Horse 2 Zebra

Lior Yaacov[1] and Daniel Manor[2]

[1]lioryaac@bgu.ac.il
[2]dmanor@post.bgu.ac.il

September 15, 2024

Abstract

In this project, we address the problem of unpaired image-to-image translation, specifically transforming horse images into zebra images. The core of the approach is a CycleGAN (1) architecture, which allows for image translation without paired datasets, leveraging the concept of cycle consistency to ensure meaningful transformations. To enhance local feature detail, we employ PatchGAN (2) as a discriminator, focusing on smaller patches of the image. The novelty of this work lies in the incorporation of a segmentation-based attention mechanism, which guides the generator to prioritize important regions of the image, ensuring more accurate and realistic translations. This attention mechanism not only improves translation quality but also helps avoid the unintended translation of irrelevant objects in the image, ensuring that only the horse undergoes the transformation. By focusing on key features, such as object boundaries and distinctive characteristics, the model preserves the content while enhancing the overall fidelity of the generated images.
Through a series of experiments, we observe promising results, with good translations between horse and zebra domains, demonstrating the potential of this approach.

1 Introduction

The task of translating horse images into zebra images presents a challenge due to the lack of paired datasets. In an ideal scenario, each horse image would have a corresponding zebra image, where the horse's posture, shape, and environment are exactly aligned with the zebra's. However, such paired data doesn't exist in the real world. There are no datasets where each horse image has a matching zebra version, with all key elements properly aligned.
This lack of paired data makes it difficult to directly train a model to learn the transformation between the two domains. Without an exact "before and after" image pair (horse and zebra), a traditional supervised learning approach is not feasible. Instead, we need a method that can learn how to map images from one domain to another without relying on corresponding pairs.
To address this challenge, we adopt CycleGAN as the basis of our framework. CycleGAN allows for the translation of horse images into zebra images without the need for paired data by using a cycle consistency loss, which ensures the structure of the horse is maintained during the translation. However, a vanilla CycleGAN can lead to unintended results, where not just the horse but other objects in the image are transformed.
For instance, when an image of Vladimir Putin riding a horse was input into the vanilla CycleGAN (see Figure 1), both the horse and Putin were turned into zebras. This scenario highlights a possible outcome where the model transforms unintended elements in the image. While the model can work well in other situations, this example shows that it doesn't always focus solely on the primary object.

Figure 1: Input image of Vladimir Putin riding a horse (left) and the output from a vanilla CycleGAN (right). In the output, both the horse and Putin have been translated into zebras, demonstrating a possible scenario where the model transforms unintended elements of the image.

To address this, we introduced segmentation-based layers to guide the network to focus on the object of interest (the horse), preventing other elements like Putin from being transformed. Additionally, segmentation-based attention enhances the accuracy of the object translation, ensuring a more focused and realistic transformation of the horse itself.

2 Dataset

The dataset used in this project is the Horse2Zebra dataset (1), which contains unpaired images of both horses and zebras. As shown in Figure 2(a), the dataset is relatively balanced, with a slightly higher number of zebra images (55% in the training set and 54% in the test set).

Additionally, we created a segmentation map dataset 2(c) for all training examples to accelerate the training process. While the model is designed to perform segmentation during inference with only an RGB image as input, we provided the segmentation maps during training to speed up the process. This required slight adjustments to the architecture, but it proved beneficial, reducing the training time by approximately 40%.

(a) (b) (c)

Figure 2: (a) Dataset distribution. (b) Horse2zebra dataset examples. (c) Segmentation examples.

3 Architectures

3.1 CycleGAN

CycleGAN is taking advantage of its ability to perform unpaired image-to-image translation. Unlike traditional GANs, which require paired data to learn a mapping from one domain to another, CycleGAN introduces **cycle consistency loss** to ensure that the transformation between domains preserves the structure and content of the original image. The model consists of two key components: **two generators** and **two discriminators**. The first generator, **G**, translates horse images (Domain X) into zebra images (Domain Y), while the second generator, **F**, performs the reverse translation from zebra back to horse. Meanwhile, the two discriminators, $\mathbf{D_X}$ and $\mathbf{D_Y}$, work to distinguish between real and generated images in each domain, ensuring that the generated images are as realistic as possible.
The **cycle consistency loss** is crucial for maintaining the integrity of the images during translation. For example, a horse image is first translated into a zebra image by **G**, and then the resulting zebra image is passed through **F** to be translated back into a horse. The loss is computed by comparing the original horse image to the reconstructed horse image, ensuring that the structural information is preserved during both transformations. A similar process is followed when translating from zebra to horse and back to zebra.
While CycleGAN generally performs well, there are cases where the model may struggle, such as when it unintentionally translates objects other than the horse or when the output images lose fine details.

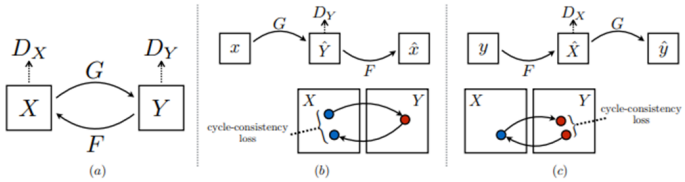


Figure 3: CycleGAN architecture: (a) illustrates the translation between domains X and Y using generators G and F, and discriminators D_X and D_Y. (b) and (c) depict the forward and backward cycle-consistency processes, ensuring the input images are reconstructed accurately in their respective domains.

3.2 PatchGAN

Another technique we adopted in our framework is the use of PatchGAN as the discriminator. PatchGAN operates differently from traditional GAN discriminators by evaluating the image at a local level. Instead of classifying the entire image as real or fake, PatchGAN divides the image into smaller patches and evaluates each patch individually. This approach allows the model to focus on finer details, such as textures and patterns, which are crucial for tasks like horse-to-zebra translation.
In our case, PatchGAN helps ensure that local features, like the zebra's stripes or the horse's fur, are accurately captured during the translation process. The output of PatchGAN is a matrix of probabilities, with each value corresponding to the likelihood that a given patch is real or generated. By encouraging the model to generate realistic textures at a local level, PatchGAN enhances the overall realism of the output images, ensuring that small but important details are not overlooked.

4 Method

In our work, we want to leverage segmentation maps for better image generation.

Using the segmentation map as-is, i.e., blend the output image with parts from the original one results in bad images, as the segmentation is not robust enough.

We propose to use segmentation to focus the model over relevant areas (i.e., horses and zebras) to generate more accurate image translations.

Our model, similar to a standard CycleGAN, features two generators and two discriminators. However, we have enhanced it by incorporating attention layers within the generators' residual

blocks and at the decoder. We have two generators: G_{AB} which generates zebras from horses, and G_{BA} which generates horses from zebras.
We also have two discriminators: D_A that should distinguish between real and fake horse images, and D_B which do the same for zebras.
In the following section we will describe how it is done step by step.

4.1 Segmentation

To generate the segmentation maps, we experimented with two models: DeepLab v3 (3) and Mask-RCNN (4).
Zebras segmentation proved to be particularly challenging, and DeepLab v3 struggled to achieve satisfactory results.
While Mask-RCNN successfully completed the segmentation task, its results were less accurate compared to DeepLab v3.
Therefore, we chose to use DeepLab v3 for the horse segmentations and Mask-RCNN for the zebras.
For training purposes, we created a segmentation dataset from the train data (horses and zebras), as noted above 2(c)

4.2 Generator Architecture

Our generator is composed of three key components: an encoder, a series of residual blocks, and a decoder (see Figure 4).

In the encoder, the input image is passed through multiple convolutional layers , each followed by instance normalization and ReLU activation. The encoder consists of three layers in total, producing an output tensor with dimensions (64, 64, 256).

Each residual block contains a convolutional layer, instance normalization, ReLU activation, followed by another convolutional layer and a second instance normalization.

The input is duplicated into a new variable to establish a residual connection. At the end of the block, the output is concatenated with the original input along the channel dimension, resulting in 512 channels. A dimension reduction block then reduces this back to 256 channels. Throughout the residual block, the output shape remains consistent with the input.

The number of residual blocks in the generator is a user-defined parameter. For our implementation, we have chosen to use 9 residual blocks.

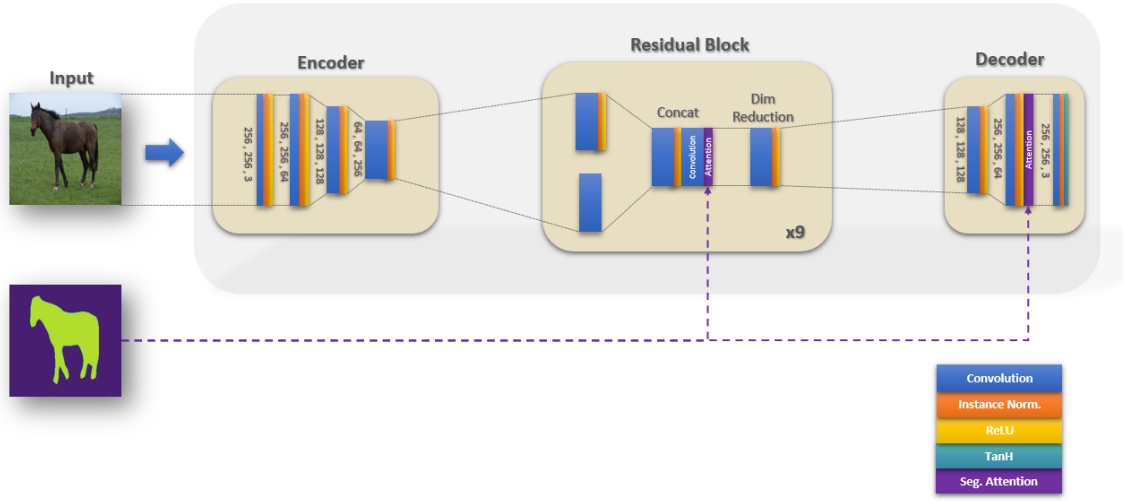


Figure 4: Generator Architecture

In the decoder, the input tensor is first passed through an upsampling layer, which increases its spatial dimensions. This is followed by a convolutional layer, and a ReLU activation function is applied. The processed tensor is then upsampled again and passed through another convolutional layer, followed by another ReLU activation. Finally, the tensor is passed through a third convolutional layer with a TanH activation. Each step progressively refines and upscales the tensor to reconstruct the final output.

We employ upsampling and convolution layers to mitigate the "checkerboard effect" often observed with transposed convolution layers

We incorporate the segmentation-guided attention layer both within the residual block and at the end of the decoder, as illustrated in Figure 4.
To align the shape of the input tensor (after processing) with that of the segmentation map, we apply interpolation to the segmentation map. The resized segmentation map is then fed together with the processed input into the attention module block, as detailed in the following section.

4.3 Attention Module

In the attention module, the input tensor is first concatenated with the segmentation map. This concatenated tensor then passes through a convolutional layer, followed by instance normalization and ReLU activation.

Subsequently, an attention map is generated using another convolutional layer, instance normalization, and a sigmoid activation function.

Finally, the attention map is computed by multiplying the original input tensor with the generated attention map and then adding the original input tensor as a residual.

The attention module block is illustrated in Figure 5.

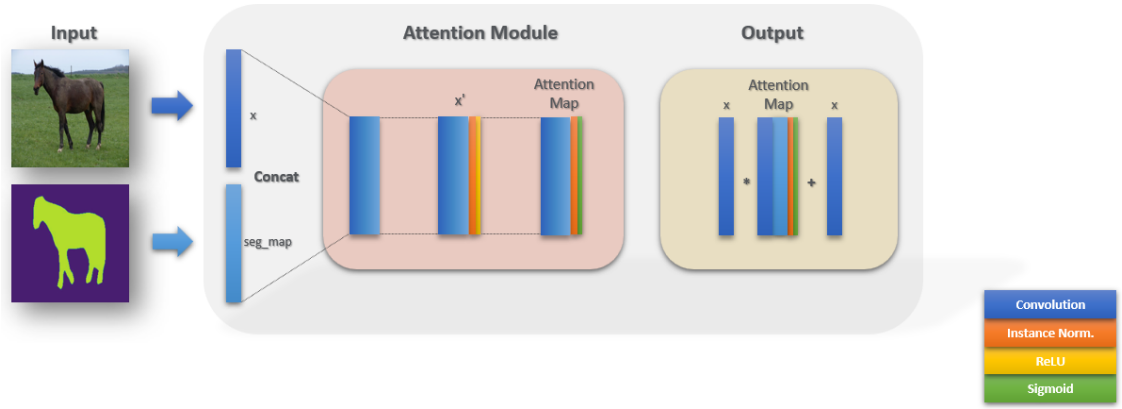


Figure 5: Attention Module

4.4 Discriminator Architecture

The discriminator architecture is composed of four convolutional layers, each followed by instance normalization and Leaky ReLU activations.

The discriminator receives a mini-batch of RGB images as input and produces an 8x8 grid as output. Each cell in this grid indicates the probability that the corresponding patch in the input image is real (true) or fake (false).

It is important to note that the discriminator does not include attention layers, as they complicated the training process without providing significant benefits.

An illustration of the discriminator can be found in Figure 6.

4.5 Image Pooling

Image pooling (Figure 7) is a technique used to stabilize the training of the generator by managing the generated images more effectively. It helps prevent the generator from producing identical or very similar images over time.

How it works First, we maintain a pool of previously generated images. This pool stores a certain number of images (e.g., 50 or 100) that the generator has produced during training.
For each new image generated by the generator, it's added to the pool if the pool isn't full.
If the pool is full, a random image is replaced by the new image with a certain probability. This probability is usually controlled by a parameter (e.g., 0.5), meaning that there's a 50% chance of replacing an existing image in the pool.
During training, when the discriminator evaluates generated images, it samples from this pool

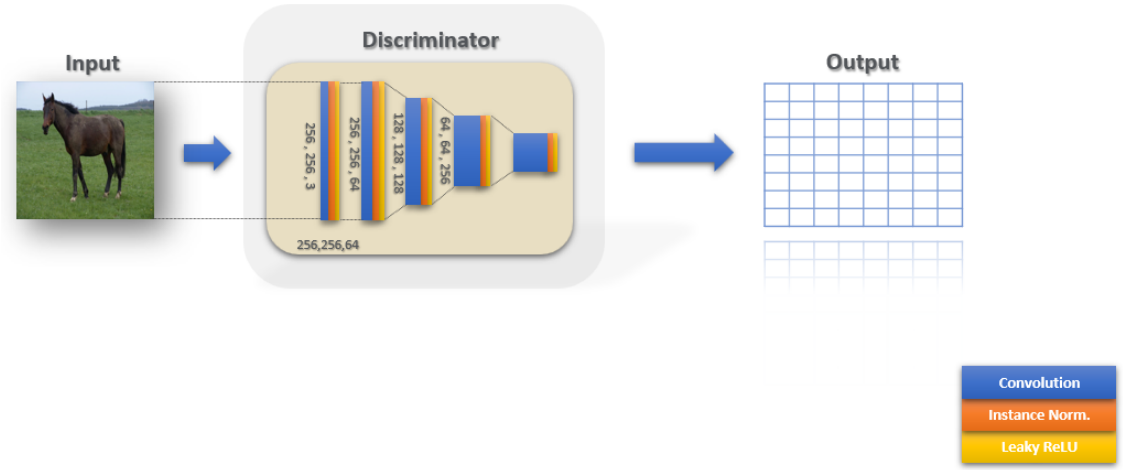


Figure 6: Discriminator Architecture

instead of only seeing the most recent images. This introduces more variability in the images presented to the discriminator. A representative diagram of the image pool structure is presented in Figure 7.

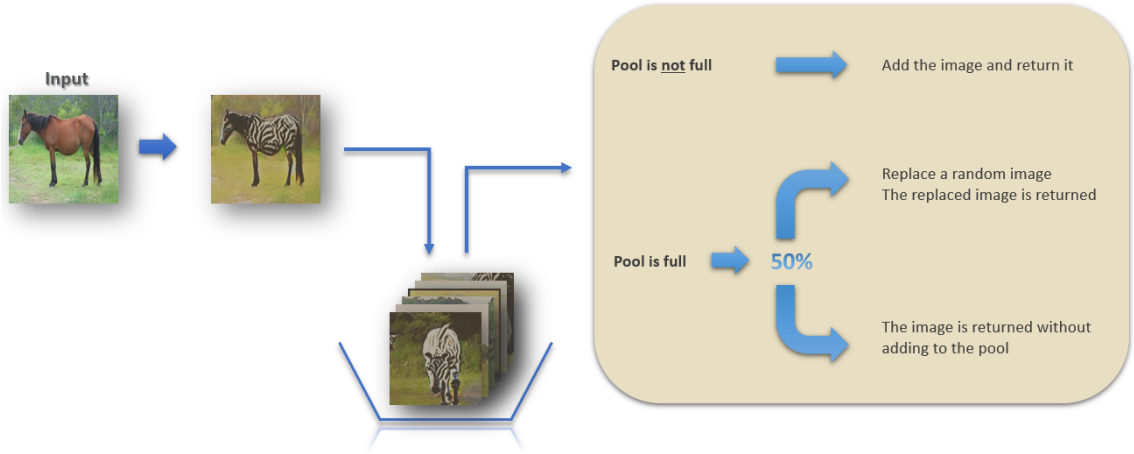


Figure 7: Image Pooling

4.6 Loss Functions

In our implementation, we are using five different loss functions, as listed below:

1. **Adversarial loss**: Encourages the generator to produce outputs that are indistinguishable from real images by fooling the discriminator
2. **Cycle-consistency loss**: Ensures that translating an image to the target domain and back to the original domain results in the same image
3. **Identity loss**: Promotes the generator to maintain the input image's characteristics when the image is already in the target domain
4. **Attention-consistency loss**: Aligns the attention maps of the residual blocks and the decoder in the generator for coherent focus across the network
5. **Segmentation-guided Attention loss**: Guides the generator's attention to focus on specific regions of the image as indicated by the segmentation map

The functions used for loss calculations are Mean Squared Error (MSE) [1] for Adversarial and Attention-consistency loss, and L1 loss [2] for Cycle-consistency, Identity and Segmentation-guided attention loss.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{x_i} - x_i)^2 \tag{1}$$

$$\text{L1 Loss} = \frac{1}{N}\sum_{i=1}^{N}|\hat{x_i} - x_i| \tag{2}$$

where: N is the number of samples in the batch, $\hat{x}_i$ and x are the predicted and ground truth values, respectively.

4.6.1 Discriminators Loss

For each discriminator, the adversarial loss function is used with MSE function 1. The tracked discriminators loss is an average of the two discriminators, i.e.:

$$\text{Discriminators Loss} = 0.5(D_A + D_B)$$

4.6.2 Generators Loss

For the generators, we use a weighted sum of the five loss functions:

$$\text{Generator Loss} = \lambda_1 A + \lambda_2 B + \lambda_3 C + \lambda_4 D + \lambda_5 E$$

where: $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 0.5$, $\lambda_4 = 2.5$, $\lambda_5 = 5$
and:

- A = Adversarial loss
- B = Cycle-consistency loss
- C = Identity loss
- D = Attention-consistency loss
- E = Segmentation-guided Attention loss

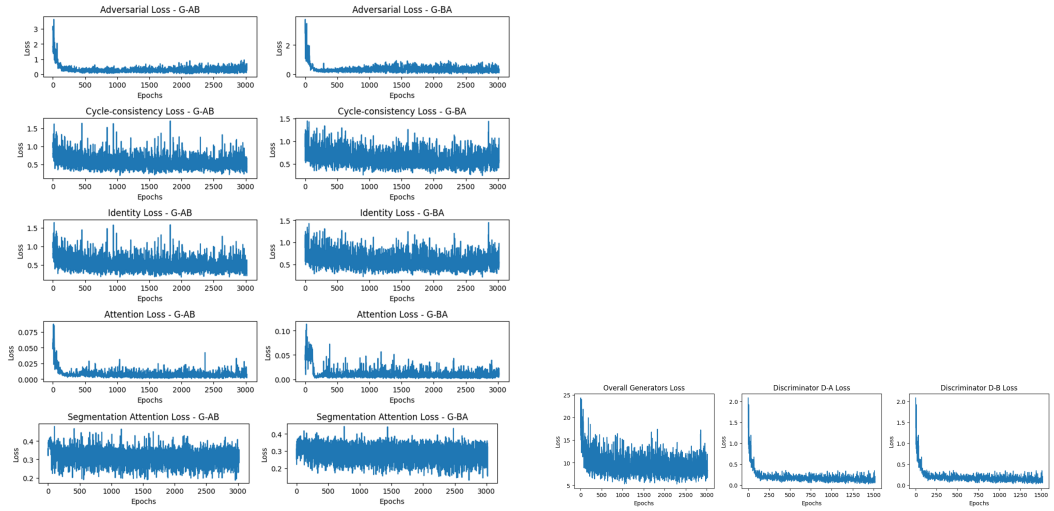
4.7 Training

For training we used the Horse2zebra dataset (1) together with our segmentation dataset which contains segmentation maps for all the training examples.

We employed the Adam optimizer (5) with a learning rate of 2e-4 and betas set to (0.5, 0.999). The batch size was set to 1, with each batch containing an RGB image and its corresponding segmentation map.

To improve training stability and convergence, we initialized all network weights with a normal distribution and set biases to zero.

We monitored all loss functions throughout the training process (Figure 8) and also tracked a sample image from each domain during inference, as the loss functions alone did not provide sufficient insight into the model's performance.



(a) (b)

Figure 8: (a) Generators' loss curves. (b) Overall loss curves for generators and discriminators.

5 Results

We evaluated our models using two metrics: visual inspection and Fréchet Inception Distance (FID) (6).

5.1 Visual Inspection

In this part of the evaluation we will first examine input-output results together with the segmentation map and both attention maps (i.e. residual-block attention and decoder attention).
Next, we will see the benefit of using the segmentation-attention mechanism by comparing images from our attention model to one without attention.

5.1.1 input-output evaluation

In this section, we present examples of horse-to-zebra and zebra-to-horse transformations, along with their corresponding segmentation maps, residual-block attention maps, and decoder attention maps. The goal is to demonstrate the output of our model and the learned attention mechanisms that enabled these transformations.

Figure 9 shows zebra generations from horses (first two rows) and horse generations from zebras (last two rows). In columns a and b, we display the input and output, respectively. Columns c, d, and e show the segmentation maps, residual-block attention maps, and decoder attention maps, respectively. The learned attention maps successfully focus on the segmentation regions (horses/zebras) while also considering the surrounding environment, preventing the model from overly relying on the segmentation maps, which may not be robust.

5.1.2 Attention vs. Non-attention

In Figure 10 we compare our segmentation-guided attention model with the same architecture, excluding the attention layers. The first two columns show the input and output, while columns c and d present the results of the model without attention and the model with attention, respectively.

The red circles highlight artifacts produced by the non-attention model, whereas these artifacts are absent in the attention model's results. This demonstrates that the segmentation-guided attention effectively directs the model's focus, preventing such artifacts, which is its intended purpose.

5.2 Fréchet Inception Distance (FID)

FID (6) is a measure of similarity between two datasets of images. It was shown to correlate well with human judgement of visual quality and is most often used to evaluate the quality of samples

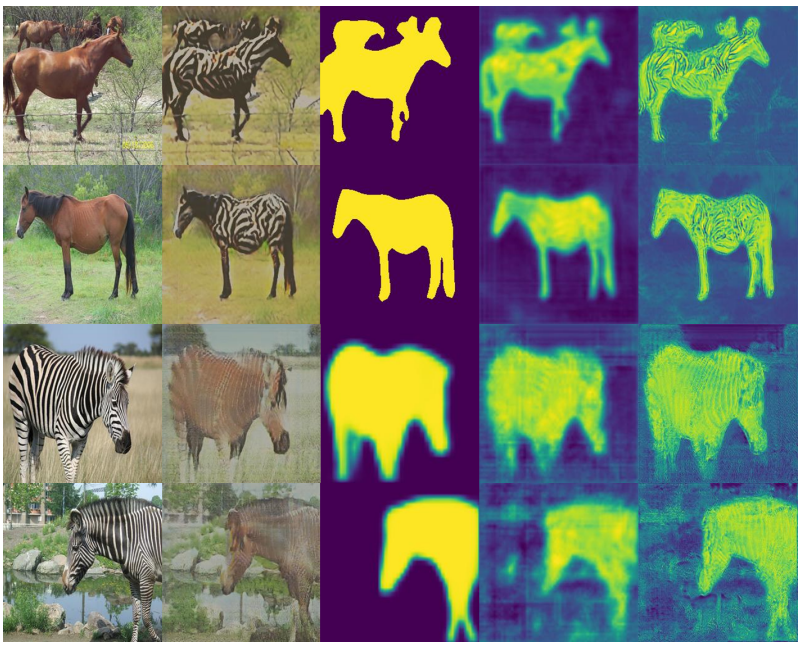


Figure 9: Model generation results. From left to right: (a) Input, (b) Output, (c) Segmentation maps, (d) Residual-block attention map, (e) Decoder attention map

Figure 10: Attention vs. Non-attention.
From left to right: (a) Input, (b) Output, (c) Non-attention results, (d) Attention results

of Generative Adversarial Networks. FID is calculated by computing the Fréchet distance between two Gaussians fitted to feature representations of the Inception network.

We used the official PyTorch implementation of FID (7) to compare the test datasets to out inference datasets.

Table I presents the FID scores calculated for the test datasets.

Fréchet Inception Distance (FID)			
Dataset	Domain	Attention Score	Non-attention Score
Test A	Horses	144.083	163.002
Test B	Zebras	194.083	220.135

Table I: Comparison of Fréchet Inception Distance (FID) scores between attention-based and non-attention models on the Horse2Zebra dataset. Lower FID scores indicate better quality.

Lower FID Scores with Attention The fact that the attention-based model has lower FID scores compared to the non-attention model aligns with expectations. The attention mechanisms help improve image quality and domain translation by focusing on important areas, so it's typical for the FID score to decrease when using attention.
Horses vs. Zebras Domain For Horses (Test A), the attention model achieves a FID of 144.083, while the non-attention model scores 163.002. The difference here indicates that the attention model is generating more realistic horse images. For Zebras (Test B), the FID for the attention model is 194.083, and for the non-attention model, it's 220.135. Similarly, the attention model outperforms the non-attention one in generating zebra images.

In summary, these results make sense, as they show that attention helps in improving the quality of the generated images in both domains.
All the results can be found in our GitHub page (8).

6 Conclusion

In our opinion, based on the results we presented, this project successfully demonstrated the integration of segmentation-guided attention into the CycleGAN framework to address the unpaired image-to-image translation task of transforming horse images into zebra images. The modifications introduced show the potential to significantly improve the accuracy and quality of the translations. Several key conclusions and future directions have emerged from our work:

- **Enhanced Image Quality**: The integration of segmentation-guided attention has the potential to improve the quality of the generated zebras by focusing on relevant features in the horse images, leading to more realistic and detailed outputs.
- **Improved Focus**: The attention mechanism effectively directs the generator's focus to specific areas of the input images, enhancing the model's ability to generate more accurate and visually convincing images compared to a standard CycleGAN.
- **Future Refinements**: As future work, the attention mechanism can be further refined and optimized for even greater performance. Improvements could include fine-tuning the segmentation layers or introducing more sophisticated attention strategies.
- **Potential for Broader Application**: The framework and techniques presented here can be expanded to other image-to-image translation tasks in various domains, demonstrating the versatility and scalability of this approach.

References

[1] Zhu, Jun-Yan *et al.*, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[2] Isola, Phillip *et al.*, *Image-to-Image Translation with Conditional Adversarial Networks*, 2018. arXiv: 1611.07004 [cs.CV]. **available at**: https://arxiv.org/abs/1611.07004.

[3] Chen, Liang-Chieh *et al.*, *Rethinking Atrous Convolution for Semantic Image Segmentation*, 2017. arXiv: 1706.05587 [cs.CV]. **available at**: https://arxiv.org/abs/1706.05587.

[4] He, Kaiming *et al.*, *Mask R-CNN*, 2018. arXiv: 1703.06870 [cs.CV]. **available at**: https://arxiv.org/abs/1703.06870.

[5] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412.6980 [cs.LG]. **available at**: https://arxiv.org/abs/1412.6980.

[6] Heusel, Martin *et al.*, *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, 2018. arXiv: 1706.08500 [cs.LG]. **available at**: https://arxiv.org/abs/1706.08500.

[7] Seitzer, Maximilian. *pytorch-fid: FID Score for PyTorch*, https://github.com/mseitzer/pytorch-fid. Version 0.3.0. 2020.

[8] Lior Yaacov, Daniel Manor. *Horse2zebra Challenge - CycleGAN with Segmentation-guided Attention*, 2024. **available at**: https://github.com/LiorYaacov/Horse2zebra.