Titan-ification using Vision LLMs on Custom Attack on Titan Dataset

Kavin Anand
Stanford University
Department of Computer Science
akavin@stanford.edu

Gaurav Rane
Stanford University
Department of Computer Science
grane@stanford.edu

Abstract

The transformation of human characters into titans in the anime "Attack on Titan" presents an intriguing problem in computer vision. A central premise of this show was the mystery surrounding certain characters and their abilities to transform into a massive humanoid titan. It is an interesting problem, given an input human image, generating the titan-ified version. Our project aims to generate titanified images from human character faces using a fine-tuned vision language model (LLM). By leveraging a custom dataset and advanced augmentation techniques, we hope to replicate and predict these transformations, potentially providing insights into character identities earlier in the narrative. We leveraged GANs, cGANs, and experimental ViTs and used SSIM as our quantitative metric. A simple GAN architecture achieved the highest SSIM score of 0.2266 but produced much lower quality images compared to the pretrained cGAN pix2pix model we were able to leverage on our custom dataset. Future works include expanding our dataset and trying more robust pretrained models, as well as new architectures for the GAN to make it deeper.

1. Introduction

Attack on Titan is one of the most popular shows in the world. Its premise relies on extensive mystery. The viewers are left in the dark who the villain is and the secrets of the world. At the center of it all is the ability for certain characters to transform into Titans – massive, humanoid, maneating beasts that are at the heart of the show's horror. Our inspiration was to create a model that, given a human face, can generate an output expected titan image. This is designed to be a very fun tool for avid fans to experiment titan forms of their favorite characters and perhaps begin uncovering the mystery of the show before the show gets there.

1.1. Data type

The input is an image. Images were cropped and normalized to size either 256x256x3 or 512x512x3 where the 3 is the three color RGB channels. The output is going to be an image of the same size as the input. We experimented with two input sizes to see if the model was able to better capture features from the larger image that included more background information or if that context was irrelevant and the smaller image was better in focusing on the direct face-to-titan transformation. The smaller image is also easier to train.

2. Related Works

Our main inspiration for this project comes from a medium post that created their own custom dataset for attack on titan and utilized it for a classification task [21]. However, our approach is geared towards generation over classification, so we must look elsewhere for a generative based model.

2.1. ViT Approaches to Image Generation

Ever since the transformer architecture was introduced the world of natural language [19], the AI world has begun overhauling their use of CNN's with transformers. In computer vision specifically, [5] the transformer architecture has proven as a comparable approach to the common convolution method. When pretrained on the ImageNet challenge dataset [16], it showed a similar performance on classification tasks as the convolutional approach without requiring as many compute resources. Our task is image-to-image detection however, which means that we need to look towards a generative model. Stable diffusion models offer an avenue into this type of image construction, but their reliance on text input as a basis for generation does not exactly meet our task at hand [14]. Thus, we need to look towards finetuning an already pre-trained model for this image-to-image generation task. With ViT's though, it's transfer learning has been shown to improve model efficiency on tasks like object detection when the model was pretrained on unsupervised data [13]. Thus, there is a potential for us to use base ViT models and expand their capabilities for titan generation through further fine-tuning.

2.2. Pix2pix

Since not many image-to-image transformer models exist, cGANs still serve as our most stable and well tested heuristic when it comes to image generation. The pix2pix model was developed with the specific use case of mapping input images to output images [10]. Pix2pix builds on the U-Net generator [15], and using the generator/discriminator architecture is able to use a loss function to learn unique relations between features of the input and output image. Already, people have been using the pix2pix model for different generative tasks. Within medicine, it's been used to denoise myocardial perfusion images [1] and also just to generate medical images since medical image datasets are sparse [2]. Additionally, it's been used for generating street level views from google maps representations of structures [8], taking away the background of an image to isolate for an object, and also generating a full hand drawn painting from a simple stick figure representation of an individual. Pix2pix can handle diverse inputs and mappings, so we believe it will be a valid starting point in our human image to titan image translation task.

3. Data

Extensive work was done in collecting images for our dataset. Due to the novelty of our project, there were no existing datasets that we could find online to leverage. We manually had to go through the scenes in the show and collect frames from characters that are known to transform to titans. To train our model, we need an input and label image which corresponds to a picture of a character in human form and a picture of the same character in titan form. This process was incredibly intensive as we needed to identify scenes where just the character or titan form was in scene, with minimal distractions in the background or the presence of other characters, and label the images so that we could accurately pair the right human with the right titan for model training.

In this time intensive process, we extracted screenshots of the show, labeled the images, and created a 1:1 mapping of human faces to titan forms. At the end of our data collection, we had 676 total images, which are 338 image pairs of human characters to titans. Data augmentation is described below, but after it was done images were channel-wise normalized to size 256x256 for majority of our experiments and we had 4056 images, or 2028 image pairs.

3.1. Data augmentation

We greatly expanded on our data preprocessing and augmentation steps however to magnify the size of our dataset. Vision models typically require vast quantities of data to accurately train and we needed to find ways to better augment the size of our dataset to train more robust models.

We augmented our dataset by performing many random transformations through the torchvision transforms function. We first employed a color jitter, randomly changing the brightness, contrast and saturation by a variance of 0.4 and hue by 0.1 with an 80% chance of application. Changing lighting and color variations aids the model to generalize better to different color conditions. We then also introduced random horizontal flips with 50% probability and random grayscale with 20% probability to introduce more variability in orientation and helping the model handle grayscale and learn the underlying image structure without color information better. Finally normalization was done as well, ensuring image pixel values have a mean of 0 and standard deviation of 1 to stabilize and speed up the training process before converting images to PyTorch tensors so that they could easily be fed into a DataLoader for training in our networks.

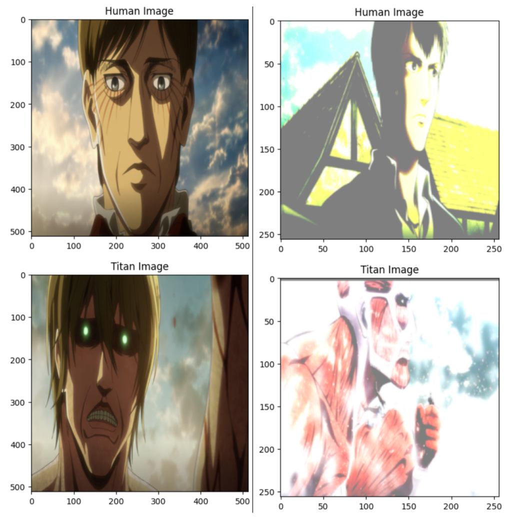

Figure 1. Example of two training pairs of images. Left column is human and titan raw image of size 512x512. Right column is a different human and titan pair from the augmented dataset of size 256x256.

4. Methods

Image to image generation is a difficult task. We knew that our dataset was fairly limited for the task at hand and that reasonable results are only likely to come from larger, pre-trained models that we can then finetune the weights of using our custom dataset of a few thousand images. These larger pre trained GPTs are created from datasets of 1M to 300M images, a scale we simply cannot compete on.[6] However, we develop a Generative Adversarial Network (GAN) as well to serve as a baseline.

4.1. GAN

Generative Adversarial Networks (GANs) are a class of machine learning frameworks where two neural networks, a generator and a discriminator, are trained simultaneously through adversarial processes. The generator creates fake data from random noise, while the discriminator evaluates whether a given data instance is real (from the training data) or fake (generated by the generator). The objective of the generator is to produce data that the discriminator cannot distinguish from real data, while the discriminator aims to correctly identify real and fake data. In ideal situations, both the generator and discriminator are equally as ”strong” meaning that one doesn’t learn much faster than the other. Consider situations in which one is stronger than the other. If the generator is stronger that means it can get away with generating any image that may not resemble our ideal label as our discriminator is too weak to distinguish real from fake. In the opposite case, the generator is penalized from ever learning because it can never ”win” against the discriminator who always discerns its correct output. Both cases result in poor output where the resulting image isn’t tied closely to the class labels.

Mathematically, the GAN relationship can be formulated as a minimax game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{1}$$

where G is the generator, D is the discriminator, x is a real data sample, and z is a random noise vector.

4.1.1 Simple GAN Architecture

The first model relies on a simple Discriminator and Generator architecture. The Discriminator contains a Flatten() function that squashes input into a linear vector before passing it into a multiple Linear layers followed by LeakyReLU activation functions. We decided to use this over normal ReLU to smoothen the gradient update steps and hopefully achieve better training results.

The Generator then goes from multiple smaller dimension Linear layers followed by more standard ReLU activation functions before it finally outputs the original 256x256x3 vector in the final step followed by a Tanh() activation.

4.1.2 Loss Functions

The discriminator loss is calculated as the sum of the binary cross-entropy losses for real and fake images. We utilize the Pytorch BCE loss function that is numerically stable in our calculation here. We didn’t choose to with Least Squares Loss function, or an LSGAN approach, because we noted in research it had excessive penalties for outliers, leading to reduced sample diversity. [3] Considering the nature of our dataset, and the lack of rigorous standardization, we felt a conventional BCE loss would be more suitable.

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \tag{2}$$

The generator loss is calculated as the binary cross-entropy loss of the discriminator being fooled by the fake images:

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z(z)}[\log D(G(z))] \tag{3}$$

4.1.3 Training procedure

The training process involves alternating between updating the discriminator and the generator. In each iteration, we:

1. Sample a batch of real images from the training set.
2. Sample a batch of noise vectors from the uniform distribution.
3. Generate fake images using the generator.
4. Compute the discriminator loss and update the discriminator.
5. Compute the generator loss and update the generator.

Algorithm 1 Training GAN

```
1: for number of training iterations do
2:     Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_z(z)$
3:     Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$
4:     Update the discriminator by ascending its stochastic gradient:
           $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$
5:     Update the generator by descending its stochastic gradient:
           $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$
6: end for
```

4.2. Pix2pix

We utilized the Pix2Pix model, a generative adversarial network designed for image-to-image translation tasks [10]. The model comprises both a U-Net generator and a PatchGAN discriminator, which we fine-tuned for a titanification application. This model was chosen for its suitability for direct image-to-image generation, allowing us to fine-tune it with our titan images and verify its performance.

The U-Net generator was coined by [15], and its architecture consists of eight down-sampling blocks followed by eight upsampling blocks. Downsampling includes eight convolutional blocks followed by a LeakyReLU activation function, with batch normalization applied to all but the first and last blocks. The eight upsampling blocks incorporate ReLU activation and batch normalization on all layers, with dropout applied to the first three layers to avoid overfitting.

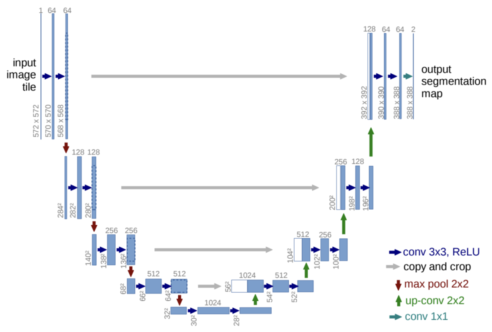

Figure 2. As taken from the paper directly, this figure illustrates the U-Net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The PatchGAN discriminator verifies each 70x70 patch and determines its realness. It uses four convolutional blocks to downsample the spatial dimensions while increasing the feature channels, followed by a final convolution layer that outputs a probability indicating the realness of the image.

4.2.1 Pix2pix Loss Functions

For training, we used a combination of GAN loss and pixel-wise L1 loss. The GAN loss, defined by BCEWithLogitsLoss, measures how well the generator fools the discriminator and how accurately the discriminator identifies real versus fake images. To ensure the generated images closely matched the real titan images, we used a simple L1 pixel-wise loss, chosen over L2 to avoid the blurriness associated with the latter.

The total loss for the generator is a combination of the GAN loss and L1 loss. The GAN loss is explained in section 4.1.2 under equation 2. On the other hand though, the L1 loss, which ensures pixel-wise similarity between the generated and real images, is given by:

$$\mathcal{L}_{L1} = \mathbb{E}_{x,y}[\|y - G(x)\|_1] \tag{4}$$

The total loss for the generator combines the loss above with the loss from equation 2.

$$\mathcal{L}_{Generator} = \mathcal{L}_{GAN} + \lambda \mathcal{L}_{L1} \tag{5}$$

where λ is a weighting factor that balances the two losses. As specified by the pix2pix paper [10], the value of lambda was found to operate best when set to 100.

4.2.2 Pix2pix Training

The training process was carried out over 25 epochs and the images used for training and evaluation were resized to dimensions of 256x256 pixels. During training, the human images and corresponding titan images were loaded into the model in batches of 1, with the generator attempting to create titan images from human images, and the discriminator evaluating their authenticity.

In each epoch, we get the loss of the real human titan pair, generate the fake titans, and then calculate the fake loss using the same BCE loss function. Then, the discriminator loss is the average of those two and we backrpop to update discriminator parameters. With the generator, we have the GAN/BCE loss from its generated images and fooling the discriminator, but then we combine that with the L1 loss of the generated titan versus the real image, making sure to backprop and update the U-Net parameters after.

We also kept a validation set so we could measure the generator's performance amongst unseen/untrained images, and we additionally used a structural similarity index (SSIM) to evaluate the quality of generated images. For each best validation loss and SSIM score, we saved the generator/discriminator pairs.

4.3. ViT

The transformer architecture were introduced as a novel concept for natural language processing tasks, but a similar architecture can be applied to computer vision tasks [5]. These ViT's are shown to perform with their convolutional counterparts, while requiring much less compute resources. Our approach attempts leverage several ViT's that have been pretrained on large vision datasets like ImageNet and ResNet, and then finetune them for our titanification prediction task. Specifically, we will use two pretrained models, then we apply transfer learning to fine-tune the models for our generation task.

4.3.1 ViT Model Architecture

We use the same core model, vit_base_patch16_224, but the models had different training sets and were loaded slightly differently. The timm model was trained on ImageNet-1k, consisting of 1.28 million images across 1,000 classes. The Hugging face version of the model was trained on ImageNet-21k, consisting of 14 million images across 21,000 classes. Being the base version of ViT, as expressed in the paper, the patches for the ViT are 16x16 pixels, and the model architecture consists of 12 transformer encoder layers, 768 hidden dimensions, and 12 attention heads [5]. Given that the model was trained on image classification, we modified the final layer to include a linear layer mapping to the output image size of 3x224x224, with a TanH activation function.

4.3.2 ViT Loss Functions and Training

For both models, we trained the timm loaded model for 20 epochs and the hugging face loaded model for 10 epochs. We set out twenty percent of our images for the validation set, and we calculated our validation loss after each epoch. For the loss, we used a mixture of MSE loss and perceptual loss. We thought that too much emphasis on MSE would create too much bluriness, so we introduced the perceptual loss [11] as well to make sure that the overall shape and figures of the output images is conserved in our model as well.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{6}$$

$$\text{Perceptual Loss} = \sum_{l} \frac{1}{C_l H_l W_l} \sum_{c=1}^{C_l}\sum_{h=1}^{H_l}\sum_{w=1}^{W_l} (\phi_l(x)_{c,h,w} - \phi_l(y)_{c,h,w})^2 \tag{7}$$

4.4. Alternative Approaches

Multiple alternative approaches are described in detail below but we finally chose GAN as our baseline, cGAN as our robust model architecture, and ViT as an experimental approach due to the extensive literature and dominance these models have on image generation. GANs are known to generate highly realistic images by training two networks via the min-max game and can capture high frequency details and produce sharp images. cGANs are similar but leverage even more information to aid in teh generation process that allows for more controlled output. Both these architectures also follow from the style of our image/true label pairs that our dataset consists of. Finally we wanted to explore ViTs as they leverage the transformer architecture which has made unfathomable leaps in success the past couple years, primarily in NLP but also in computer vision. Long range dependencies are captured here and we hoped to capture Neural Style Transfer (NST) to mitigate our smaller dataset problem and take advantage of the larger swath of images the pre-trained CNNs are trained on.

4.4.1 VAE

In addition to the methods above, we considered other generative models that could've been used for our task. The first, Variational Autoencoders (VAEs), are a type of generative model that learn the underlying distribution of data by optimizing a lower bound on the log-likelihood of the observed data.[12] This means that fundamentally VAEs are more controlled than GANs, they provide a probabilistic manner of describing an observation in latent space leading to controlled image generation. However, one major limitation is that since the optimize reconstruction loss, VAEs do not prioritize high frequency details and therefore tend to be blurrier and realistic than GANs. [4]

4.4.2 DDPM

Another advanced strategy that we briefly considered was Denoising Diffusion Probabilistic Models (DDPMs), a class of generative models that start from an input image, iteratively add noise to it and then learn to denoise from that noisy image to generate new image samples. [7] These models are shown to have remarkable performance in generating high quality images and can produce highly detialed images, even for a smaller dataset as it doesn't have to explicitly learn to create every facet of the image from scratch. It starts from a template.

Forward Diffusion:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

where x_t is the data at timestep t, β_t is a variance schedule, and $\mathcal{N}$ denotes a normal distribution. [18]

Reverse Diffusion:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)I)$$

where μ_θ and σ_θ are learned parameters of the model. [9] The reverse process is trained to model the reverse of the forward diffusion, thereby allowing the generation of clean data samples from noise. However we decided not to go with this approach due to the computationally expensive and slow training time required by iterative denoising processes. We didn't have the compute resources nor the capability to test multiple large models with this process and were lacking easy-to-use libraries that were familiar and modular as the ones we used for the other models. This made it less practical for us to implement and explore.

5. Results and Discussion

Implementation details, such as hyperparameter and optimizer choice, are included in the below subsection. We obtained a mix of quantitative and qualitative results. The reason being is that image generation itself is a difficult task, and it's further difficult to compare with pure metrics how your generated images compare to test images. We used Structural Similarity Index Measure (SSIM) as our quantitative metric to optimize. SSIM is a perceptual metric that quantifies image quality degradation caused by processing such as normalizing or image generation. It works better than traditional measures such as Mean Squared Error which only captures pixel-wise differences as SSIM assesses changes in structural information, considering luminance, contrast and structure aligning it better with human visual perception.[20] This provides us a more holistic view of image quality. We decided these pros were worth the drawbacks of increased computational complexity of SSIM because we felt we were only going to obtain SSIM results for the hold-out test set which is much smaller and should be relatively fast regardless.

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{8}$$

μ_x, μ_y are the mean intensities of x and y, σ_x^2, σ_y^2 are the variances of the two images, $\sigma_x y$ is the covariance and C_1 and C_2 are constants to stabilize the division.
The SSIM scores of the primary models are included below.

Model	SSIM Score
Simple GAN	0.05107536
Simple GAN Aug	0.12660249
Simple GAN Aug Large	0.22667356
pix2pix val	0.11689669
pix2pix ssim	0.10948225

Table 1. SSIM Scores for Various Models

The Simple GAN is the baseline model that we built. It performed the worst out of all the models. The Simple GAN Aug is the model that was trained on 6x the training information, essentially on the augmented dataset. The Simple GAN Aug Large performed the best out of the Simple GANs that we trained, large meaning that it was trained for significantly longer. The nuances of these categorizations are included in the Implementation Details section. It wasn't very surprising for us to see these results. It made sense that the augmented dataset would perform better as there were many more images and the images were more robustly manipulated. GANs typically take 50k-100k images to properly train however, so we expect to continue to see sharp rises in performance even with a simple GAN architecture.[17] However we were clearly limited as curating such a large dataset would be infeasible given the constraints. Next, training for longer time horizons offered much better results as well, even on the same dataset. The loss continued to drop and we see better results for the GAN. This leads us to believe that if we had access to more compute we could obtain even stronger results but this is the limits of the Colab Pro compute units allocated to us. The two pre-trained cGANs, pix2pix val and pix2pix ssim are also included in the table. It was surprising that these larger models didn't perform as well, especially considering that they were pretrained on millions of images. We didn't measure SSIM scores for the ViT models. These severely underperformed and the transfer learning experiment didn't go nearly as well as originally hypothesized or intended. The ViTs weren't able to appropriately transfer their robustness in classification to our task of image generation, and constantly produced fully grayed out images.

However, qualitative analysis is also incredibly important for evaluating our models as fundamentally image generation over a show and its characters will invoke a very human response. To do this, we generated images on characters from the hold-out test and tried to see how close to the label the model was able to output results. There are also many ways that weaker models can hide behind SSIM scores – models can generate blurry images that might have similar luminance and structural patterns but fail to capture any of the details that you'd expect in a high quality output.

5.0.1 Implementation Details

The models were implemented using PyTorch in Google Colabs, leveraging python scripts that we developed ourselves or developed in classwork. Adam optimizers were used for both the discriminator and generator, and we used grid search to find optimal hyperparameter values for learning rate, beta1 and beta2. Training was done on an accelerated compute A100 and L4 GPU on Colab Pro to enable training with more RAM and speed up training times. The Simple GAN architecture is described in methods. Augmented means using the augmented dataset as described in dataset. Large means training for 25 epochs which is also the standard for the two pix2pix models. The smaller simple GAN is trained for 10 and the simple pix2pix is for 5.

5.1. Simple GAN

Figure 3 depicts qualitative output of the three Simple GANs. The input human image, real titan image and the generator output image are included row by row for the Simple GAN, Simple GAN Aug, and Simple GAN Aug Large respectively. As we can see, qualitative results tell a shockingly different story. The first model has the worst output which is expected from the SSIM score. However, the latter two models should've had better output considering their SSIM, the Simple GAN Aug Large especially so as it reached scores of 22%. This failure in generating proper titan images I think can be attributed almost entirely to dataset size. The loss values were going down on both the Discriminator and Generator but there simply wasn't enough images to train on. Furthermore, the architecture could've been expanded to include a more robust design that included Convolutional Neural Networks and DropOut. We implemented these fixes in the cGAN, Pix2Pix.

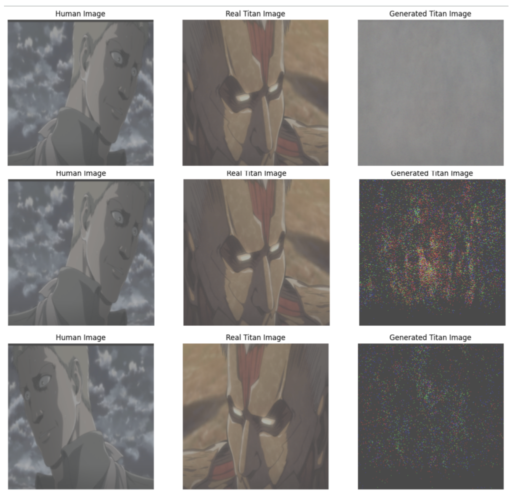

Figure 3. Row by row: Simple GAN input human image, real labeled titan image, Generator output, Simple GAN Aug, Simple GAN Aug Large

5.2. ViT

The purpose of the ViT models was to see whether we could use transfer learning to get a model trained on image classification to perform on image-to-image generation. Training for the base ViT model loaded by Timm was done in 20 epochs, and training for the base ViT model loaded by Hugging Face was done in 10 epochs. With MSE error and perceptual loss as the guiding factors of the model, we saw practically no change in validation loss over the epochs for Timm, and for Hugging Face we saw no change in validation loss at all. Thus, when we visualized the images our model would generate, it wasn't surprising that we received blank gray images as outputs for both models as seen in Figure 4.

Since the generated image was essentially just a flat color across 224x224 pixels, we decided not to include it in the SSIM table above. It seemed like transfer learning wouldn't be too possible without many more images, but we still had one more cGAN to test out.

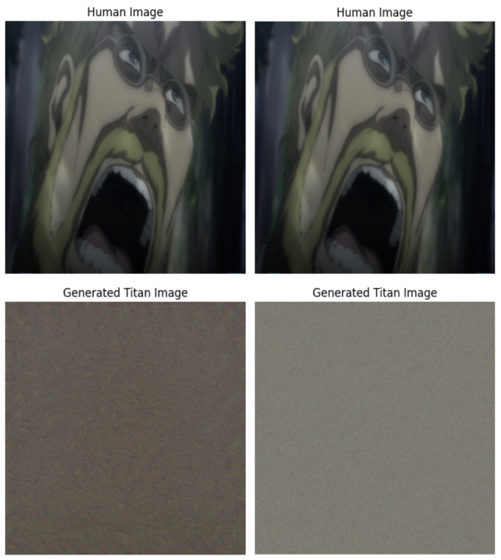

Figure 4. Left two: Input human image and output from ViT pretrained on ImageNet-1k, Right two: Input human image and output from ViT pretrained on ImageNet-21k.

5.3. Pix2pix

The ViT model's showed us that transfer learning wasn't the best idea when it came to image-to-image generation, so we tried our final model, the pix2pix. To make sure the model would yield interpretable results, we did a small test batch with a simple 5 epoch iteration. We only loaded the original dataset, and we saw that after a simple 5 epochs of training, the model was showing us generated images that actually resembled the likeness of their inputs, as seen in Figure 5.

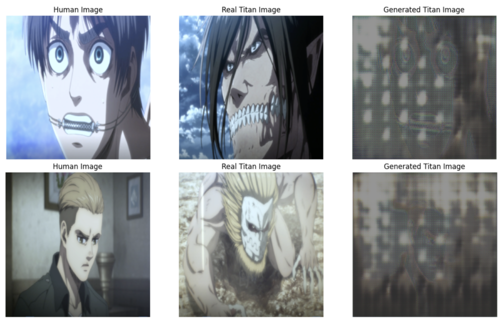

Figure 5. Both rows consist of human input, real titan label, and pix2pix 5 epoch trained no augmented data model output.

The model clearly held some potential, so we augmented the dataset as specified in the dataset section and loaded it. The training dataset was now a total of 2331 images, with the rest being used for the validation set. When we split validation from training, we made sure to only pull the validation set from the non-augmented image pairs. With the data prepared, we proceeded with the full 25 epoch training. We had four statistics reported throughout each epoch: discriminator training loss, ssim score, generator training loss, generator validation loss. All these values were averaged across each epoch, and there progress during training is shown below in Figures 6 and 7.

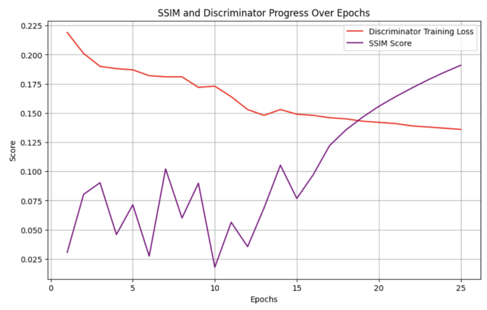

Figure 6. SSIM (Purple) Score and Discriminator (Red) Loss over Epochs

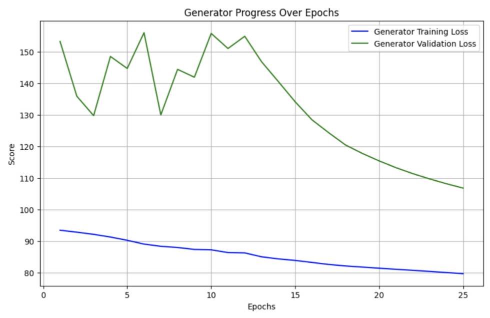

Figure 7. Generator Training (Blue) and Generator Validator (Green) loss over Epochs.

There seems to be a lot of instability in the earlier epochs with the ssim score and the generator validation loss. However, they seem to stabilize later on and we can see that they are moving in the direction we want. We saved two generator models during training; one model we saved was the model that got the best generator validation score and the other was the one with the best SSIM score. When we visualize the images that are outputted by both, we see them as such in Figure 8, below.

Earlier on the table, we saw that the SSIM score was slightly better for the best validation model, which is surprising because we would expect the model that got a higher SSIM score during training to achieve a better score during test time. However, when we look at the images qualitatively, the best SSIM model does a much better job at identifying features like eyes and the titan images generated by the best val model tend to be much blurrier.

6. Conclusion & Future Works

We challenged the difficult problem of generating titans from human images. We found that definitely GANs were the best approach given existing literature on the subject, ease of use and support in terms of existing packages, and computational limits. However, our dataset was a major limiting factor. To build a more robust GAN, research states the lower bound to be 2-3 magnitudes higher than the dataset that we were working with. This is supported by our results as well – the models trained on the augmented dataset performed unequivocally better and the models trained for more epochs performed better on both the training and validation set. This supports the theory that data was a large bottleneck. The best performing model on paper was the Simple GAN Aug Large, which is our simplest architecture but trained on the augmented dataset for the longest time of 25 epochs measured by achieving the best SSIM score of 0.2266. However, after qualitative analysis, it was clear it wasn't capturing the real structural similarities or shapes indicative of titans. This was much better represented by the cGAN that was pretrained on millions of images before finetuned on our custom dataset, pix2pix. This model, while although achieving lower SSIM scores of 0.109, produced much better qualitative output where relevant features like eyes could be identified.
In the future, if we have access to more compute, we'd like to explore the DDPM models. These are supposed to be able to rival the output of GANs and are able to train on much smaller quantities of data as the model isn't tasked with a novel generation task but stripping away noise to uncover unique images that match the label. Unfortunately given time and compute restraints this model couldn't have been explored. We also lacked subject matter expert in these architectures as they weren't covered in homeworks. Another way to optimize this project would be to hire a team for more accurate and extensive labeling. With a dataset that rivals how modern GANs are trained, and a more robust GAN architecture, or just leveraging the existing pretrained pix2pix model, we'd expect to see much better results, quantitatively and qualitatively.

7. Contributions & Acknowledgements

Kavin and Gaurav contributed extensively to the project ideation, data collection, experimentation and writeup steps. Granularly, both were involved in the brainstorming process and ideation for possible models that'll help with the task. Attack on Titan is a show that both of us are incredibly fond of and thought this project would be an incredible opportunity to see the potential of computer vision in a domain we're knowledgeable of. Next was data collection. Both of us spent many hours together scraping image pairs from different seasons of the show. Next we both split

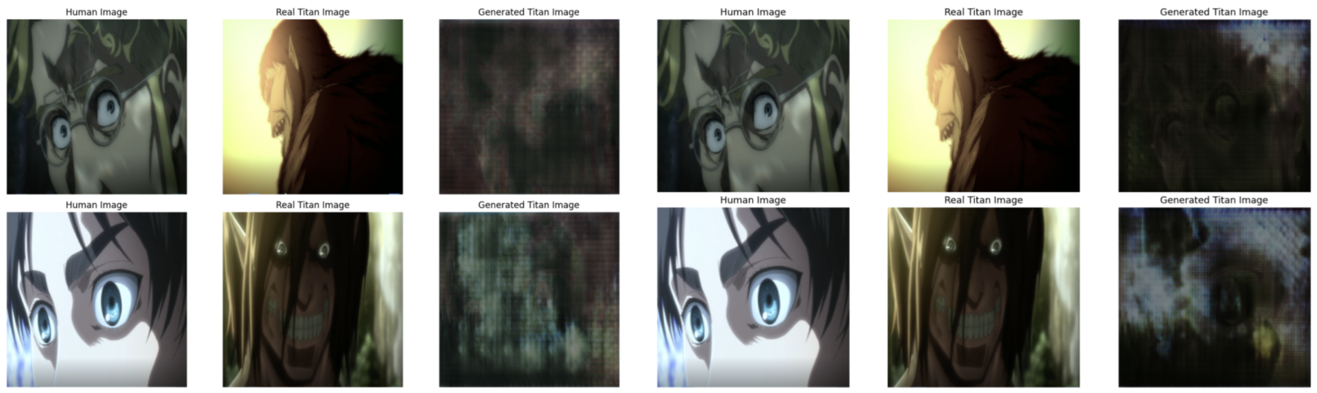

Figure 8. Left 6: Pix2pix model with best validation results trained for 25 epochs. Right 6: Pix2pix model with best SSIM scores trained for 25 epochs. Compares output images across exact same human/titan pairs

up the experimentation work so that we're running relevant models and wrote about our relevant models in the methods section of the writeup. Finally we both contributed to the paper and poster presentation as well.

Individual highlights: Kavin was more involved with data preprocessing, augmentation and making it easily digestable by a Dataloader for future training. Kavin focused on the GAN baseline models as well. Kavin also worked on these sections in the writeup, the introduction and the dataset, and parts of the results section.
Gaurav focused extensively on the larger model experimentation and work on the transfer style models. This includes both the ViT and the cGAN such as the pix2pix model. Gaurav also worked on these sections in the writeup, the literature review, and parts of the results section.

We'd like to acknowledge Stanford CS231N professors and teaching staff. Via quality lectures and assignments we were able to finetune our skills to work on a large individual project. We are not using this project as part of another class or research team.

References

[1] A. Aljohani and N. Alharbe. Generating synthetic images for healthcare with novel deep pix2pix gan. *Electronics*, 11(21), 2022.

[2] A. Aljohani and N. Alharbe. Generating synthetic images for healthcare with novel deep pix2pix gan. *Electronics*, 11(21):3470, 2022.

[3] C. Dewi, R.-C. Chen, Y.-T. Liu, and H. Yu. Various generative adversarial networks model for synthetic prohibitory sign image generation. *Applied Sciences*, 11:2913, 03 2021.

[4] C. Doersch. Tutorial on variational autoencoders, 2021.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[6] H. Gani, M. Naseer, and M. Yaqub. How to train vision transformer on small-scale datasets?, 2022.

[7] A. M. Gitau. A friendly introduction to denoising diffusion probabilistic models. July 2023. Accessed: 2024-06-02.

[8] J. Henry, T. Natalie, and D. Madsen. Pix2pix gan for image-to-image translation. *Research Gate Publication*, pages 1–5, 2021.

[9] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020.

[10] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[11] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[13] Y. Li, S. Xie, X. Chen, P. Dollar, K. He, and R. Girshick. Benchmarking detection transfer learning with vision transformers, 2021.

[14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[15] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[17] I. Salian. Nvidia research achieves ai training breakthrough using limited datasets, December 2020. Accessed: 2024-06-02.

[18] V. Singh. An in-depth guide to denoising diffusion probabilistic models ddpm – theory to implementation, March 2023. Accessed: 2024-06-02.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[20] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. Image quality assessment: from error visibility to structural similar-

ity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[21] T. Whitehurst. Attack on titan image classifier w/custom dataset and keras. *Medium*, 2018. Accessed: 2024-06-05.