Assignment 2

Chico Demmenie (4451856)
Bart Holterman (4484010)
Bart Westhoff (3697932)

13 December 2024

1 Introduction

The goal of this assignment is to train a range of different image generation models for differing tasks. In task 1, the goal is to create an image network that can recreate images from the training set and also generate new ones from random vectors in the latent space. This will be accomplished using 2 types of networks, first Variational AutoEncoders (VAEs) use an encoder-decoder architecture that allows you to use the decoder to create new images. Second is the Generative Adversarial Network (GAN), which uses a discriminator as a "critic" and can therefore be an effective unsupervised learner.

2 Generative Models

In this task, the goal was to create a set of generative models with the ability to generate new images based on the training data and a set of random vectors within the latent space, the area that describes all possible images that the model could generate. The data was first trained on a Convolutional AutoEncoder (CAE), before the VAE and GAN examples, which allowed for the generation of new arbitrary images. The data used in this task includes a set of portrait paintings from the Dutch Rijksmuseum and a set of "Cryptopunk" images. The models were first trained on the Golden Age portraits but performed relatively poorly on these, so the Cryptopunk dataset was included as a less noisy alternative. The Hypothesis for this task is that the GAN will perform better due to its adversarial nature and how well adapted it is to unsupervised learning.

2.1 CAE, VAE and GAN

A Convolutional AutoEncoder (CAE) uses an encoder to learn efficient representations of images, otherwise known as latent variables. This is done using convolutional layers to extract important features from the image while reducing its dimensions gradually. This is then followed by a decoder, which does the opposite, taking the compressed latent variable

representation of the image and, using convolutional layers, gradually increases its size until another image is created [6].

Variational AutoEncoders (VAEs) work in much the same way, with an encoder and decoder that can sample from the training data, create a latent variable and then recreate the image from that variable. VAEs, however, use continuous, probabilistic representations, which allows us to generate new, randomly sampled vectors in the latent space, which will create new images that will look similar to the training data [4].

Lastly, Generative Adversarial Networks (GANs) consist of two competing networks, one that generates fake images from training data and a discriminator that tries to distinguish real-world data from the generator's fake data. This system goes through a loop where the generator tries to fool the discriminator, which in turn learns to distinguish real from fake and both networks improve over time in a feedback loop. This approach is why the hypothesis makes sense, as this approach produces much higher-quality images than a VAE, while being able to keep the diversity that makes VAEs better than CAEs [3].

2.2 Dataset

The initial dataset used was a set of portraits from the Dutch Golden Age in the Rijksmuseum. This didn't create very good images, however, likely due to its small size and variable nature, with differing face angles, expressions and clothing (Appendix Figure 10). It seems that pixel art images allowed for more stable and interesting outputs, in this vein, a Cryptopunks dataset was used, which consisted of around 10k images of pixel art faces [1].

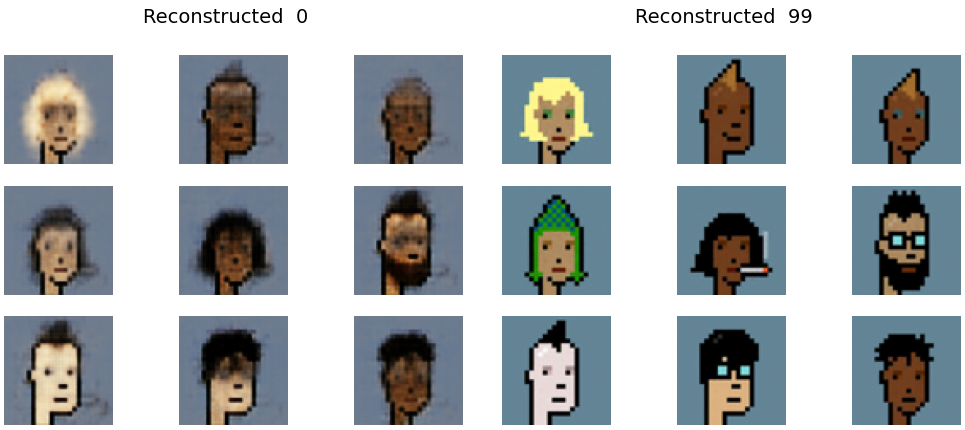
2.3 Preprocessing

The data was first downloaded and then preprocessed by scaling the images down from 336 pixel square images to 64 pixels square to fit the size of the models, which were created for a set of 64 pixel face images. The result of this can be seen in Figure 1 which shows a sample of images from the dataset in 64x64 pixel format.

Figure 1: A sample of 9 different Cryptopunks images from the dataset in 64x64 pixel format.

2.4 Training and Results

When training a CAE model, the weights are first initialised in the decoder. Then the training images are passed through the encoder to generate a vector, which is then turned back into an image in the decoder. Once an image is generated, a loss function, typically Mean Squared Error (MSE), finds the image loss, and a gradient is generated through backpropagation. An optimiser, like Adam, is then used to update the weights as is usual in model training. The first results of this method can be seen in Figure 2, which shows the output of the VAE model after it was trained on the original images in Figure 1.

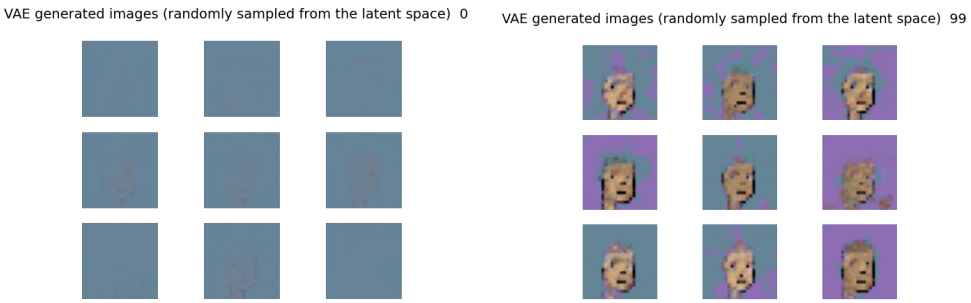


(a) CAE generated images at epoch 0 (b) CAE generated images at epoch 99

Figure 2: A comparison of training output images of the CAE at epoch 1 (a) and epoch 99 (b), showing the improvement over the training run as well as the stability and clarity of the CAE model.

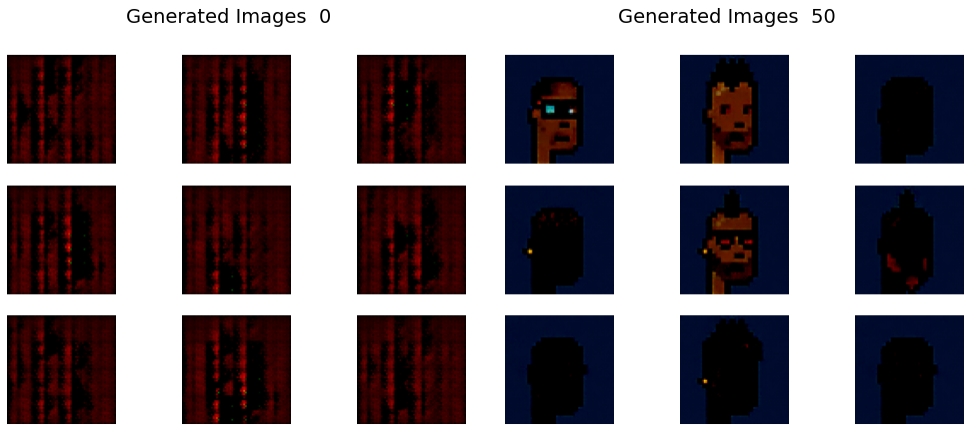
The VAE model is also trained by taking a training image and putting it through both the encoder and decoder, to create a reconstruction of the data. This differs from CAE as the output is then put through a negative Kullback-Leibler divergence formula [5] and an expected log-likelihood formula to find the loss [4]. The strength of this method is that it can generate new samples which resemble the training data but are still novel [4], as can be seen in Figure 3, which shows the output of the VAE model on the same original images.

Finally, the GAN model is trained similarly to the VAE, by setting randomised weights for both the discriminator and generator, defining loss functions for both and then going through the generate and discriminate cycle as described. This architecture can create much better images through this process, while maintaining the variability of the VAE. This process does also have the quirk that the output can change from one epoch to another as can be seen in Figure 4 and may be considered a pro or con depending on the application.

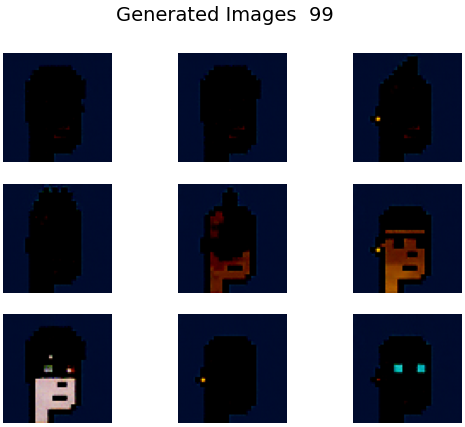


(a) CAE generated images at epoch 0 (b) CAE generated images at epoch 99

Figure 3: A comparison of training output images of the VAE at epoch 1 (a) and epoch 99 (b), showing that VAEs are good at creating novel images but not very good at being faithful to the originals.



(a) CAE generated images at epoch 0 (b) CAE generated images at epoch 50



(c) CAE generated images at epoch 99

Figure 4: A comparison of training output image of the GAN at epoch 1 (a), 50 (c) and 99 (b), showing that the GAN can vary its output significantly from one run to another.

2.5 Latent Variable Interpolation

Once the VAE and GAN models are trained on the data, it is possible to pass in a randomly generated vector or latent variable. This vector represents an image that is possible within the latent space and when passed into the model, creates a new, generated image. If we take two of these latent variables, we can even interpolate between them and generate images of the intermediary steps to show the transformation.

You can see how this didn't work so well in the VAE model, with the Rijksmuseum dataset in Figure 5 and on the GAN model in 6. You can, however, still see that there is a clear difference between the first and last image in the sequence and that the images in between look like the "between" stages.

Figure 5: VAE interpolation with Rijksmuseum dataset.

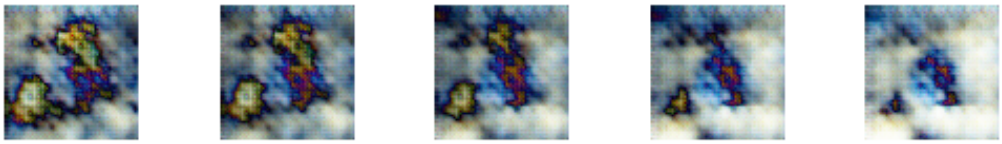


Figure 6: GAN interpolation with Rijksmuseum dataset.

After switching to the Cryptopunks dataset, the training had some modestly better results than before, so the gains in interpolation were expected to be largely in line with this. The VAE performed as expected and showed 2 slightly different faces and the interpolation between them, albeit with some artefacts (Figure 7). When the GAN was applied to this dataset, it outperformed its results with the Rijksmuseum dataset markedly (Figure 8), which is especially surprising after its questionable output during training. This outcome does still confirm the hypothesis, however, showing a clear improvement in the GAN model over the VAE, despite the questionable colour of its output.

Figure 7: VAE interpolation with Cryptopunks dataset.

Figure 8: GAN interpolation with Cryptopunks dataset.

3 Sequence Modelling with Recurrent Neural Networks

In this task, we will explore encoder-decoder Recurrent Neural Networks (RNNs) to solve a sequence-to-sequence problem based on the arithmetic operations of addition and subtraction of two-digit integers. These calculations are presented in text or image format. The network will process input queries like ‘’81+24’‘ or images representing the same, and produce outputs like ‘’105 ’‘ or the corresponding image, demonstrating the result of the operation. We will use the MNIST dataset for digit representations and develop text-to-text, text-to-image, and image-to-text mappings to achieve this. Using the encoder-decoder structure for sequence modelling, the network can generalise well to unseen examples by understanding the underlying arithmetic principles. The referenced splits are made after taking out a test set of 10% and treating the 90% of training data as 100%.

3.1 Dataset

The data set consists of two parts, the first set is a collection of handwritten digits called MNIST [2]. It also contains a smaller dataset of generated images of the arithmetic operators ’+’ and ’-’. These will form a ’query’ like 16-8 together with the answer, in this case, 8. The labels of the dataset consist of the string ’16-8’ for the query and a one-hot encoded vector for the answer, like the example in figure 9.

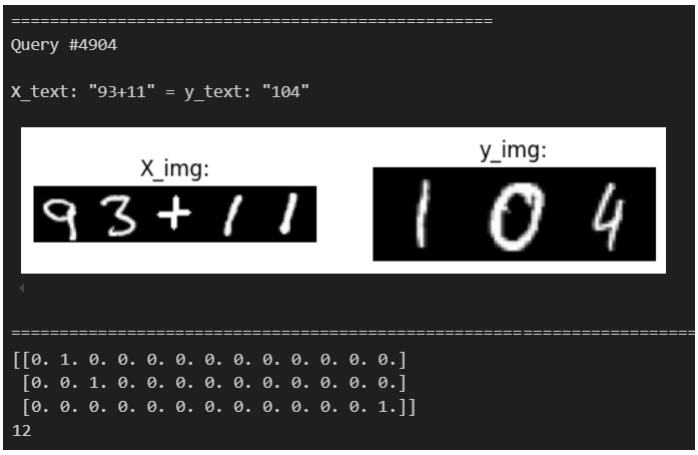


Figure 9: An example about the structure of the dataset, showing a query image and text interpretation and a number with the one-hot-encoded vector

3.2 Text-to-text model

The objective of this sub-task is to develop a text-to-text model that understands and generalises the principles of addition and subtraction. By creating a randomised dataset that we can easily understand, we prevent the model from memorizing all the given examples. By training on only a subset of the possible operations, the model's ability to generalise is tested. For example, the network achieved an accuracy of 58.78%. Table 1 presents benchmarks across various training segments, highlighting the trade-offs between the size of the training set and the model's generalization capabilities. This approach ensures that the network develops a foundational understanding of arithmetic operations, enabling it to handle unseen examples effectively, as shown by the best test accuracy.

Since the model predicts a sequence of characters and not an integer. We have to clean certain outputs, for example, removing the answer '-' since it is not a valid answer. We remove every sample in the prediction set which does not appear in the answer set. After cleaning we can see a distribution of the true and predicted set to see what the model 'thinks'. The distribution is shown in Figure 11. The results can be declared from the fact that the model is still training and the loss is still high. With more than 10 epochs or a smaller batch size, it could generalise much more. In figure 12 you can see that the predicted and actual values are much closer than they appear in figure 11.

3.3 Training a larger model

For the second part of the text-to-text model, we tried to change the number of encoding and decoding layers. The results in Table 2 show that the amount of layers in the model has some influence on its accuracy.

3.4 Image-to-text model

The next part is about feeding an image to a model where the output should be the answer to the equation, this is how humans see text, as written characters that represent an integer. The model is trained using the previously mentioned data split of 90/10, with the history plot in Figure 13 showing that the model converges slowly, requiring more epochs to gain similar levels of accuracy. After searching for the ideal number of encoders and decoders, table 3 shows that the accuracy lies at 56.06%.

4 Text-to-Image model

This last part focuses on the text-to-image model, which is evaluated on the MSE of the generated image as the performance is not measured easily. The MSE values shown in table 4 are really close together and thus inconclusive. A generated image can be seen in 14 where the plus and minus signs are very visible since these each appear in half of the datasets.

5 Conclusion

The first section of this report describes the workings of three different generative models and each of their advantages and disadvantages, as well as the two different datasets, one well-suited and the other less well-suited to this kind of task. The output of the models, especially the generation of new images and interpolation between them clearly showed that the GAN architecture was the best suited to this kind of task due to its ability to both generate novel images and stay relatively faithful to the original training data, proving that the hypothesis in the introduction was correct.

In the second section of the report, several methods were shown for training models on similar tasks with the common attribute that these models never really reach their full potential possibly due to them being capped at 10 epochs, because of computational power constraints. After evaluating the results it is clear that the models work much better when they have more encoder and decoder layers. It was not possible to test multiple encoder and decoder layers at the same time, which, according to the tables, could result in good evaluation scores.

References

[1] Wasiq Abdullah. Pixelated treasures: 10k cryptopunks, 2023.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

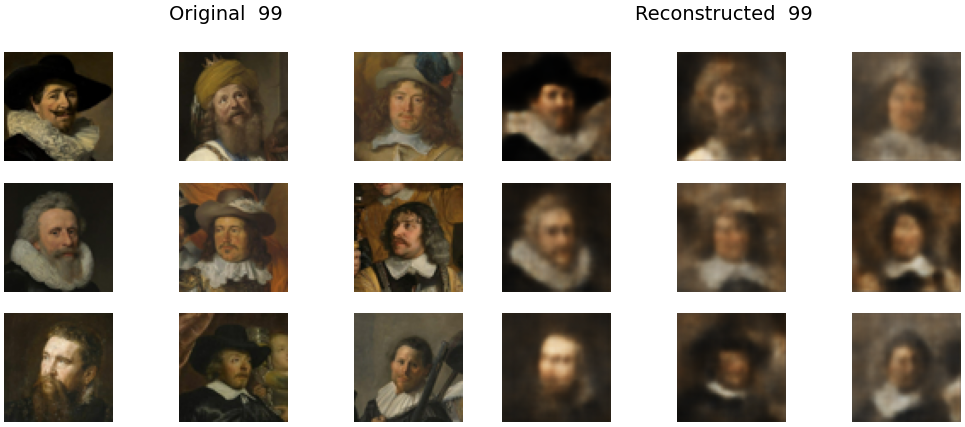
[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Proceedings of the International Conference on Neural Information Processing Systems*, 2672-2680, 2014.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[5] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[6] Emmanuel Pintelas, Ioannis Liveris, and Panagiotis Pintelas. A convolutional autoencoder topology for classification in high-dimensional noisy image datasets. *Sensors*, 21 (22), 2021.

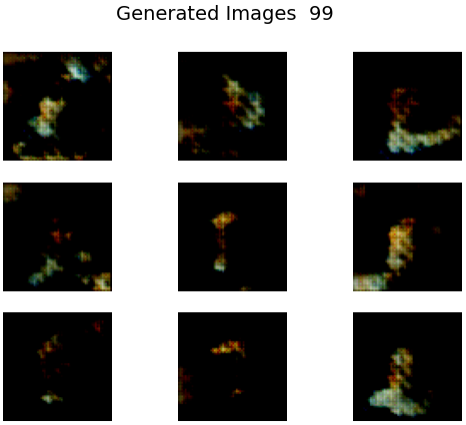
6 Appendix



(a) The original Rijksmuseum dataset.

(b) CAE performance on the Rijksmuseum dataset.

(c) VAE performance on the Rijksmuseum dataset.



(d) GAN performance on the Rijksmuseum dataset.

Figure 10: A comparison of training output images of the CAE (b), VAE (c) and GAN (d), to the original images (a) of the Rijksmuseum portraits dataset showing that this dataset was not well suited to image generation of this kind.

training size %	test accuracy	test loss
0.9	58.73	1.1486
0.8	55.78	1.2187
0.7	56.12	1.2283
0.6	54.7	1.2778
0.5	52.42	1.3224
0.4	48.8	1.4134
0.3	46.63	1.492
0.2	42.47	1.6475
0.1	40.05	1.8311

Table 1: The 5 configurations with the highest average score.

number of encoders %	number of decoders	test accuracy (%)
1	1	56.61
1	2	58.72
1	3	63.68
1	4	62.31
1	5	62.59
2	1	59.16
3	1	60.59
4	1	62.01
5	1	59.12

Table 2: The 5 configurations with the highest average score.

number of encoders %	number of decoders	test accuracy (%)
1	1	48.63
1	2	49.81
1	3	48.77
1	4	49.53
1	5	47.77
2	1	53.66
3	1	55.91
4	1	56.06
5	1	52.01

Table 3: The 5 configurations with the highest average score for the image to text model.

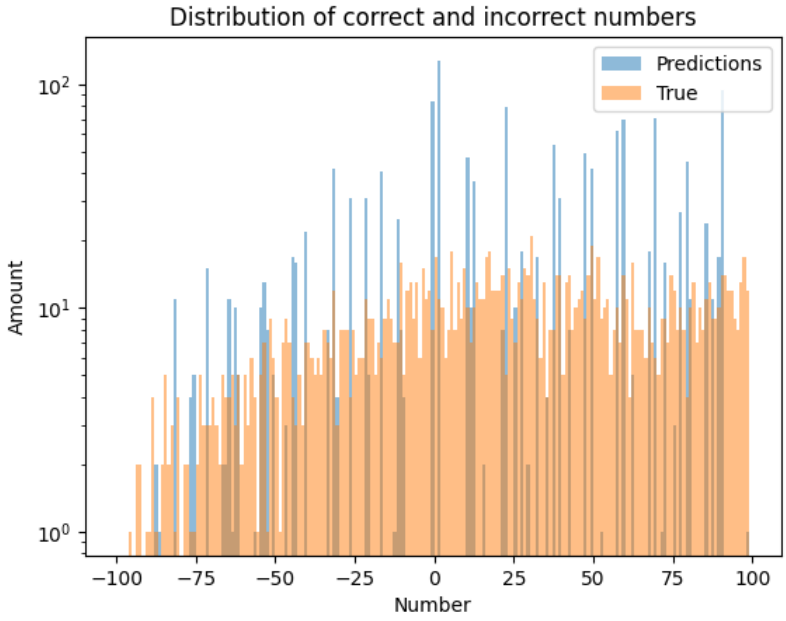


Figure 11: This figure shows a distribution of all values in the testset and what the models things the answers were.

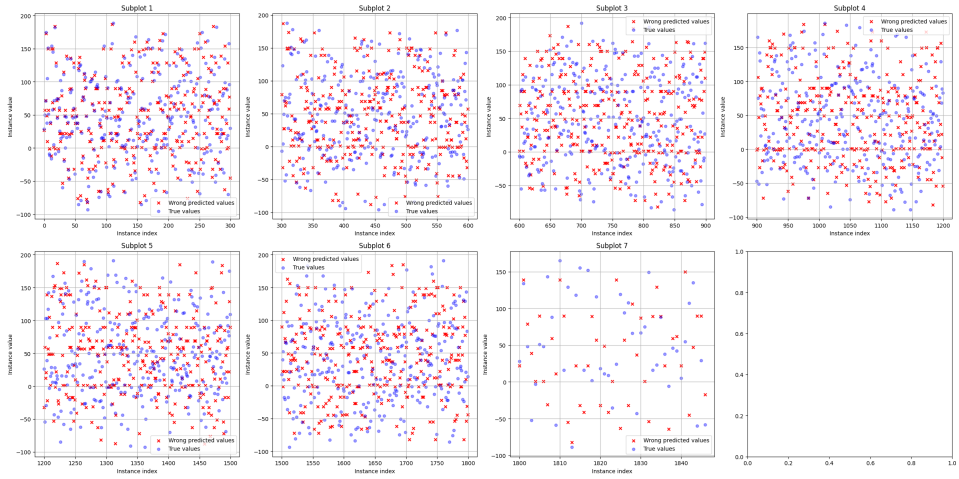


Figure 12: These subplots show how much the distance is between a predicted answer and the real answers.

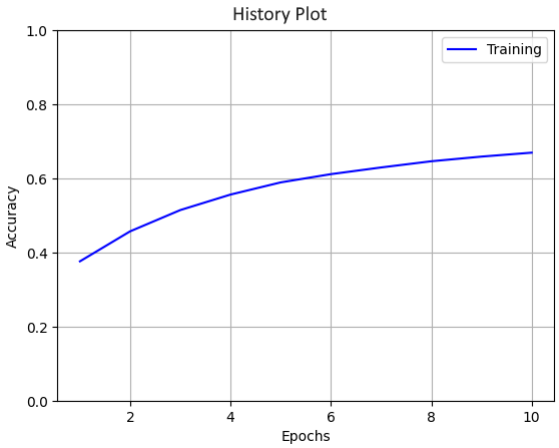


Figure 13: This graph shows the training accuracy of the model over 10 epochs.

number of encoders %	number of decoders	test MSE (%)
1	1	0.5809
1	2	0.5878
1	3	0.5877
1	4	0.5872
1	5	0.588
2	1	0.5871
3	1	0.5872
4	1	0.5876
5	1	0.5875

Table 4: The 5 configurations with the highest average score for the image to image model.

Figure 14: This graph shows the output of the image generator, where the plus sign is very visible.