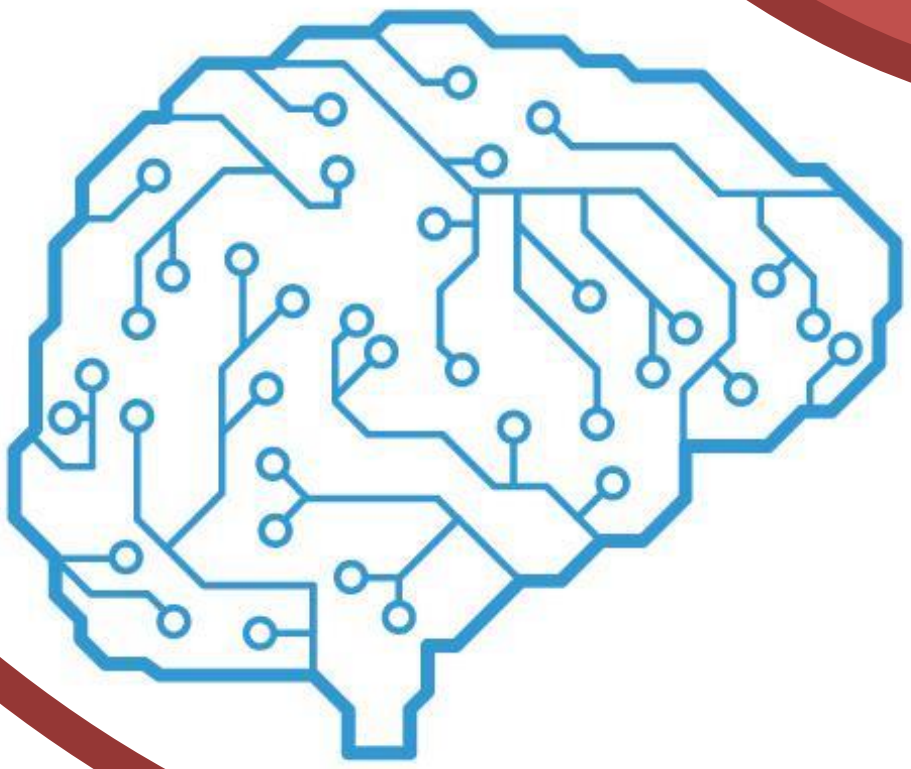


Fashion.AI: Project Report

Submitted in partial fulfillment of ISY5002 Pattern Recognition: Practice Module

---

Team Members:

- CHEN MINGYI EDMUND/ A0031105E
- VISWANATHAN CHANDRASHEKAR/ A0088591N
- CHENG YUNFENG/ A0215320Y

NUS National University of Singapore | ISS INSTITUTE OF SYSTEMS SCIENCE

Contents

1 Executive Summary

The ease of access to digital technologies had been rapidly transforming the e-commerce industry for retailers and consumers. Then in 2019, boon or bane, COVID19 became a catalyst that spurred online sale more than ever before. With this impetus our project team had our eyes set on the fashion retail industry, and proposed three tracks in which the fashion industry could be transformed with the help of AI technologies:

Generating Fashion Articles for Design Inspirations. In this track, we created a model that can auto-generate different fashion article designs. It served to jumpstart the design process with the design inspirations generated. We explored Generative Adversarial Networks (GAN), and ultimately used it to generate three different clothing designs; women's tops, women's dress, and men's tops.

Swapping Model Clothing Style for Product Marketing. Marketing and advertising are big components in the fashion industry. We hope to aid in this aspect by auto-swapping the clothing style of a human model so that a single photoshoot could possibly serve the purpose of advertising for multiple clothing styles. Cycle GAN was use for this purpose. We created and a trained a Cycle GAN that could swap between long-sleeves and short-sleeves clothing designs.

Virtual Try-on for Consumers. A recent survey concluded that more than 50% of people are uncomfortable with using a dressing room right now[1]. As such, we like to allow people to try on clothing virtually, in the safety of their own homes. We researched and explored using Conditional Analogy GAN to solve this problem. A CAGAN model was created and trained on a dataset comprising of women's tops. The model allowed users to, virtually, swap their clothes to another.

To demonstrate our ideas, we had overlaid the three models with an easy to use web app.

[1] https://www.huffpost.com/entry/is-it-safe-try-on-clothes-stores-coronavirus_l_5f0484dac5b6e97b568a6c59

2 Problem Background

2.1 Description

The ease of access to digital technologies has been rapidly transforming the e-commerce industry for retailers and consumers. With raising year-on-year sales coming from the e-platform, retailers are all jumping into the bandwagon of improving digital-buying experience for consumers. Then in 2019, boon or bane, the COVID19 pandemic became a catalyst that spurred online sale more than ever before.

The impetus above motivated our project team to leverage on AI technologies to give e-commerce industry the next leap in improving their business processes and consumer experience, in particular, we had our eyes set on the fashion industry. We proposed three ideas in which the fashion industry could be transformed with the help of AI technologies.

2.2 Objectives

Our team aimed to create a MVP to explore and demonstrate the potential and viability of the ideas below. We had also scoped our problem to only fashion tops for this project.

Idea1: Generating Fashion Articles for Design Inspirations. Before a fashion article can go into production, there would be a designing phase. In this track, we hope to create a model that can auto-generate different fashion article designs, thereby jumpstarting the design process with the design inspirations generated. We scope our task to generating three clothing styles: (i) women's tops, (ii) women's dress, and (iii) men's tops.

Idea2: Swapping Model Clothing Style for Product Marketing. Marketing and advertising are big components in the fashion industry. These typically consisted of posters displaying human models in different clothing styles. We hope to aid in this aspect by auto-swapping the clothing style of a model so that a single

photoshoot could serve the purpose of advertising for multiple clothing styles. For project purpose, we scope our tool to swap between long-sleeve and short-sleeve fashion styles.

Idea3: Virtual Try-on for Consumers. Since early 2020, the world has been presented with an unprecedented situation with the rapid spread of the COVID-19 virus. At this time, shopping for clothes has been identified as one of the high-risk activities given that several different people physically try on clothes in shops rendering them vulnerable to the virus. A recent survey concluded that 65% of women and 54% of men are not comfortable using a dressing room right now[2]. Therefore, we believe it would be very timely and mutually beneficial for both consumers as well as fashion companies to provide a Virtual Try On solution which allows consumers to try on clothing virtually in the safety of their own homes. To add further impetus to this, with e-commerce booming in the field of fashion retail, this is a well sought after tool that will enable people to comfortably order fashion apparel online from different companies across different parts of the world without concerns on whether it will look good on them. At the same time, companies can also benefit financially by avoiding returns, re-shipping, inventory costs and so on. For the purpose of this project, we limit the scope of our tool to enable virtual try-on of women's tops only.

To demonstrate our ideas, we had encompassed the three tools into an easy to use web app. For fashion article generations, we intend for user to only select a style and clicked on a button to generate different designs. For fashion swapping, user need only to upload a human model in a particular style, and with a button click, transformed the model into another style. For virtual try-on, user can easily upload a photo of themselves in apparel A, a photo of apparel A standalone. For a photo of apparel B standalone, user have the option to upload one or pick one from our fashion article generator. With a click of a button, their current clothing is instantly swapped into apparel B.

[2] https://www.huffpost.com/entry/is-it-safe-try-on-clothes-stores-coronavirus_l_5f0484dac5b6e97b568a6c59

3 System Design

3.1 Overall Design

The overall system design is depicted as a flowchart in the Figure 1. A user-friendly frontend was created to solicit user inputs and display the generated images. The Article Generator module and Virtual Try-on module was a cooperative system. Specifically, user could make use to the Article Generator to create a clothing design and try on the newly created design.

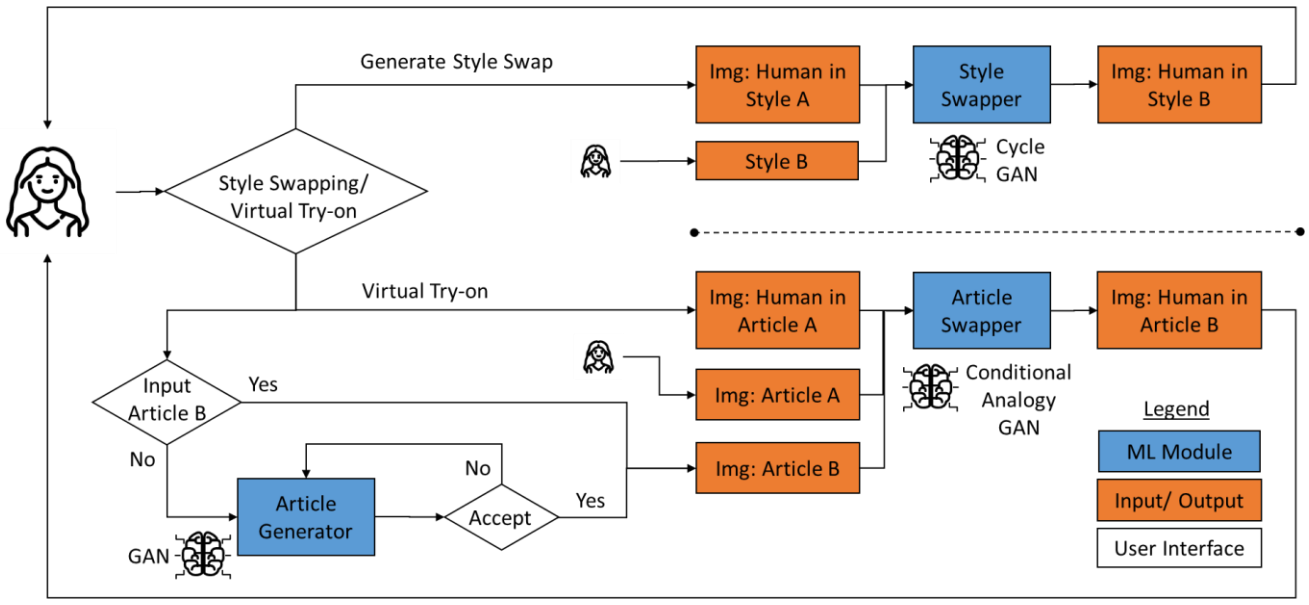


Figure 1: Overall System Design

3.2 System Implementation

The ML models, specifically deep learning models of GAN, Cycle GAN, and CAGAN, were modelled and trained using Keras-Tensorflow or Pytorch. The final models were integrated into our webapp backend which was implemented using python. The webapp frontend was implemented using HTML, CSS, and Javascript. The end-to-end system architecture can be seen from Figure 3. Within the modelling block,

each module (Article Generator, Style Swapper, Virtual Try-on) goes through the same process as depicted below.

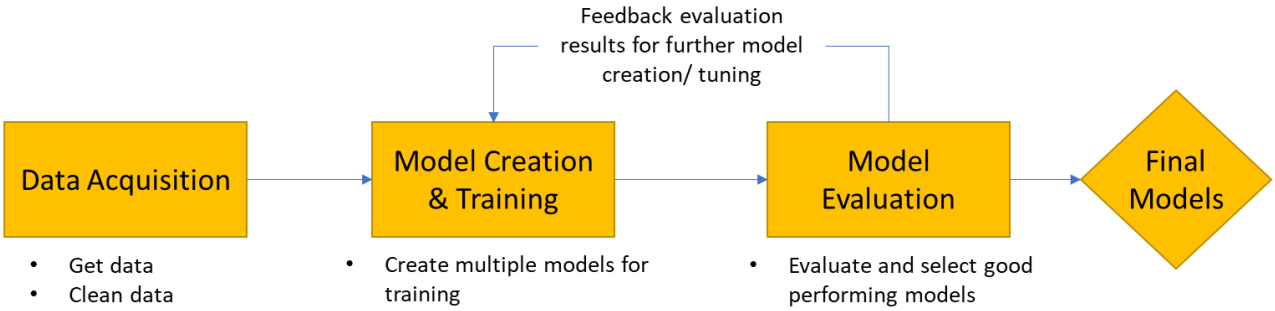


Figure 2: Model Creation Process

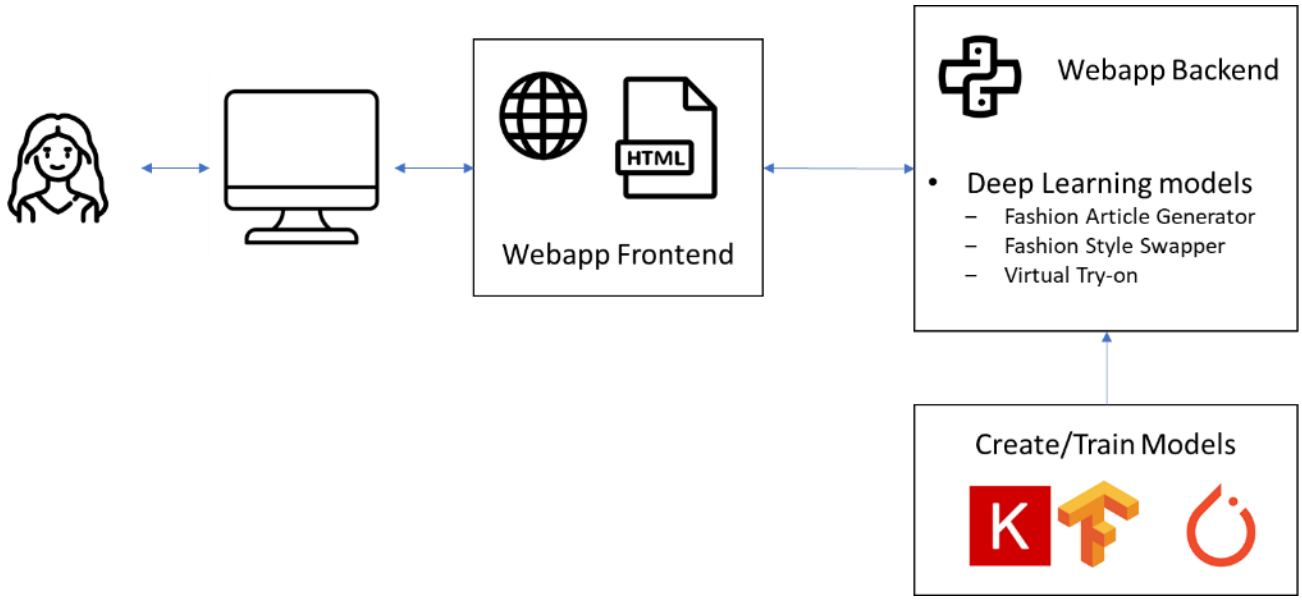


Figure 3: End-to-end System Architecture

4 The Dataset

Our dataset came primarily from three sources: (1) images scrapped from e-commerce sites[3], (2) images from existing open-source projects[4] of similar theme, and (3) publicly available images curated for hackathons[5].

For track 1 (generating clothing articles), the images required comprised purely of clothing articles and no models. These were readily available from source 1 and 2. For track 2 (changing model clothing style), we required images of models wearing clothing articles, these images also had to be labelled into style A (long-sleeve) and style B (shirt). We found images from source 3 to be most suitable, as images there were already arranged into folders of a particular fashion style. For track 3 (virtual try-on), we required images of models wearing clothing articles paired with corresponding standalone images of the clothing article. Images from source 2 was ideal for our requirement.

Figure 4: Example of one training pattern for
Fashion Article Generator, Style Swapper, and Virtual Try-on respectively

From these raw images, each had to be preprocessed and cleaned in specific manner for the three tracks. Details to these steps were documents in the respective sections.

[3] A python script was created to scrapped images from https://www.zalando.co.uk/. The dataset is stored in https://drive.google.com/drive/folders/1vwTmzV10aKtpRhYdWrLArkFtl73TWb3y
[4] https://drive.google.com/file/d/1MxCUvKxejnwWnoZ-KoCyMCXo3TLhRuTo/view
[5] https://www.kaggle.com/dqmonn/zalando-store-crawl

5 Generative Adversarial Network

Generative Adversarial Network (GAN) model was the base Deep Learning model used in all three tracks. For the style-swapper and virtual try-on, the GAN model was adapted into Cycle GAN and Conditional Analogy GAN, details for these two models can be found in sections **Error! Reference source not found.** and 8.1 respectively. Below is an overview of the concept of GAN model that cuts across all three tracks.

In GAN (vanilla or adapted forms) two classes of networks, a discriminator network and a generator network, competes with each other in a zero-sum game. The discriminator learns to discern a generated data from a real data, while the generator learns to generate data that is undiscernible by the discriminator. At equilibrium, we achieve a generator network that generates data similar to the (real) input data.

6 Fashion Article Generator

6.1 Algorithm – GAN

The vanilla GAN is a generative model that learns the distribution of input data and generate an output data that has similar distribution.

The trait to generate data similar to input data means we can supply the network with real images of fashion articles for distribution learning. After which, by supplying the generator with a vector of random variables, an output image similar to the real images could be created. This was a very appealing property when we wish to create new fashion article designs.

6.2 Methodology

6.2.1 Dataset Preparation

We had set out to create fashion articles of three different styles; women's tops, women's dress, and men's tops. For that we needed to provide the network frontal

standalone images of each style of apparels. For women's tops, there exists data from open-source projects. This dataset was mostly clean, but some unwanted images still existed (e.g. non-frontal views, zoomed-in photos, multiple apparels in one image). They were manually identified and removed.

Women's dress and men's tops had to be scrapped from e-commerce sites, a python script was written to do this task. Being raw images from the internet, the cleaning process was significantly more time-consuming and tedious. Table 1 listed the image defects and methods used to resolve them.

Defects	Methods to Resolve
Images of human models with apparels	A pretrained face detection model was used to sieved out such images. Photos showing partial facial features had to be manually removal
Zoomed-in images on fashion apparels	Manually identified and removed
Grey/ off-white background on images	OpenCV Grabcut was used for foreground extraction. When unsuccessful, photo editing tools were used to ensure a clean white background

Table 1: Defects on Images

Ultimately, 738 images of women's tops were removed, leaving 13,483 images. 2,743 images of women's dress were removed, leaving 2967 images. 471 images of men's tops were removed, leaving 4053 images. Refer to Figure 20 of the Annex for sample of input images used. Remaining images were then resized to 128 rows by 96 columns.

6.2.2 GAN Models

Several variants of the GAN model were explored, mainly the depth of the generator network and dimension of the generator's latent input were varied. As the amount of data for women's dress and men's tops were much lesser, GAN models with fewer parameters (i.e. deeper conv transpose layers) were explored for these two fashion styles to avoid overfitting the model. A particular GAN structure is shown in Figure 22 of the Annex.

The generator network uses the DCGAN structure. Leakly Relu was used for all activations, except for the output layer of the discriminator which used sigmoid, and output of the generator which uses tanh.

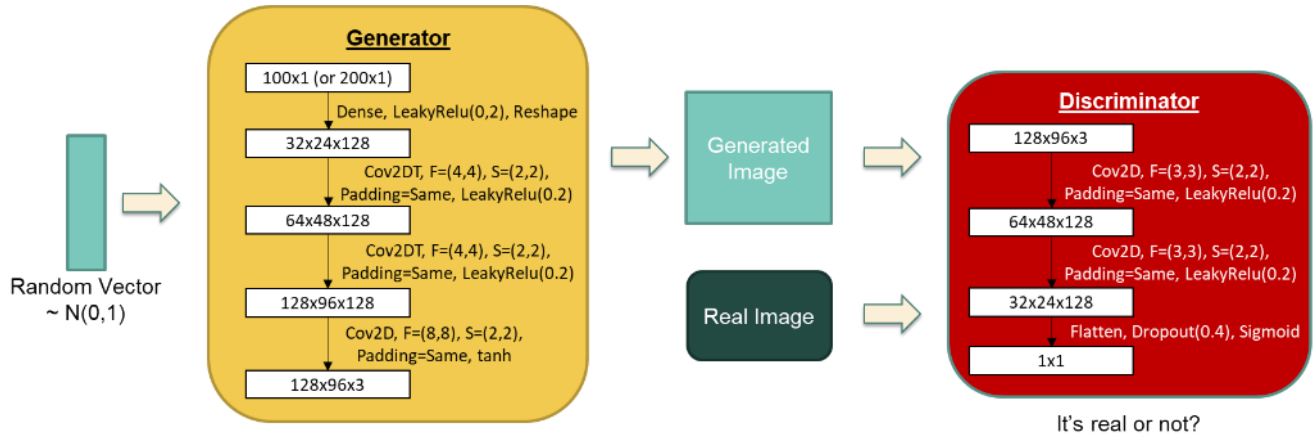


Figure 5: Generative Adversarial Network (GAN)

The GAN loss is optimized in a minimax fashion, where the discriminator maximises the loss while the generator minimises it. Implementation wise, to train the discriminator, the generator was made untrainable, then to the discriminator we supplied a half-minibatch of generated images (labelled as false) and another half-minibatch of real images (labelled as true). Next to train the generator, the discriminator was made untrainable, we supply the network a minibatch of random vectors (labelled as true).

Adam optimizer was used with learning rate of 0.0002. Binary cross-entropy loss function was used for the discriminator. All models were trained for at least 500 epochs on Google Colab.

6.2.3 Model Evaluation

GAN models did not have a "correct" label/output to compare with, hence we devised two methods to evaluate the generated images.

(1) Manual inspection. We generated 50 images for each model and through manual inspection, lookout for (i) mis-shaped clothing, (ii) diversity in design, and (iii) general correctness of the generated image.

(2) Comparing against pretrained classification model. VGG16 pretrained on ImageNet was used. Of the 1k classes in ImageNet, we manually identified those[6] that resembled or was associated to apparels. If a generated image was classified into one of these classes, we considered that a successful image generation.

Method 2 was used to narrow down the good performing models to a handful, then method 1 was used to ascertain that the models were generating correct images.

6.3 Results

6.3.1 Asymmetry of Initial Generated Images

Initial images generated presented a big problem. Figure 6 shows some initial images generated, we could see that asymmetry was present in most of the images generated.

Figure 6: Asymmetrical Generated Images (asymmetry in red circles)

We explored an idea to overcome this by adding a customised loss function, L_{sys}, to the original loss of GAN models.

$$\mathcal{L}_{sys} = \frac{1}{2}\mathrm{E}\|G_L - \mathrm{Flip}(G_R)\|$$

Suppose our generated image had 128 rows and 96 columns. G_L represented the first 48 columns of pixels, and G_R represented the last 48 columns. The *Flip* function flipped the image wrt to the vertical axis so that columns i and (n-i) were swapped, where n=no. of columns (n=96 in our example). The norm used was absolute-

[6] Classes identified are: academic_gown, apron, bikini, brassiere, breastplate, bulletproof_vest, cardigan, Cardigan, chain_mail, cloak, cuirass, fur_coat, hoopskirt, gown, jersey, kimono, lab_coat, military_uniform, miniskirt, overskirt, pajama, poncho, sarong, suit, sweatshirt, vestment, trench_coat.

value norm. As we can see, L_{sys} ensured the left-hand and right-hand side of the images are roughly the same. With this our GAN model (discriminator and generator combined) is as shown in Figure 7 and the generator loss function becomes

$$Loss = \lambda_1 \mathcal{L}_{sys} + \lambda_2 \mathcal{L}_{gan}$$

From experimentation, $\lambda_1 = 0.1$ and $\lambda_2 = 0.9$ achieved the most balanced results where there was symmetry in most of the generated images, but not to the extent of being mere mirror reflection wrt to the vertical axis.

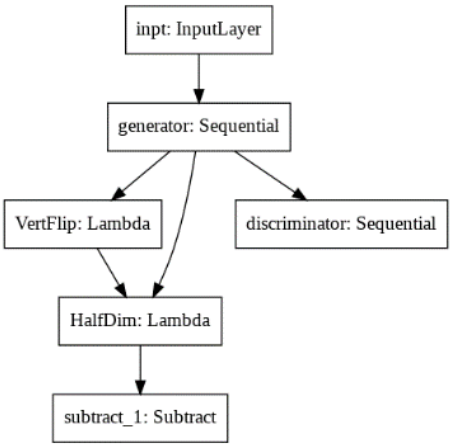


Figure 7: GAN model with customised loss. The model becomes a multi-output model, where the left branch was for L_{sys} and right branch was for the usual L_{gan}.

6.3.2 Results for Women's Tops, Women's Dress, and Men's Tops

The accuracy wrt to VGG16 is shown in Figure 9. At least 50% of our generated images could be recognized as apparels by VGG16 for women's and men's tops. This was around 17% for women's dress, it could be attributed to fewer number of classes within ImageNet that were associated to dresses. The top performing model for each style was chosen to generate 50 images each, these were than manually inspected for correctness and diversity in designs. Loss curve for each of the models can be found in Figure 21 of the Annex.

Figure 8: Sample of Final Generated Images for each style.
Top row: women's tops, middle row: women's dress, bottom row: men's tops

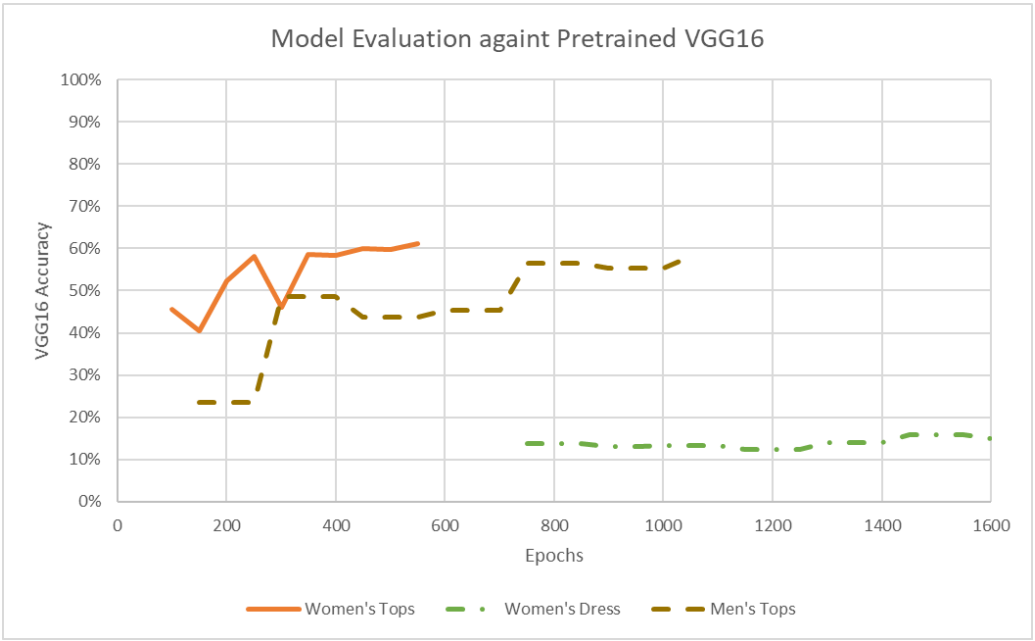


Figure 9: Model Evaluation against Pretrained VGG16. Note that not all epochs had a corresponding data point. Lower epochs were deemed to be poor performing, hence not passed into VGG16

6.4 Limitations and Improvements

1. Better generation of design prints on fashion articles. We acknowledge that designs/patterns printed on the clothes were not well captured by the GAN model. This was something that could be explored further, e.g. exploring on StackGAN which can generate photo-realistic images, thereby capturing the patterns on a fashion article more accurately.
2. Expanding to other types of fashion articles. For the project we had focused on tops, but other fashion articles could also be explored, for example pants, shorts, and the more difficult types such as footwear or headwear.

7 Virtual Try-on

7.1 Algorithm – Conditional Analogy GAN

Conditional Analogy GAN (CAGAN) is a special type of GAN that aimed to solve the image analogy problem using the traditional GAN architecture. In addition to generating new images never seen in the training dataset, the CAGAN architecture also understood the relation between paired images in the training dataset and ensured that the new images corresponded to the same relation. For example: Given 3 images A and A' and B, CAGAN would generate an image B' that had the same relation with B as A' was related to A. Hence, this particular GAN model is referred to as an analogy GAN.

Next, we explain conditional aspect of CAGAN. Traditional GAN networks generated plausible images that followed the input data distribution, these are random images from the given domain thereby having the innate limitation that it was not possible to generate targeted images for a given domain. For ex: If a GAN model was trained on the MNIST-dataset or the CIFAR-10 dataset, we cannot expect the generator to generate images specific to the number 5 or specific to a ship, respectively. If we required the model to generate images for a specific class, we need to train the model with images along with their class labels, also referred to

as 'conditioning information'. This type of model is referred to as Conditional-GAN (cGAN) models.

In our project, we make use of cGAN models to solve the image analogy problem thereby explaining the name, Conditional Analogy GAN. In this project, we used the CAGAN model to implement the Virtual Try On feature i.e., a consumer could visualize how a new clothing top would look on him/her by using our system. The following sections covers the exact model architecture, loss functions, results and limitations.

7.2 Methodology

7.2.1 The Dataset

For the Virtual Try On section, we scoped our problem to women's tops. As per the requirements of our CAGAN model, we require 3 different input images –

1. Image of a human wearing a clothing top A,
2. Standalone image of the clothing top A without a human,
3. Standalone image of the clothing top B without a human that the user wanted to purchase which would be used as the 'conditioning information'.

To address these requirements, we were able to identify an open-source dataset that comprised ~14k pairs of women tops and models wearing the respective tops. Due to resource and time limitations, for CAGAN we randomly selected ~5k samples from this dataset for training. We also considered using our self-curated dataset extracted using the Python web crawler for Virtual Try On. However, since we were specifically focused on women tops only, we believed the available open-source dataset with purely women tops was already cleaned, pre-processed and well maintained for direct usage and hence we relied only on this dataset for Virtual Try On using CAGAN.

7.2.2 CAGAN Model

Since CAGAN followed the same architecture as a traditional GAN network explained earlier, it contained a generator as well as discriminator. The primary objective of the generator and the discriminator had been covered in detail under section 5 Generative Adversarial Network. In this section we would talk about the exact architecture of the generator as well as the discriminator models. After carrying out extensive literature review, we eventually followed the architecture and implementation as described in a conference paper titled – The Conditional Analogy GAN: Swapping Fashion Articles on People Images[7].

7.2.2.1 Generator Model

As per this implementation, the generator followed an encoder-decoder architecture with skip connections to perform image to image translation. An encoder essentially learnt to represent the input data (for ex: image) in the smallest possible space, extracting and representing the most important features in a manner that can be understood by the decoder. Subsequently, the decoder learnt to understand the encoded information and reconstructed an output following the input distribution. While a standard encoder-decoder model did not have skip connections, this particular model made use of skip connections since in any convolutional network, the earlier layers comprise of low-level feature maps while the later layers comprise of the high-level or more specialized feature maps. By using skip connections, we ensure that the low-level features were not lost and thereby enriching the overall features in the final output. An illustration of the generator model is provided below:

[7] https://arxiv.org/pdf/1709.04695.pdf

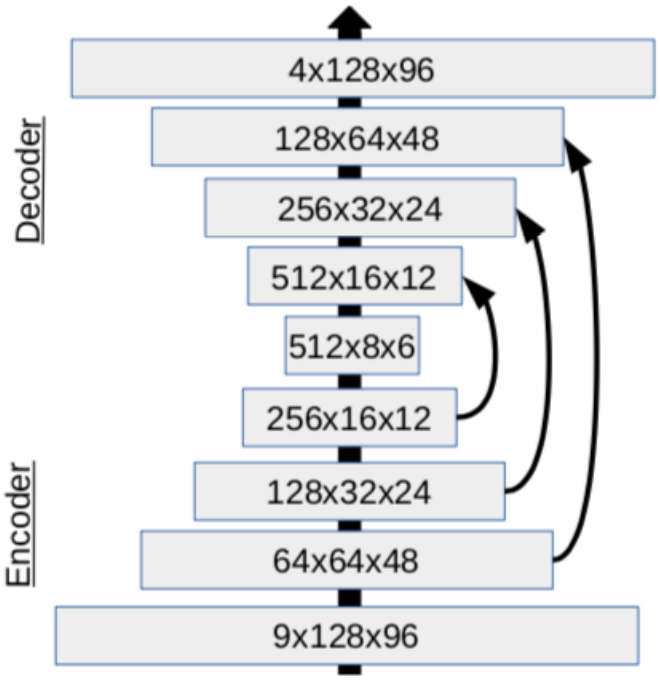


Figure 10: Encoder-decoder model with skip connections in the CAGAN generator

As we could see in the above image, the model took an input of 3 images (3 x 3 (RGB) = 9 channels), each image resized to 128x96 pixels. This was inline with what was explained earlier under 7.2.1. The Dataset – the user needed to upload 3 different images which would be concatenated and passed into the model. The generator model was trained using a loss function which would be explained later, to generate a 4-channel output, the first channel would be a segmentation mask α and the next 3 channels would be an RGB colour image. These 2 outputs would be combined to generate the final output shown below.

$$\alpha x' + (1-\alpha)x$$

x' = 3 channel (RGB) output image

x = Original image with human wearing their own clothing top

α = Segmentation mask (First channel of output)

The segmentation mask essentially segmented out the region occupied by the original clothing article on the human. Therefore, when the segmentation mask is multiplied with the x' it extracted only the region coloured with the new clothing article from the output image. However, to provide the complete final image including the human body parts, we multiply the inverse of the segmentation mask with x which extracted only the human body without the region occupied by the clothing article. When we summed these 2 terms, we got the final output image which was returned to the consumer. We would present more details in subsequent sections on how the model was trained using the loss function to swap the new clothing article. For all the layers, LeakyReLU activation function was used except when we generate the segmentation mask in the final output layer where we use the sigmoid function in order to suppress the output values between 0 and 1. An illustration of this is shown below:

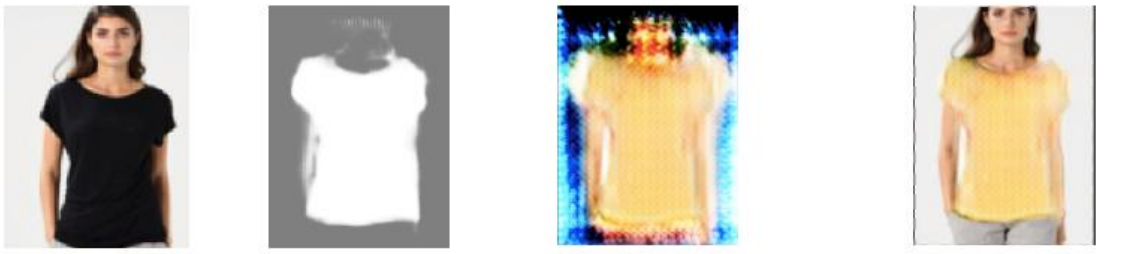


Figure 11: Illustration indicating the segmentation mask α, original image x, RGB Colour Image output x' and the final output image

7.2.2.2 Discriminator Model

For the discriminator, we developed a PatchGAN discriminator which performed classification only for small patches of the image and not for the entire image at once. An illustration of the discriminator model is provided below.

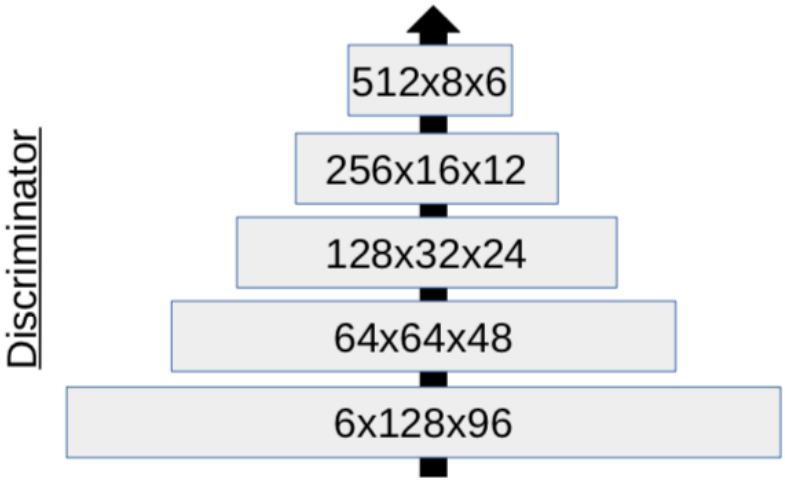


Figure 12: PatchGAN discriminator used in the CAGAN model

As seen in the above image, the discriminator model took in 2 images as input (2 x 3 (RGB) = 6 channels), each image resized to 128 x 96 pixels. In order to train the discriminator model, we passed 3 types of paired input images:

1. Positive sample – A human wearing a clothing article A and a standalone image of clothing article A,
2. Negative Sample I – A human wearing a clothing article A and a standalone image of clothing article B,
3. Negative Sample II – A human wearing a clothing article B and a standalone image of clothing article B.

The first type was a positive sample because these images were ground truth images obtained from the input dataset. The second type was a negative sample because the images were wrongly paired and did not represent the input distribution. The third type was a negative sample because the images were generated by the generator model and the discriminator aimed to identify that the image was a generated image and not a ground truth image.

As evident in the above illustration, in a PatchGAN discriminator, a filter was ran across the input images convolutionally across multiple layers. At each layer, we apply the LeakyReLU activation function except the final layer where we once again

we apply the sigmoid function to suppress the values between 0 and 1. This output was compared against the ground truth label which was a tensor of ones and zeros for the positive and negative samples respectively.

7.2.3 Loss Function

In this section, we explain the loss function which helped train the discriminator and the generator models. With regards to the generator model, this section would help us understand how the training helps in localizing the clothing article on the human being and repainting that section with the new clothing article. Alternatively, for the discriminator, we would understand how the model was able to discern between the negative and positive samples which were explained in the previous section.

First, we explain the overall loss function for the GAN model before looking into each individual term in detail:

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \gamma_i \mathcal{L}_{id}(G) + \gamma_c \mathcal{L}_{cyc}(G)$$

Equation 1: Overall loss function for the CAGAN model

The first term in the above loss function was the same as the minimax loss function in the traditional GAN architecture that the generator aimed to minimize and the discriminator aimed to maximize. Essentially, the generator aimed to minimize the probability of the discriminator classifying the newly generated images as fake. Contrarily, the discriminator aimed to maximize the probability of classifying newly generated images (Negative Sample II) and wrongly paired images (Negative Sample I) as fake while also maximizing the probability of classifying ground truth data as real.

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x_i,y_i\sim p_{\text{data}}} \sum_{\lambda,\mu} [\log D_{\lambda,\mu}(x_i,y_i)]$$
$$+\mathbb{E}_{x_i,y_i,y_j\sim p_{\text{data}}} \sum_{\lambda,\mu} [(1-\log D_{\lambda,\mu}(G(x_i,y_i,y_j),y_j))]$$
$$+\mathbb{E}_{x_i,y_{j\neq i}\sim p_{\text{data}}} \sum_{\lambda,\mu} [(1-\log D_{\lambda,\mu}(x_i,y_j))]\,,$$

Figure 13: Minimax loss function for the Generator and Discriminator

Typically, we observed in GAN models that the generator loss would comprise only of the second term in the above function because the generator had no control over the discriminator and aimed to minimise only the second term.

Apart from the minimax loss function, we also observed 2 more loss terms in the original CAGAN loss function displayed above in Equation 1. The first one is L_{id} which essentially ensured that the generator model did not paint parts of the human body apart from the area occupied by the clothing article. This is evident because this loss function aimed to reduce the area occupied by the segmentation mask thereby training the model to avoid segmenting parts of the human body and forcing it to segment out the clothing article only.

The next loss term is L_{cyc}, the cycle loss function tried to reduce the difference between the original image and the final reconstructed image. The final reconstructed image was obtained by feeding the same encoder-decoder generator module 3 more input images, this time the 3 inputs were (i) Final output of the generator, (ii) Standalone image of the clothing article that the user wants to purchase, (iii) Standalone image of the clothing article that user was currently wearing. This step was the reverse of the original step which enables the generator to reconstruct the original image. Subsequently, the generator aims to minimize the cyclic loss obtained by comparing the reconstructed image and the original image. These 2 additional loss terms are included as part of the generator training.

7.3 Results

Figure 14: CAGAN results obtained from the test dataset. Top row: original image of the user. Second row: standalone image of the clothing article the user was wearing. Third row: clothing that the user wanted to purchase. Last row: final image with the user in their desired clothing article

The top 3 images form the 9 channel input to the CAGAN model. Alternately, we can also test the CAGAN model via the web application that we have developed. A sample screenshot is shown below.

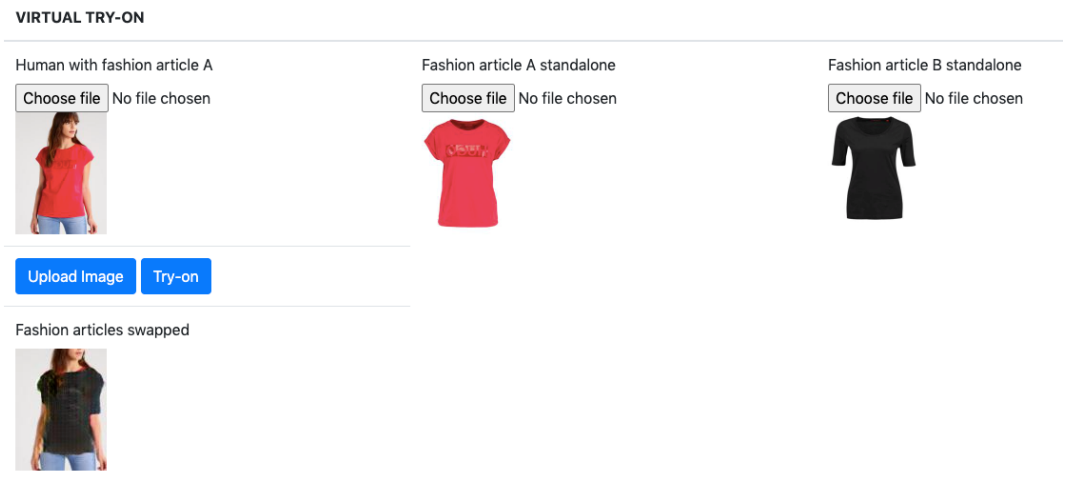


Figure 15: CAGAN in action in web application

7.4 Model Evaluation

As mentioned under the earlier GAN architecture, for the CAGAN model as well, we did not have any benchmark label to compare the model against since the quality of the output was largely subjective. We aimed to measure fairly abstract ideas in the output such as (i) Is the clothing article swapped at the right location, (ii) Is the human image looking normal or distorted, (iii) Are the designs of the clothing article clearly painted, etc. Given these limitations, we stuck to manual inspection to verify whether the final image satisfied our requirements. Furthermore, we verified the model training by plotting loss curves to understand if the loss was increasing/decreasing as expected.

7.5 Limitations

1. One of the most significant limitations was that the model was not invariant to poses. We observed specifically with women models that there were several different poses and often the target clothing articles did not fit accurately with the pose resulting in partially distorted images.
2. Distorted human body parts – Since the segmentation mask occasionally segmented out locations belonging to the human body parts, these parts were also incorrectly reproduced in the final image.
3. Complex designs – The model often struggled to understand the intricate floral or checked designs that were found in women clothing articles thereby not painting the final image accurately with the correct design. This could be observed in the right-most result shown in Figure 14.
4. Due to lack of resources such as GPU and in the interest of time, we were able to train the CAGAN model only for 150 epochs and we strongly believe that with more training, we would be able to solve some of the above mentioned limitations to a certain extent.

8 Fashion Style Swapper

8.1 Algorithm – Cycle GAN

The Cycle GAN was based on the normal GAN architecture. It aimed to learn to translate an image from a source domain X to a target domain Y. It contained two generators (G and F) and two discriminators (D_x and D_y). For generator G, it aimed to learn a mapping: X -> Y and the discriminator D_y was trying to find out whether it is from domain Y. But because this mapping did not have many restrictions, so there may exist a situation where all input images map to the same output image. To solve this problem, the Cycle GAN also used another generator F to enhance the constraint. First, the generator F was to learn a mapping: Y -> X and the discriminator D_x was going to clarify whether it was from domain X or not. Then it introduced a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa).

In our project, we were trying to use CycleGAN to do image style transfer. Based on the existing dataset, we are going to transfer image styles between long-sleeve domain and shirt domain.

8.2 Methodology

8.2.1 The Dataset

The dataset was provided by Zalando. There are 4680 images with shirt clothes and 1700 images with long-sleeve clothes.

8.2.2 Cycle GAN Models

For the architecture of generator, it contained two stride-2 convolutions, several residual blocks, and two fractionally-strided convolutions with stride ½. It also used instance normalization. Let c7s1-k denote a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1. Dk denote a 3×3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. Rk denoted a residual block that contained two 3×3 convolutional layers with the

same number of filters on both layer. uk denote a 3 × 3 fractional-strided-Convolution-InstanceNorm-ReLU layer with k filters and stride ½. The architecture of generator shows below:

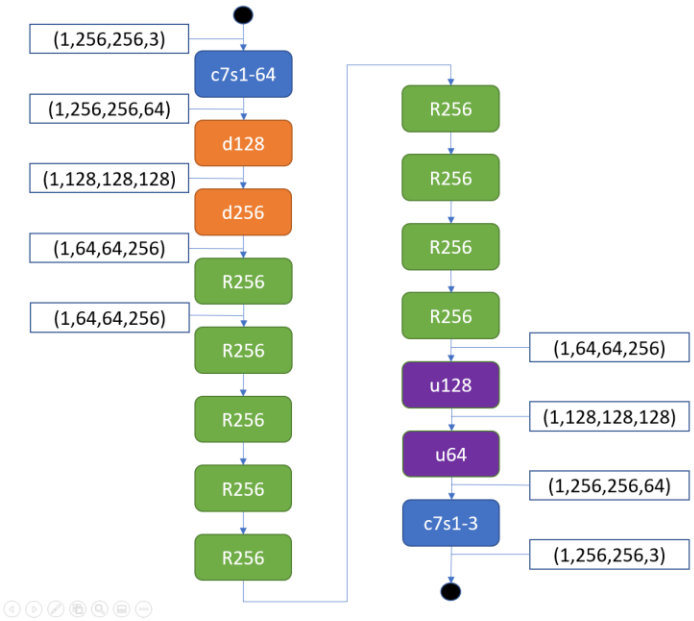


Figure 16: Architecture of generator

For the architecture of discriminator, it contained 70×70 PatchGANs. Let Ck denote a 4×4 Convolution-InstanceNorm_LeakyReLU layer with filters and stride 2. After the last layer, we apply a convolution to produce a 1-dimensional output. We did not use InstanceNorm for the first C64 layer. We used leaky ReLUs with a slope of 0.2. The architecture of discriminator shows below:

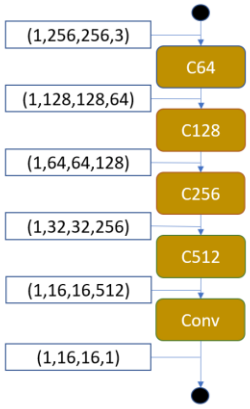


Figure 17: Architecture of discriminator

8.2.3 Objective

The full objective was:

$$L(G,F,D_X,D_Y) = L_{GAN}(G,D_Y,X,Y) + L_{GAN}(F,D_X,Y,X) + \lambda L_{cyc}(G,F)$$

where the G and F represented two different generators. D_X and D_Y represented two different discriminators. $L_{GAN}(G,D_Y,X,Y)$ and $L_{GAN}(F,D_X,Y,X)$ were similar to those of normal GAN. The third term of equation above called cycle consistency loss, designed to guarantee that the map in generator is individual for each input. In addition, λ controls the relative importance of the two objectives.

Here are details in cycle consistency loss:

$$L_{cyc}(G,F) = \mathrm{E}_{x\sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathrm{E}_{y\sim p_{data}(y)}[\|G(F(y)) - y\|_1]$$

where 1 means we used L1 norm. The equation above means: for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \to G(x) \to F(G(x)) \approx x$.Similarly, for each image y from domain Y, G and F should also satisfy: $y \to F(x) \to G(F(y)) \approx y$

8.3 Results

8.3.1 From long-sleeve to shirt

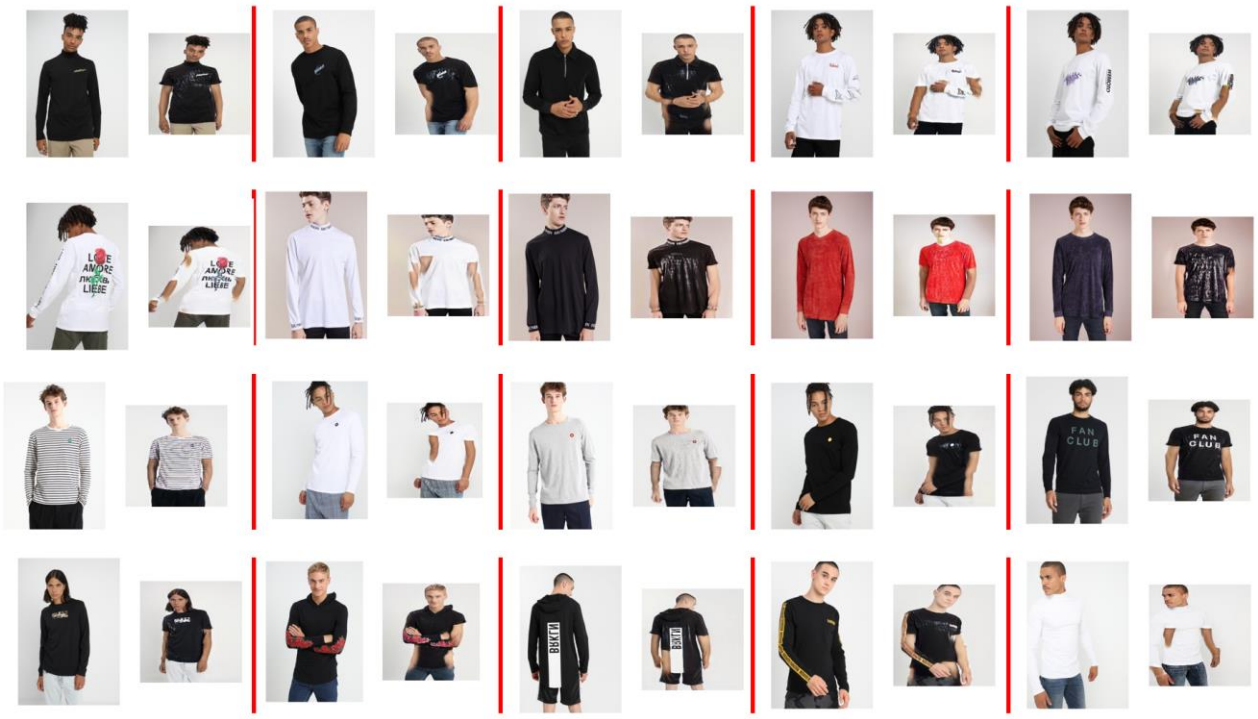


Figure 18: Results Long-sleeve to Shirt

8.3.2 From shirt to long-sleeve

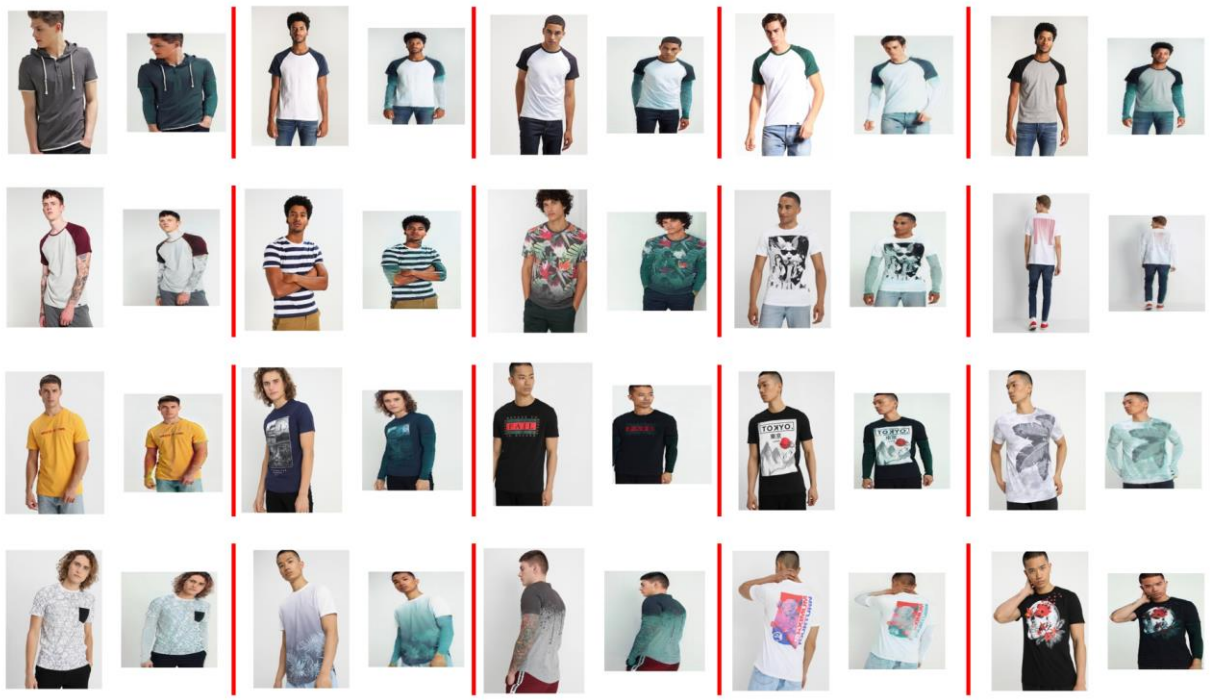


Figure 19: Results Shirt to Long-sleeve

8.3.3 Limitations

Due to the lack of diversity in dataset and limited computational resource during training time. The model at times could not reproduce the position of arms in the correct position, resulting in occasional cases where skin appeared at a odd position. In addition, during the transformation, the patterns in clothes and other details in the image were slightly modified. Complex poses of arm often resulted in styles not getting swapped successfully.

9 Conclusion

Through this exercise, we had put together a suite of tools that could help in the fashion industry and demonstrated its potential through a simple webapp. Albeit the results were not the most polished, it was able to show that the ideas had potential if more resource and exploration could be committed.

The Fashion Article Generator could help with design inspiration. The Virtual Try-on module allowed consumers to try on clothing in the safety of their own homes, thereby boosting consumer satisfaction. It could also work hand-in-hand with the Fashion Article Generator by allowing consumers to try on newly created clothes.

The Style Swapper allowed retailer to save on advertising/marketing cost by reusing a single photoshoot for multiple fashion styles.

In this project we had set for ourselves challenging goals and attempted something out of our comfort zone (i.e. something entirely different from MLP, CNN). We spent much time researching, implementing, and finally deploying our model. Through it, we had enriched our knowledge in complex deep learning models such as GAN, Cycle GAN, and CAGAN.

Annex

1 Fashion Article Generator

1.1 Sample Input Images

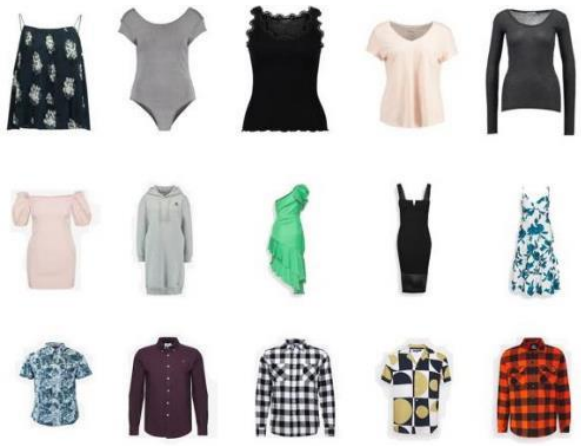
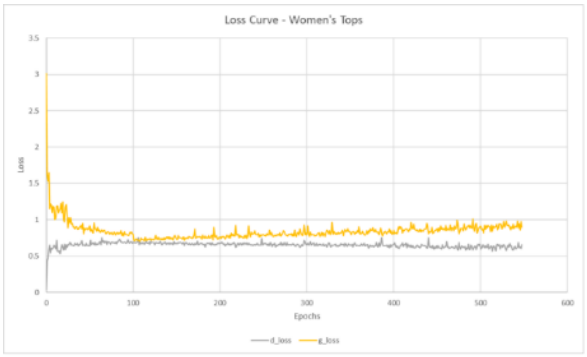
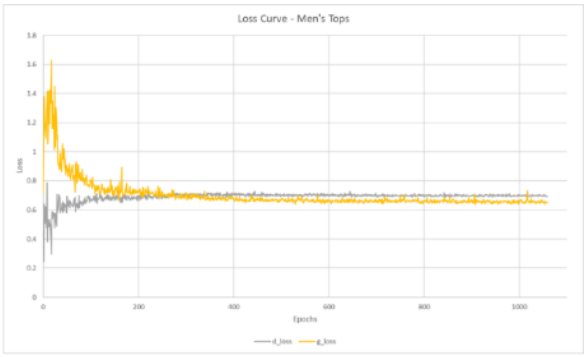


Figure 20: Sample Input Images for GAN Model. Top row: women's tops, middle row: women's dress, bottom row: men's tops

1.2 Loss Curves





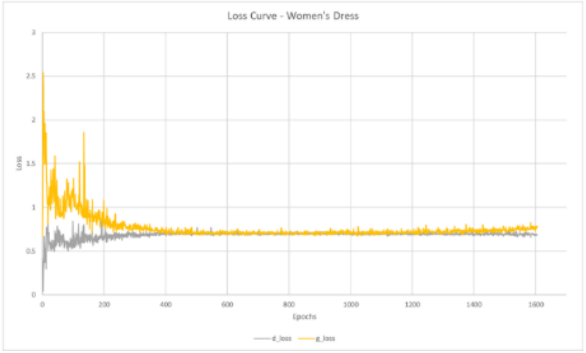


Figure 21: Loss Curves of GAN Models.
Top left: Women's Tops. Top Left: Men's Tops, Bottom: Women's Dress

1.3 GAN Model Structure

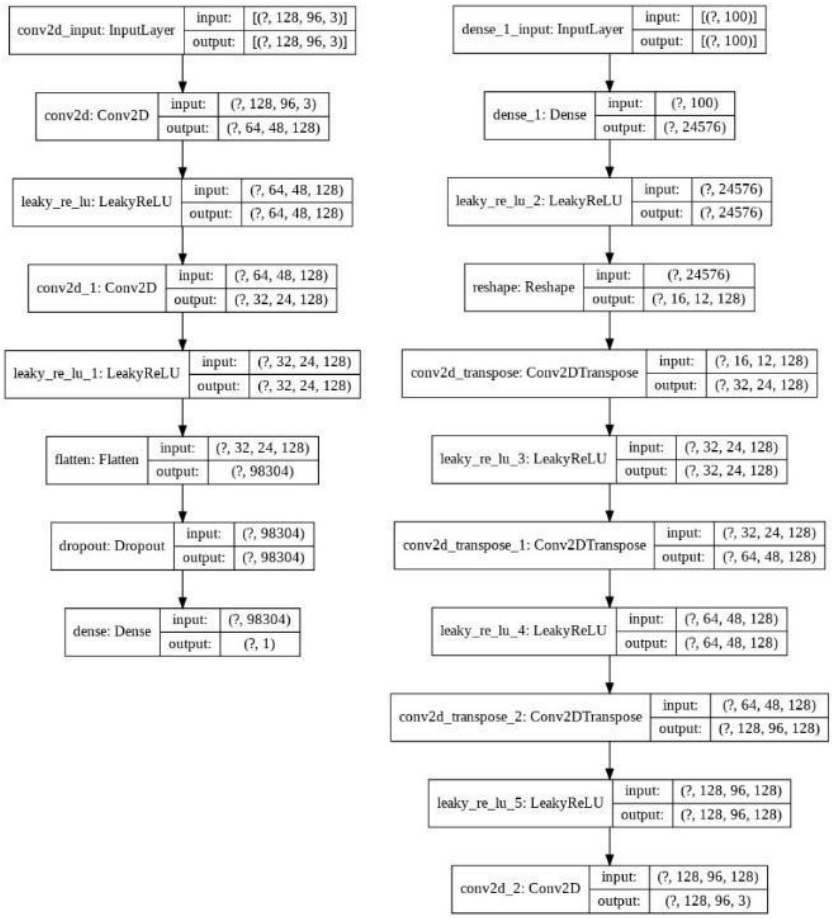


Figure 22: GAN Structure. Discriminator network on the left, Generator network on the right

2 Installation and User Guide

2.1 Starting the Webapp Backend on Heroku Cloud Services

1. You can easily navigate to https://prpm-ay2021-fashionai.herokuapp.com/ to access the demo app.

2. Note that initial loading may be slow as we are using the free tier service on Heroku Cloud Services.

NOTE: The RAM available for Heroku's free tier service is very limited, and is easily exceeded if you had ran the deep learning models several times. Heroku will then display an "Application Error" message. The only solution is to wait for Heroku to restart the dyno (a container for the webapp), clearing the RAM. It does this periodically but there is no control on when the dyno restart will occur, however dynos are always restarted every 24 hours. In-lieu of this, consider running the demo app on LocalHost if you intend to run the DL models several times.

2.2 Starting the Webapp Backend on Local Machine

1. This installation instruction is written on a Windows 10 OS.
2. Clone the source code from the following GitHub link: https://github.com/chen-mingyie/FashionAI
3. Download and install Anaconda Navigator via the https://www.anaconda.com/products/individual
4. Open Anaconda Navigator and start the Anaconda Prompt. The command prompt should display a "(base)", this shows that you are using the base venv.

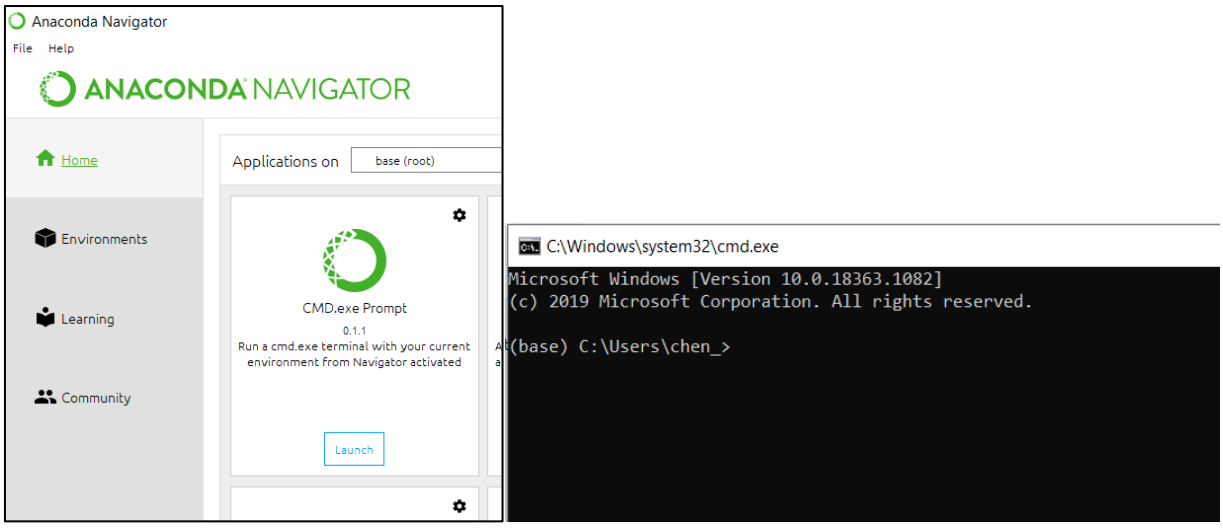


Figure 23: Anaconda Navigator on the left image. Anaconda Prompt on the Right image.

5. The demo requires the following packages, do a *pip install <package name>* of them to the base venv if they are not already installed:
   - *Flask==1.1.2*
   - *Flask-SQLAlchemy==2.4.4*
   - *gunicorn==20.0.4*
   - *Jinja2==2.11.2*
   - *PyMySQL==0.10.0*
   - *PyYAML==5.3.1*
   - *SQLAlchemy==1.3.19*
   - *psycopg2-binary==2.8.5*
   - *Pillow==7.2.0*
   - *Werkzeug==1.0.1*
   - *tensorflow==2.3.0*
   - *Keras==2.4.3*
6. Change directory to the location where you had placed the source codes. For this example, the command is: *cd C:\Users\chen_\PycharmProjects\FashionAI*
7. Start the webapp backend with command: *python fashionai.py flask start*. The webapp be started locally at http://127.0.0.1:8080
8. Navigate to your browser of choice (Firefox, Chrome, Edge, Internet Explorer) and enter the address above to go to the webapp.
9. To end the webapp, enter the command: *cntrl-c* into Anaconda Prompt.

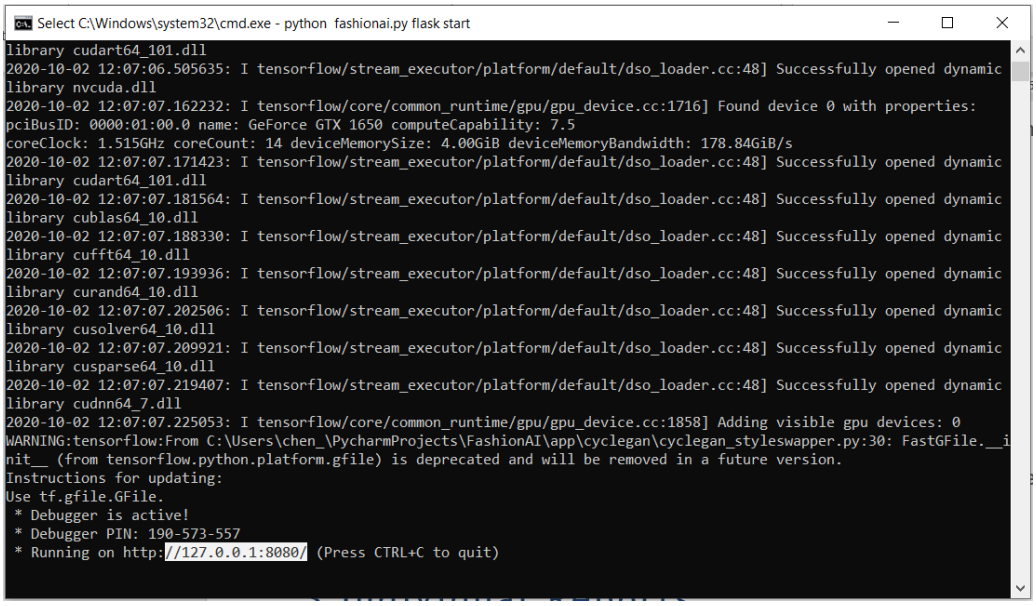


Figure 24: Webapp started locally at http://127.0.0.1:8080

2.3 User Guide on Demo Webapp

1. To generate fashion articles, select the style of choice from the dropdown and click on Generate button. This will generate 8 articles of the selected style.

Figure 25: Generating Fashion Articles

2. To do virtual try-on. Browse to an image of the model with fashion article A, fashion article A standalone, and fashion article B standalone (for article B, you can also save and use the generated images). Click on Upload button, then Try-on button.

Figure 26: Virtual Try-on of Fashion Articles

3. To do style swapping. First navigate to the Style Swapper on the top menu bar. Browse to an image of the model with style A, select style A, and select the target style (style B). Click on Upload button, then Swap Style button.

Figure 27: Swapping Fashion Styles

3 References

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. Generative adversarial nets. Advances in neural information processing systems. 2014: 2672-2680.
- Zhu, J. Y., Park, T., Isola, P., et al. Unpaired image-to-image translation using cycle-consistent adversarial networks. Proceedings of the IEEE international conference on computer vision. 2017: 2223-2232.
- Jetchev, N., Bergmann, U. The conditional analogy gan: Swapping fashion articles on people images. Proceedings of the IEEE International Conference on Computer Vision Workshops. 2017: 2287-2292.
- Sbai, O., Elhoseiny, M., Bordes, A., LeCun, Y., & Couprie, C. (2018). DeSIGN: Design Inspiration from Generative Networks. *Arxiv*. Retrieved from http://arxiv.org/abs/1804.00921v2
- Browniee, J. "How to Develop a GAN for Generating MNIST Handwritten Digits." *Machine Learning Mastery*, 28 Jun 2019, https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/
- Browniee, J. "How to Develop a Conditional GAN (cGAN) From Scratch." *Machine Learning Mastery*, 5 Jul 2019, https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
- Browniee, J. "How to Implement CycleGAN Models From Scratch With Keras." *Machine Learning Mastery*, 7 Aug 2019,

https://machinelearningmastery.com/how-to-develop-cyclegan-models-from-scratch-with-keras/

- Shaoan, L. "Cloth Swapping with Deep Learning: Implement Conditional Analogy GAN in Keras", 26 Oct 2017, https://shaoanlu.wordpress.com/2017/10/26/reimplement-conditional-anology-gan-in-keras/