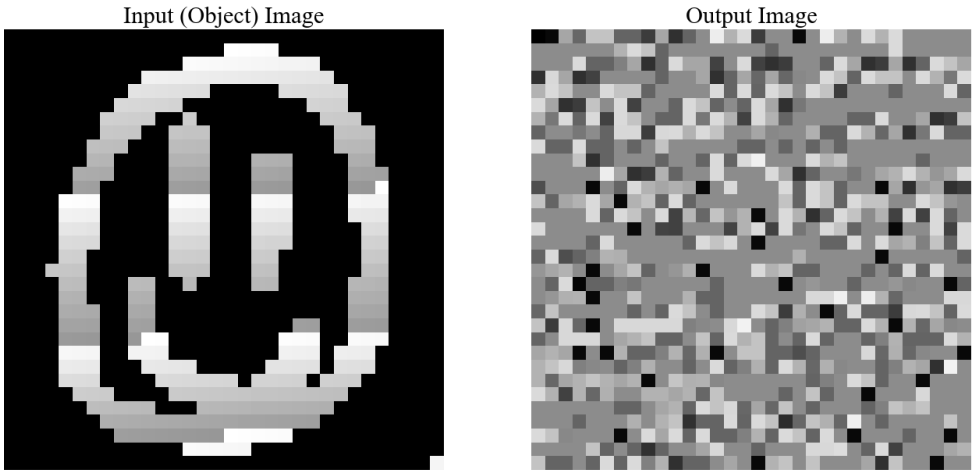

Figure A.13: Input smiley face 1 with its corresponding output after 433 epochs.

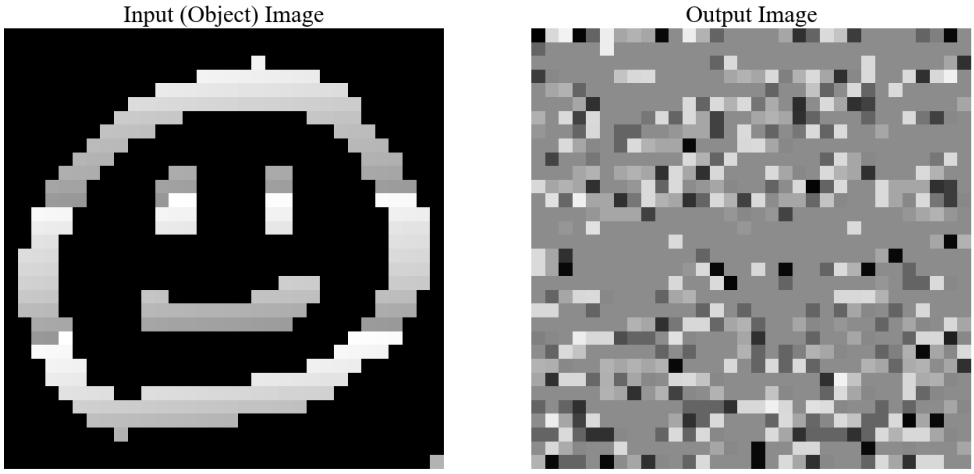

Figure A.14: Input smiley face 2 with its corresponding output after 433 epochs.

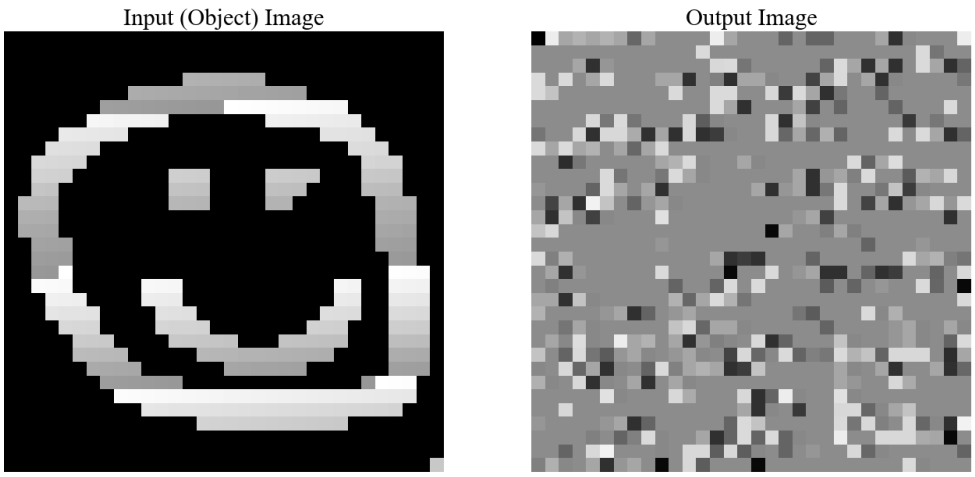

Figure A.15: Input smiley face 3 with its corresponding output after 433 epochs.

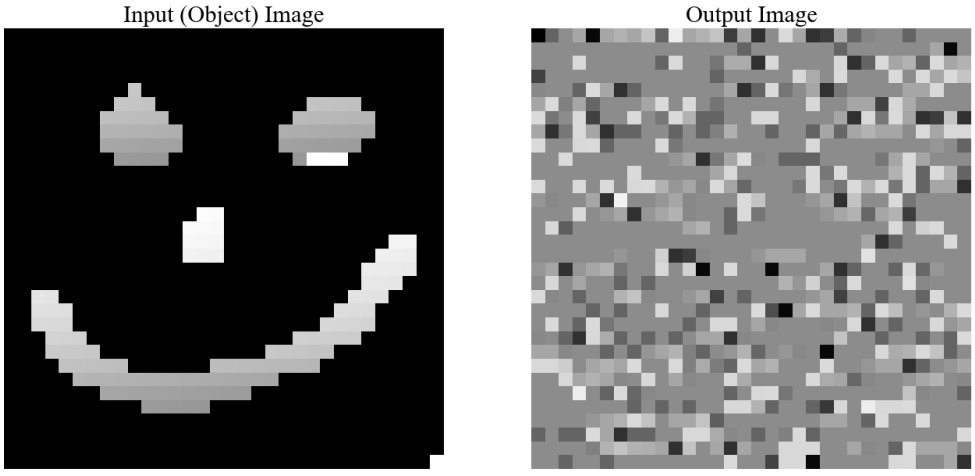

Figure A.16: Input smiley face 4 with its corresponding output after 433 epochs.

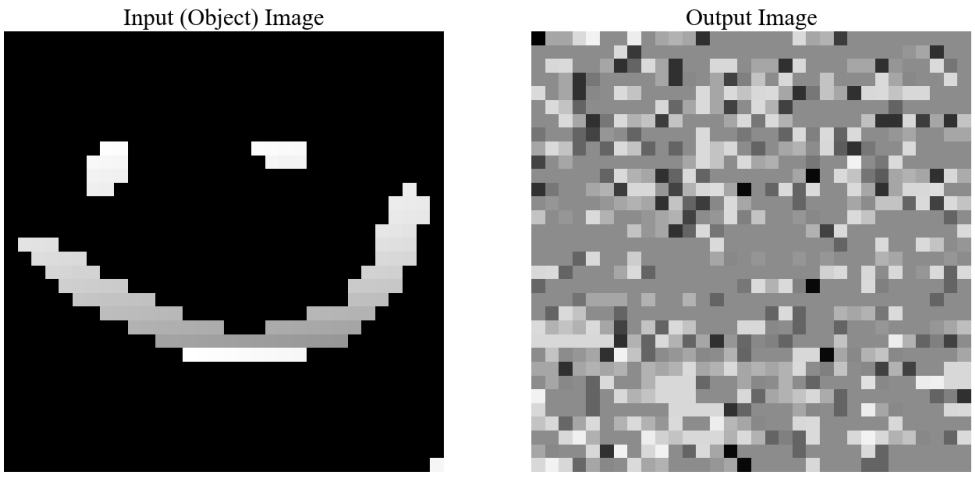

Figure A.17: Input smiley face 5 with its corresponding output after 433 epochs.

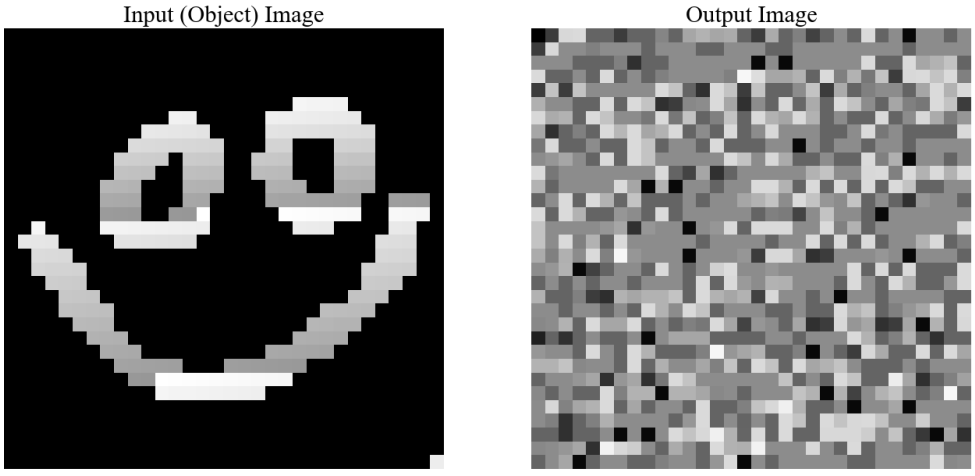

Figure A.18: Input smiley face 6 with its corresponding output after 433 epochs.

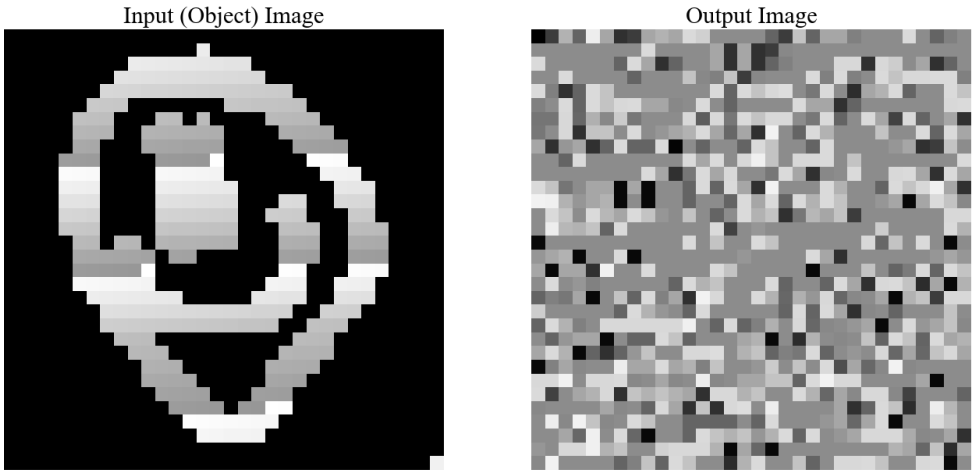

Figure A.19: Input smiley face 7 with its corresponding output after 433 epochs.

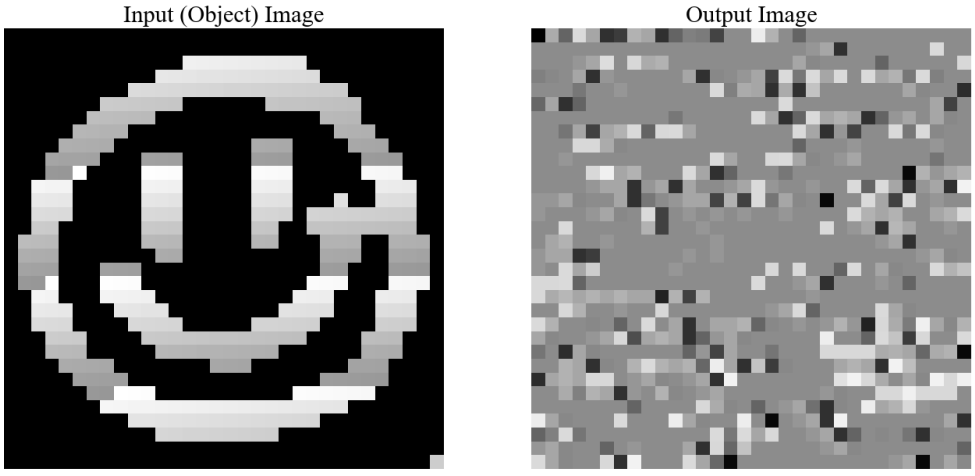

Figure A.20: Input smiley face 8 with its corresponding output after 433 epochs.

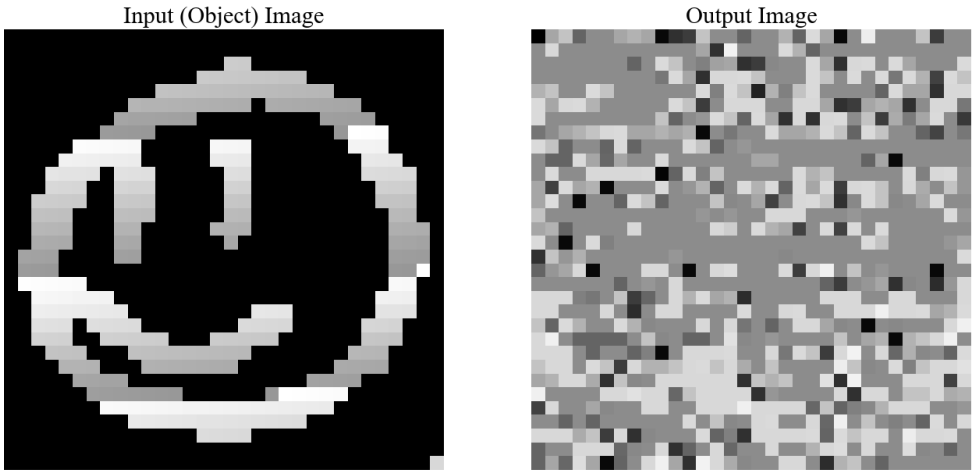

Figure A.21: Input smiley face 9 with its corresponding output after 433 epochs.

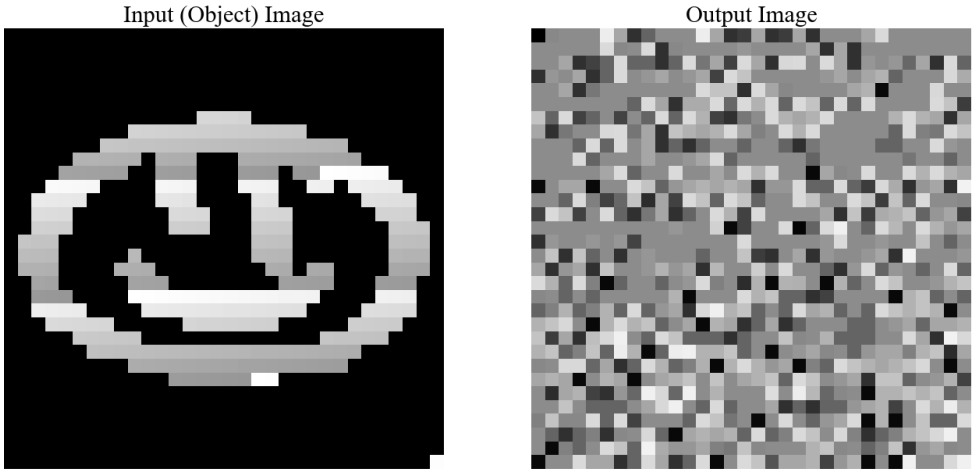

Figure A.22: Input smiley face 10 with its corresponding output after 433 epochs.