COMP4423 Computer Vision

Group Project – Fashion Image Generation

Group 7

Individual Report

Student Name:	**SHEK Yun Sang**
Student ID:	**19037587D**

Group Members:

LAM Chun Pang	**19080189D**
YIM Man Tsun	**19064851D**

Introduction

In this project, we will explore the use of Generative Adversarial Network (GAN) models in the fashion industry, specifically for professional fashion designers. The assistance of an application that can automatically generate a plethora of fashion images would be invaluable where GAN models offer a promising solution to this challenge, as they can generate new images based on existing designs and styles.

Dataset Description

The fashion dataset utilized for the implementation and evaluation is provided in two .csv files containing grayscale images sized 28x28. The dataset is composed of a total of 70000 images, separated into a train set and a test set, consisting of 60000 images and 10000 images, respectively. To ensure the generalization of the Conditional GAN (C-GAN) used for generating images specific to a particular class, it is essential to investigate the distribution of the class labels. As shown in Figure 1, the dataset is balanced with each class containing an equal number of 6000 images, indicating an ideal balance, preventing the generator bias towards majority classes, and enabling it to learn distinguishing features for each category.

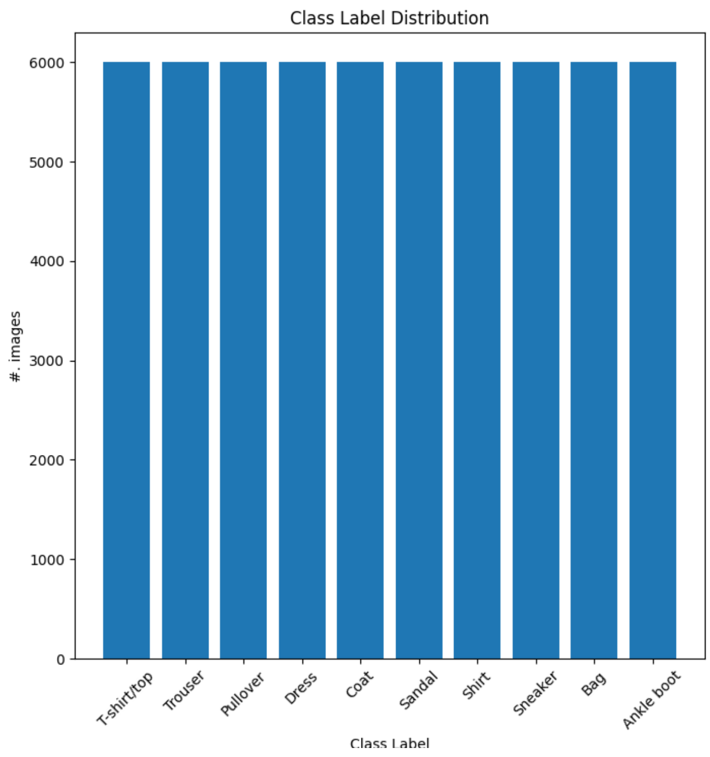


Figure 1: Class label distribution of dataset

Figure 2: Visualization of training samples

Data Preprocessing

As with other computer vision tasks, it is necessary to normalize the pixel values of images to the range of [-1, 1] before incorporating them into the training process of the GAN models to enhance stability and generalization. Normalization of the pixel values of real images for both the discriminator and generator can converge effectively. In addition, it can help to prevent mode collapse in the generator, a common issue in GAN training where the model produces only a limited set of images that are difficult for the discriminator to distinguish.

Model Architecture

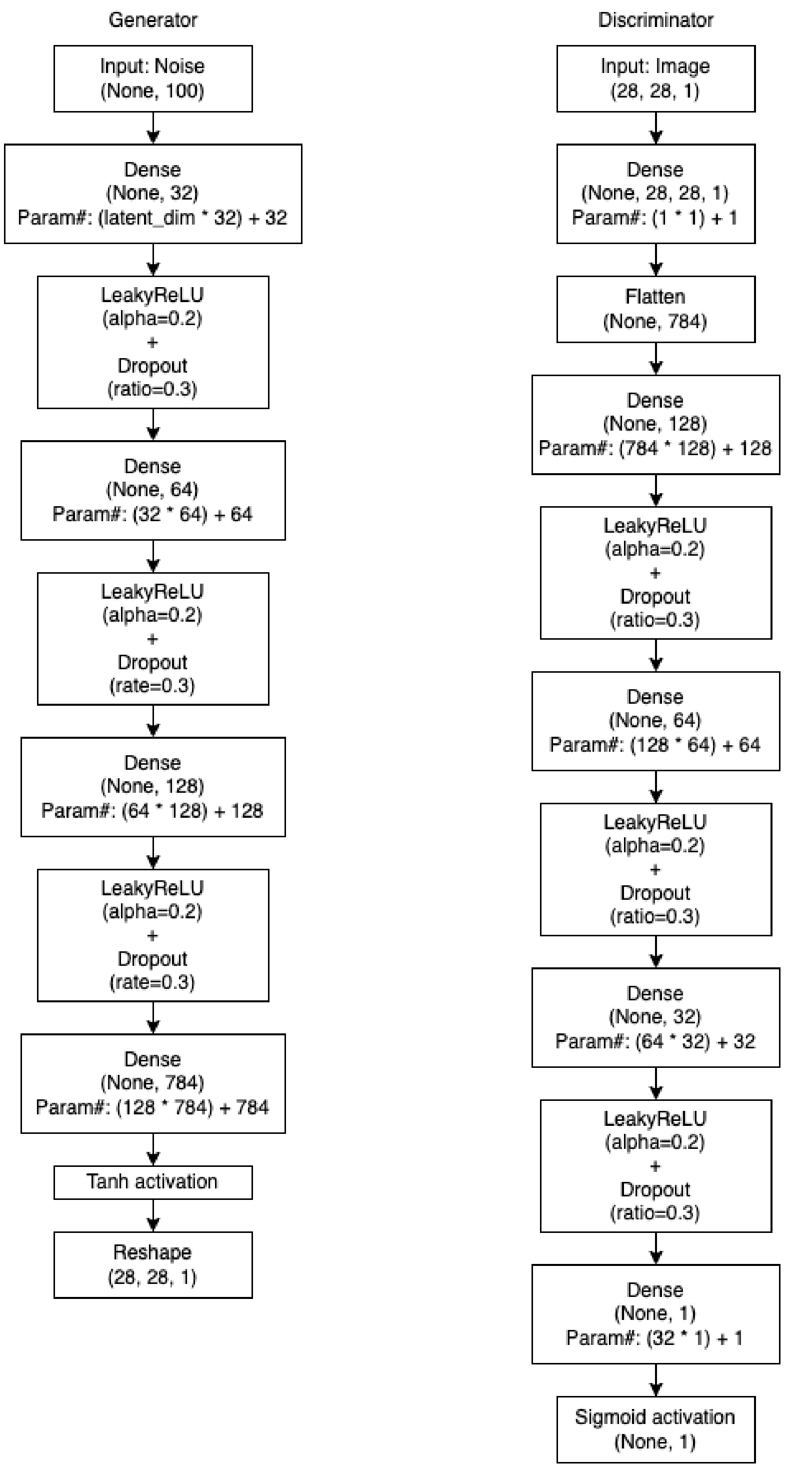
- <u>**Without Control label**</u>
  - **Baseline Model**

    The fully connected GAN model consists of two neural networks with the simplest structures among the implemented models.

    The generator of the baseline model is constructed by four fully connected layers and uses LeakyReLU activation with a slope of 0.2 for each layer. A dropout regularization with a rate of 0.3 is applied for each of the first three layers to prevent overfitting. The final layer has a Tanh activation to generate the image output and is reshaped for the discriminator input.

    The discriminator takes the input of the generated grayscale image size 28x28 from the generator. The discriminator flattens the input image in a 1D vector and then passes it to the following four fully connected layers. Each of the three dense layers in between the first input and last output layers is also utilized as a LeakyReLU activation with a slope of 0.2 and a dropout layer with a rate of 0.3. The final output layer uses a sigmoid activation to produce the probability, indicating whether the input image is real or fake.

The major limitation of the baseline model is that the model architecture is based on fully connected layers, which could not effectively capture the complex features and patterns in fashion images. It also lacks robustness to noise and potentially leads to overfitting that the model with only fully connected layers is limited to capturing spatial information from training images.



Generator
Input: Noise (None, 100)
Dense (None, 32) Param#: (latent_dim * 32) + 32
LeakyReLU (alpha=0.2) + Dropout (ratio=0.3)
Dense (None, 64) Param#: (32 * 64) + 64
LeakyReLU (alpha=0.2) + Dropout (rate=0.3)
Dense (None, 128) Param#: (64 * 128) + 128
LeakyReLU (alpha=0.2) + Dropout (rate=0.3)
Dense (None, 784) Param#: (128 * 784) + 784
Tanh activation
Reshape (28, 28, 1)
Discriminator
Input: Image (28, 28, 1)
Dense (None, 28, 28, 1) Param#: (1 * 1) + 1
Flatten (None, 784)
Dense (None, 128) Param#: (784 * 128) + 128
LeakyReLU (alpha=0.2) + Dropout (ratio=0.3)
Dense (None, 64) Param#: (128 * 64) + 64
LeakyReLU (alpha=0.2) + Dropout (ratio=0.3)
Dense (None, 32) Param#: (64 * 32) + 32
LeakyReLU (alpha=0.2) + Dropout (ratio=0.3)
Dense (None, 1) Param#: (32 * 1) + 1
Sigmoid activation (None, 1)

Figure 3: Model architecture of the baseline model

- **Suggested Model 1: Deep Convolutional GAN (DCGAN)**

  This implementation of DCGAN followed the tutorial of GAN model in TensorFlow [1].

  The generator begins by taking latent input and passing it through a dense layer with a LeakyReLU activation and batch normalization. It then proceeds to two series of Conv2DTranspose layers, LeakyReLU activations, and batch normalization. The final Conv2DTranspose layer generates the fake image with a size of 28x28.

  The discriminator is implemented with two sets of a convolutional layer, LeakyReLU activation and dropout layer. The output of these layers is flattened and fed to a dense layer to produce the probability.

  The DCGAN replaces fully connected layers with convolutional layers, where the Conv2DTranspose layer is more appropriate for image data to up-sample the spatial dimensions of the input and generate new images. Batch normalization helps stabilize the training process by reducing internal covariate shifts that can prevent network convergence. It can also overcome a certain degree of limitation and improve the stability of the FCGAN model that captures limited spatial information.

  However, this model may still encounter the problem of mode collapse and suffer from vanishing gradients and instability that the model is sensitive to the selection of hyperparameters.

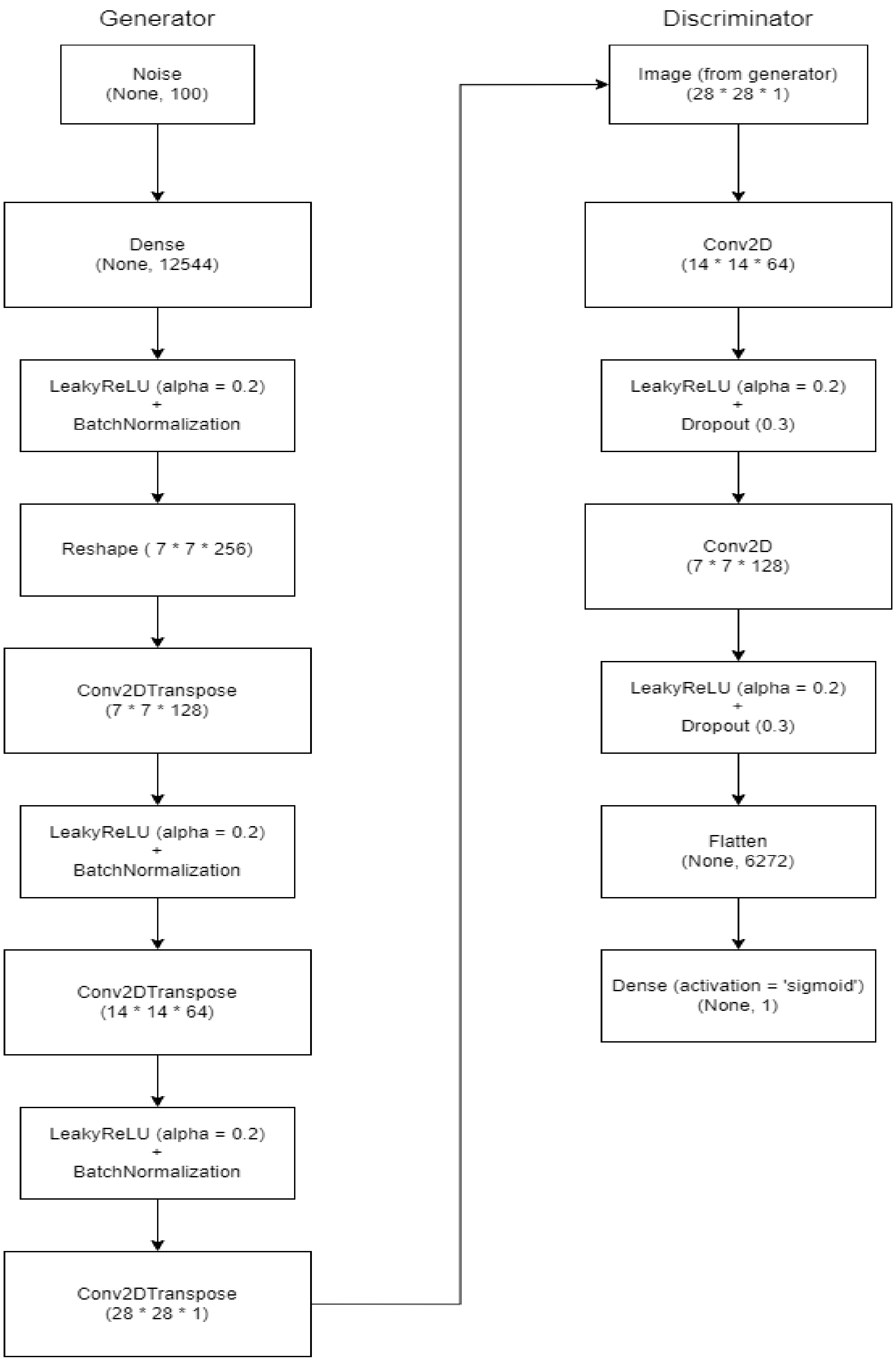

Generator
Noise (None, 100)
Dense (None, 12544)
LeakyReLU (alpha = 0.2) + BatchNormalization
Reshape (7 * 7 * 256)
Conv2DTranspose (7 * 7 * 128)
LeakyReLU (alpha = 0.2) + BatchNormalization
Conv2DTranspose (14 * 14 * 64)
LeakyReLU (alpha = 0.2) + BatchNormalization
Conv2DTranspose (28 * 28 * 1)
Discriminator
Image (from generator) (28 * 28 * 1)
Conv2D (14 * 14 * 64)
LeakyReLU (alpha = 0.2) + Dropout (0.3)
Conv2D (7 * 7 * 128)
LeakyReLU (alpha = 0.2) + Dropout (0.3)
Flatten (None, 6272)
Dense (activation = 'sigmoid') (None, 1)

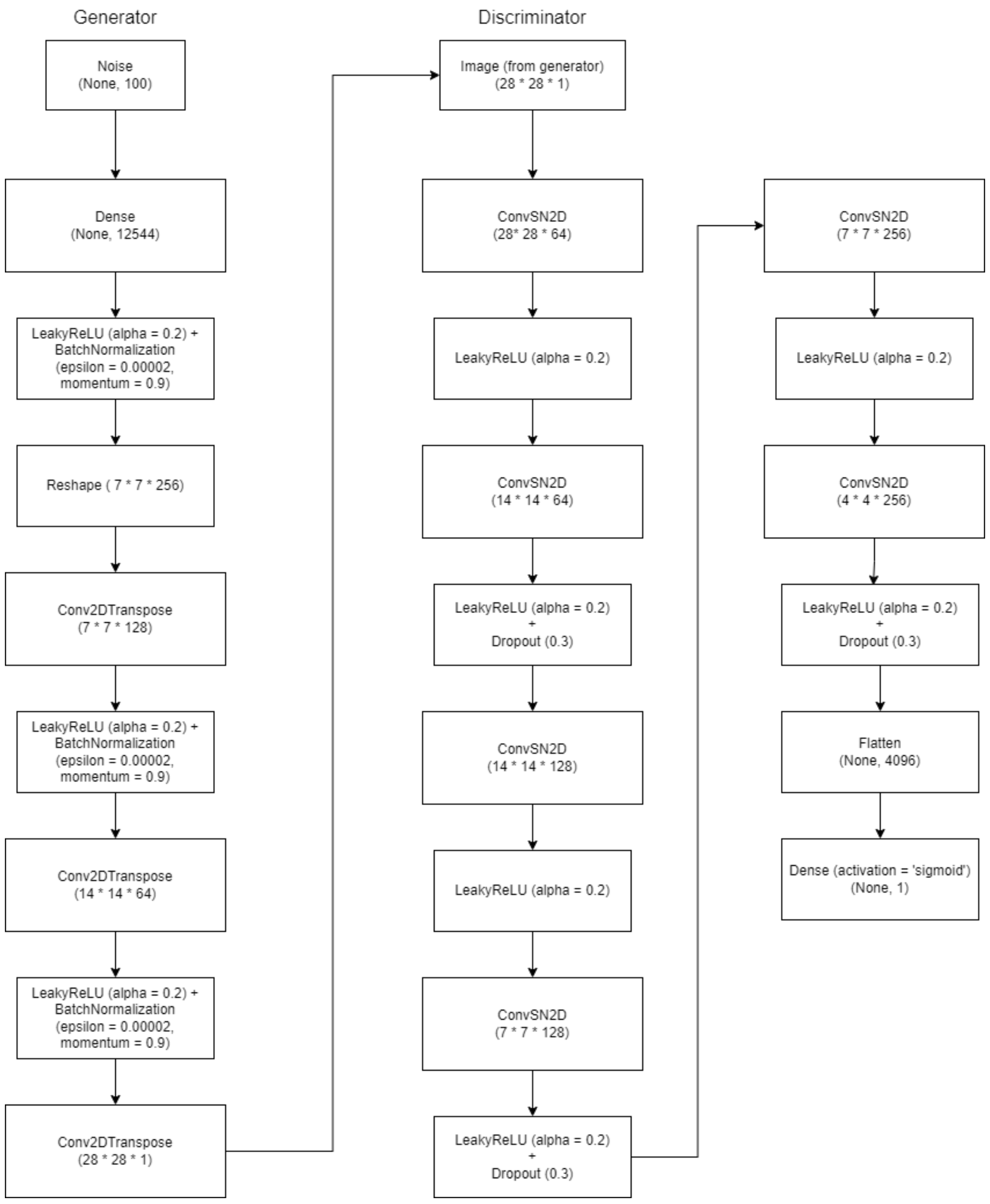
Figure 4: Model architecture of the DCGAN

- **Suggested Model 2: Spectral Normalization GAN (SNGAN)**

  The implementation of SNGAN modifies the discriminator architecture obtained from the GitHub repository [2].

  There is only a minor modification to the generator by assigning batch normalization with epsilon=0.00002 and momentum=0.9, which is beneficial for a large dataset variation.

  The Conv2D layer in the discriminator is replaced with ConvSN2D, and the network is modified to have three sets of two ConvSN2D layers, each followed by a LeakyReLU activation and dropout layer. The ConvSN2D introduces a spectral normalization to normalize the spectral norm of the weight matrices. It helps constrain the Lipschitz constant of the discriminator, preventing it from dominating the generator in the training process. Additionally, using two ConvSN2D layers with different stride and filter values helps capture features from varying scales of the input image.



Generator
Noise (None, 100)
Dense (None, 12544)
LeakyReLU (alpha = 0.2) + BatchNormalization (epsilon = 0.00002, momentum = 0.9)
Reshape (7 * 7 * 256)
Conv2DTranspose (7 * 7 * 128)
LeakyReLU (alpha = 0.2) + BatchNormalization (epsilon = 0.00002, momentum = 0.9)
Conv2DTranspose (14 * 14 * 64)
LeakyReLU (alpha = 0.2) + BatchNormalization (epsilon = 0.00002, momentum = 0.9)
Conv2DTranspose (28 * 28 * 1)
Discriminator
Image (from generator) (28 * 28 * 1)
ConvSN2D (28* 28 * 64)
LeakyReLU (alpha = 0.2)
ConvSN2D (14 * 14 * 64)
LeakyReLU (alpha = 0.2) + Dropout (0.3)
ConvSN2D (14 * 14 * 128)
LeakyReLU (alpha = 0.2)
ConvSN2D (7 * 7 * 128)
LeakyReLU (alpha = 0.2) + Dropout (0.3)
ConvSN2D (7 * 7 * 256)
LeakyReLU (alpha = 0.2)
ConvSN2D (4 * 4 * 256)
LeakyReLU (alpha = 0.2) + Dropout (0.3)
Flatten (None, 4096)
Dense (activation = 'sigmoid') (None, 1)

Figure 5: Model architecture of the SNGAN

- **<u>With Control Labels</u>**
  - **<u>Condition GAN (CGAN)</u>**

    To generate images for a particular class label of clothes, we consider the CGAN [3] which is a supervised learning version of GAN and provides an extra information, class label, during training.

    Figure 6 illustrates the network structure of the generator. A modification is made from its original source by reducing the number of channels from 256 to 128. The reason behind these changes is to improve the generalization capability of the generator. This network can generate a sample image sized 7x7 and up-sampling it to a grayscale image sized 28x28 by inputting a label and noise at the top.

Similarly, the discriminator receives the generated image from the generator, which is concatenated with a reshaped dense vector obtained by embedding a class label. The resulting output is a probability to indicate the degree of matching between the image and label input.

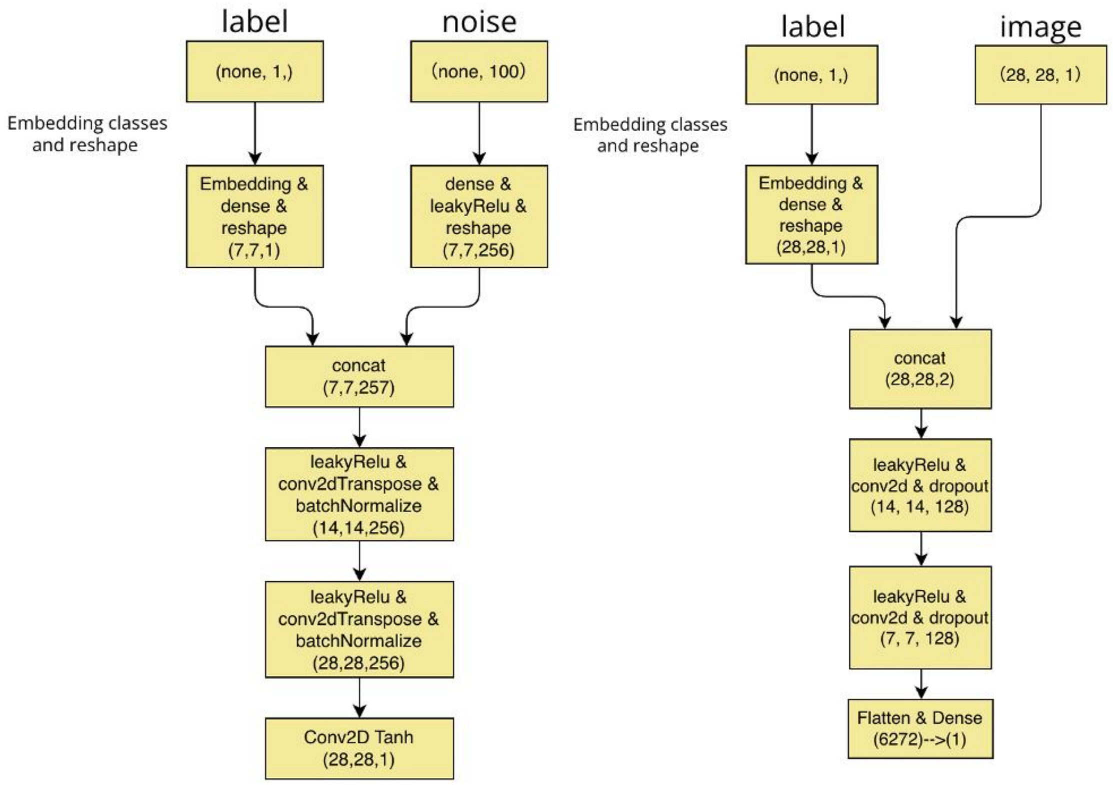

label
(none, 1,)
noise
(none, 100)
Embedding classes and reshape
Embedding & dense & reshape (7,7,1)
dense & leakyRelu & reshape (7,7,256)
concat (7,7,257)
leakyRelu & conv2dTranspose & batchNormalize (14,14,256)
leakyRelu & conv2dTranspose & batchNormalize (28,28,256)
Conv2D Tanh (28,28,1)
label
(none, 1,)
image
(28, 28, 1)
Embedding classes and reshape
Embedding & dense & reshape (28,28,1)
concat (28,28,2)
leakyRelu & conv2d & dropout (14, 14, 128)
leakyRelu & conv2d & dropout (7, 7, 128)
Flatten & Dense (6272)-->(1)

Figure 6: Network structure of the generator (left) and discriminator (right) in CGAN

Model Evaluation

- **Without Control label**

  For the evaluation of the baseline and suggested models, there are generated images across the baseline mode, DCGAN, and SNGAN, illustrated in Figure 7.

  From the result of the baseline model, we can observe some obvious noises and barely observe the object shape and the fine details of objects.

  Then, the DCGAN can generate images with fine details of the objects, such as the collar of a coat and the laces of shoes, displaying clear shapes and edges. Nonetheless, the model may occasionally produce images containing distorted or incomplete clothing.

  The SNGAN produces fashion images with shaper edges and a diverse range of visual characteristics, including intricate patterns on t-shirts, rather than distorted objects. These results reveal that the SNGAN outperforms both the baseline model and the DCGAN.

- **With Control label**

  To examine the quality of the generated images from the CGAN, we randomly generated 20 sample images for each class label trained at epoch 100. As shown in Figures 8 to 17, although there are some partial fragments and distortion of the objects, the variety of the generated images in each class label displayed a certain degree of generalization.

The results of the suggested models and CGAN demonstrated improved overall performance compared to the baseline model, which shows a clear shape and partial fine details in the generated fashion images. However, there is still room for improvement in image quality when it comes to testing samples.

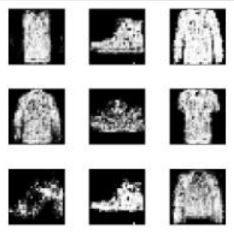
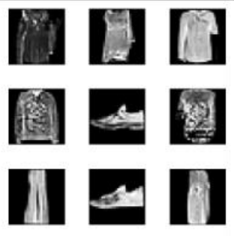
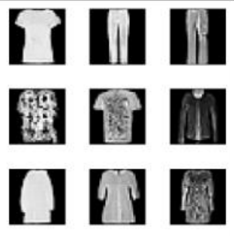
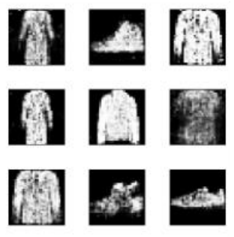
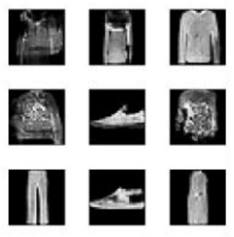
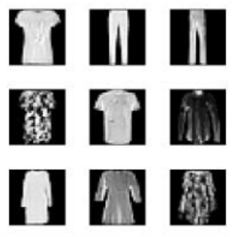
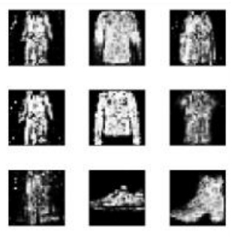
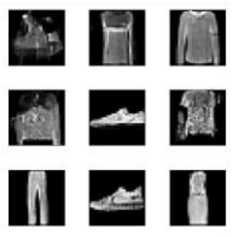
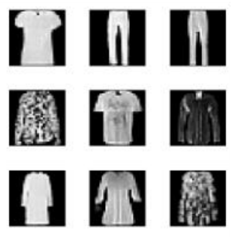
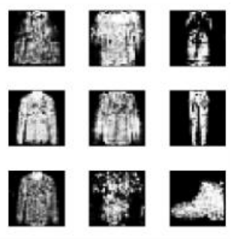
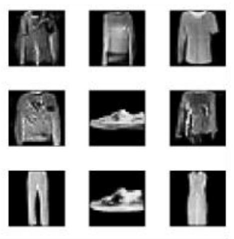
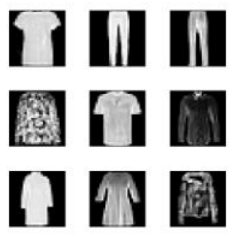
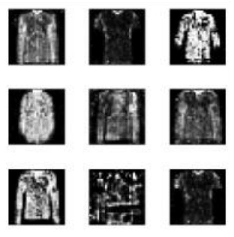
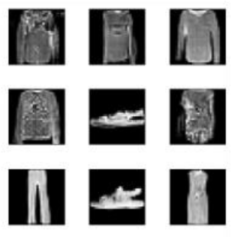
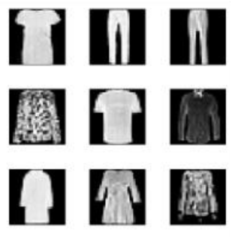
	FCGAN	**DCGAN**	**SNGAN**
Epoch 50			
Epoch 100			
Epoch 150			
Epoch 200			
Epoch 250			

Figure 7: Comparison of generated images across various GAN models at different epochs

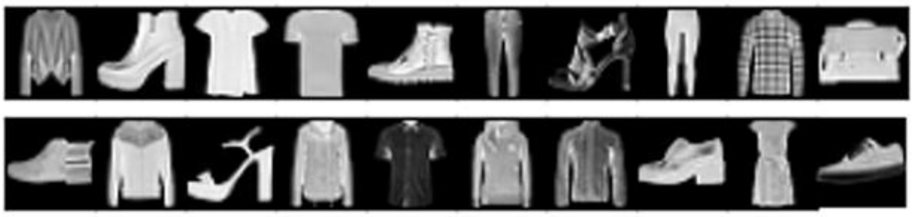


Figure 8: Visualization of testing samples

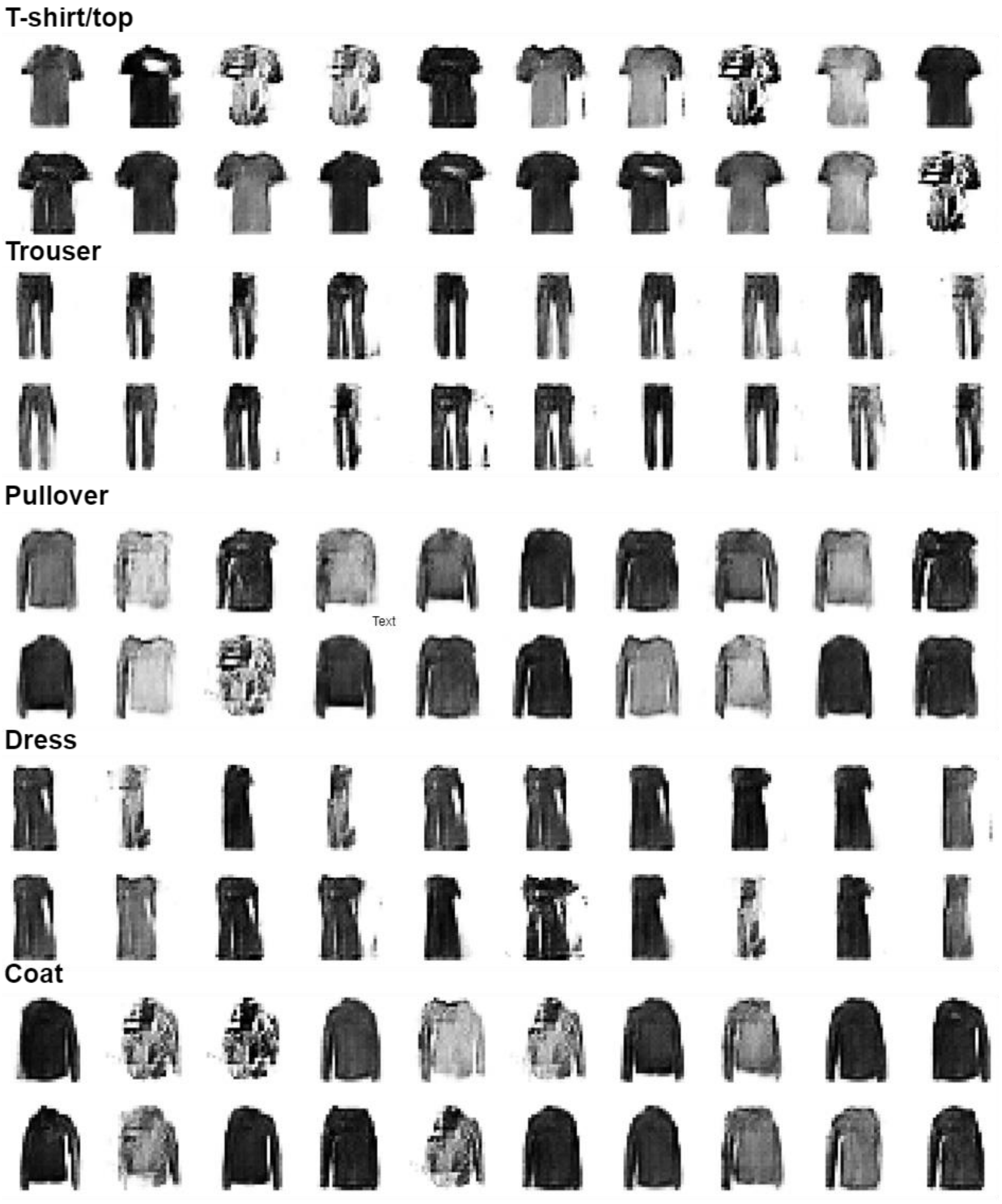


Figure 9: Visualization of generated fashion samples from the CGAN:
t-shirt/top, trouser, pullover, dress, and coat

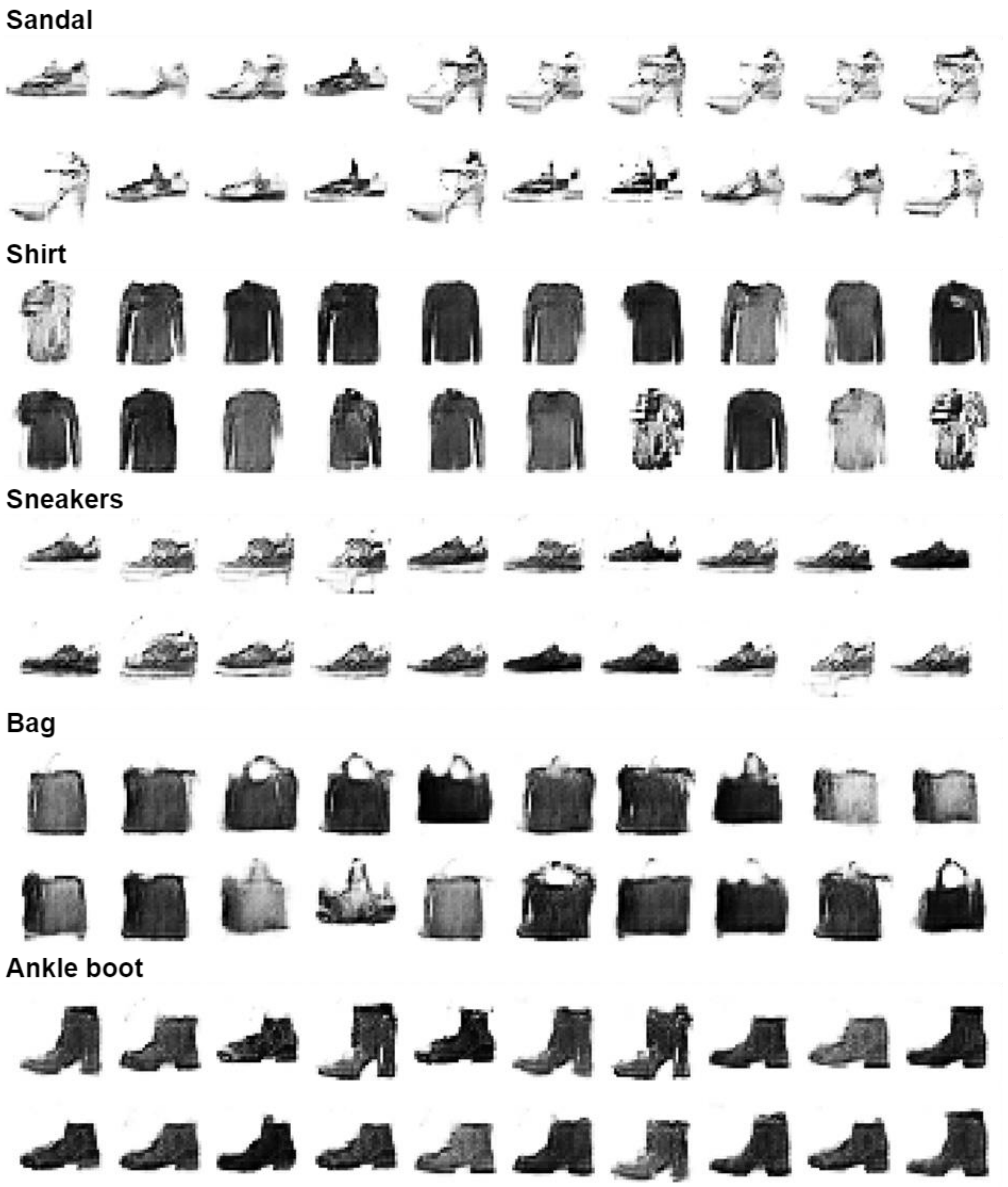


Figure 10: Visualization of generated fashion samples from the CGAN: sandal, shirt, sneakers, bag, and ankle boot

Limitation

One limitation of our GAN model training process is that the discriminator is always trained first. This situation can cause the generator to produce samples that are too similar to the real data, resulting in a discriminator that becomes too well at distinguishing between real and fake samples. It makes it harder for the generator to learn. Conversely, training the generator first may lead to a weak discriminator, resulting in poor-quality samples. To address this issue, the training step can be modified to interchange the training order of the generator and discriminator.

On the other hand, the suggested models have primarily focused on altering the architecture of the discriminator, which could potentially compromise the learning ability of the generator by reducing the quality of feedback provided by the discriminator. Specifically, in the case of SNGAN, the training time is quadrupled compared to the baseline model, leading to a significant increase in overall training cost to enhance the model performance.

Moreover, selecting an implementation framework can be a minor issue, as training time in Keras is much longer compared to the PyTorch framework, which results in higher training costs.

Objective evaluation metrics such as Fréchet inception distance and Inception score should be employed for model evaluation, as the current approach is subjective.

References

[1] "Deep Convolutional Generative Adversarial Network," *TensorFlow*, Dec. 2022. [Online]. Available: https://www.tensorflow.org/tutorials/generative/dcgan. Accessed: Apr. 27, 2023.

[2] IShengFang, "SpectralNormalizationKeras," *GitHub*, Mar. 2019. [Online]. Available: https://github.com/IShengFang/SpectralNormalizationKeras. Accessed: Apr. 27, 2023.

[3] Martra, P., "A beginner's guide to building a conditional GAN.," *Medium*, Mar. 29, 2023. [Online]. Available: https://pub.towardsai.net/a-beginners-guide-to-building-a-conditional-gan-d261e4d94882?gi=df1e89eb90a9 .Accessed: Apr. 27, 2023.