SYNTHETIC IMAGE DATA GENERATION

Introduction:

The process of generating any kind of data synthetically or artificially via programming is called Synthetic Data Generation. The data that can be used in these techniques can be images, text, audio, video, and so on. In this project, we will focus on Synthetic Image data Generation.

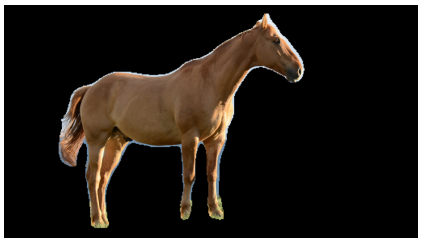
Synthetic data is important because it can be generated to meet specific needs or conditions that are not available in existing (real) data. This can be useful in numerous cases such as

1. when privacy requirements limit data availability or how it can be used
2. data is needed for testing a product to be released however such data either does not exist or is not available to the testers
3. training data is needed for machine learning algorithms. However, especially in the case of self-driving cars, such data is expensive to generate in real life.

Ways of generating synthetic data,

Method 1: Resizing and reshaping foreground

The easiest way to increase the trainable dataset is to extract the foreground or the subject image, randomly change its shape, size, rotate about the 2 axes and then paste it over another background.


Method 2: Using GANs

Another method is to create a generative model from the original dataset that produces synthetic data that closely resembles the real data; it is this later option we choose to explore here to generate synthetic data.

A class of generative models we choose to pursue in this work learns from large, real datasets to ensure the data it produces accurately resembles real-world data and builds on robust statistical, machine learning and state-of-the-art deep learning techniques. Specifically, we use generative adversarial networks (GANs), a class of powerful deep neural network algorithms to learn to mimic any distribution of data.

GAN and Conditional GAN:

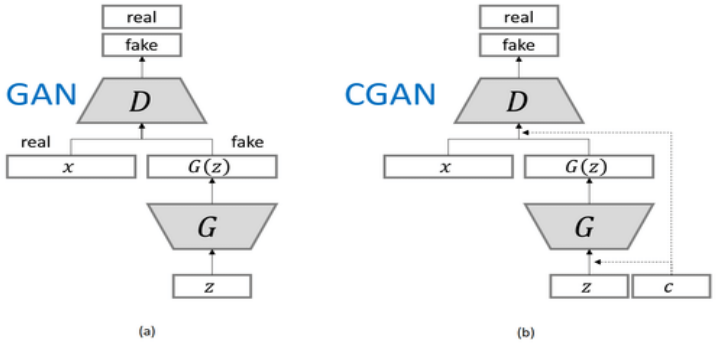
A **generative adversarial network (GAN)** is a deep neural system that can be used to generate synthetic data. **Synthetic data** can be identified as artificially generated data that

mimics the real data in terms of essential parameters, univariate and multivariate distributions, cross-correlations between the variables and so on.

Generative Adversarial Networks, or GANs, are an architecture for training generative models.

The architecture consists of a **generator and a discriminator model**. The generator model is responsible for generating new plausible examples that ideally are indistinguishable from real examples in the dataset. The discriminator model is responsible for classifying a given image as either real (drawn from the dataset) or fake (generated).

Conditional GANs use the extra information of classes available with the dataset to generate class targeted images.



Coding steps:

Moving toward the code, below is an detailed explanation for the code written

1. Input Image and preprocessing:

The image dataset used in this article is from kaggle repository. The dataset is a collection of over 6000 images distributed into 10 classes. The classes are dress, hat, longsleeve, outwear, pants, shirt, shoes, shorts, skirt and t-shirt.

Some examples of the images are

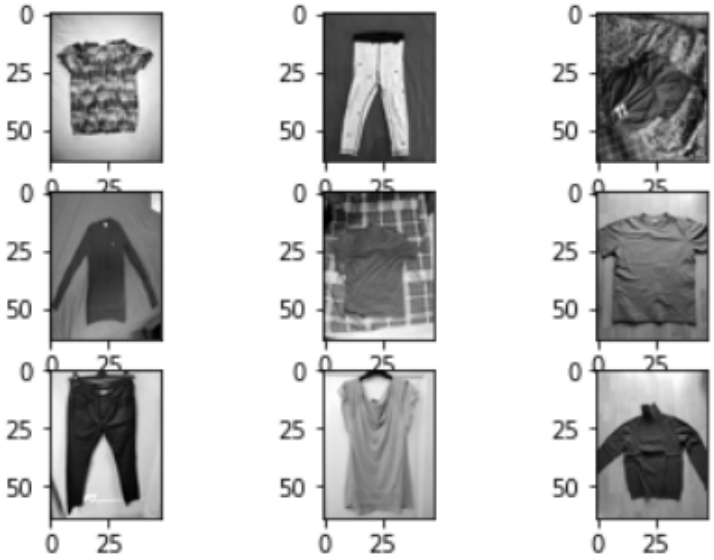
2. Image Preprocessing:

Let's consider the input image is of the size of 28x28x1. If not, then there is always an option of reshaping the images.

If the input image is an RGB picture, then convert it to grayscale image.

```
from PIL import Image
w = 28
h = 28
def get_image_data(path):
    img = Image.open(path)
    img = img.resize((w, h))
    img = img.convert('LA')
    np_img = np.array(img)[:, :, 0]
    return np_img
```

After preprocessing, we have something of this sort,

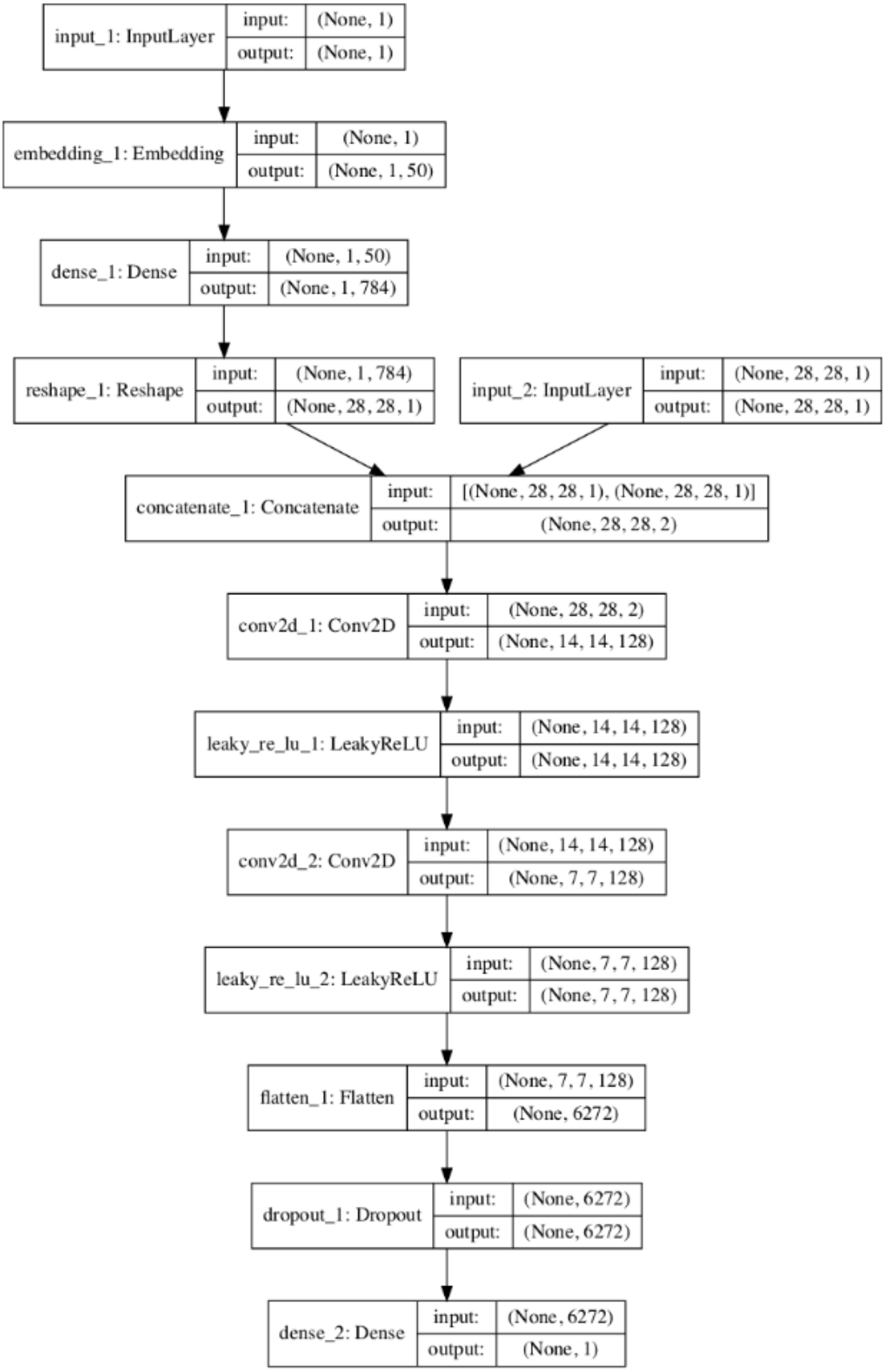


Now we move towards building the GAN model. Let's start with building the discriminator first.

3. Define discriminator:

As an input, the models take the image and its class as an input and give a True or a False value as an output, i.e. the discriminator tells us if the input image belongs to the particular class or not.

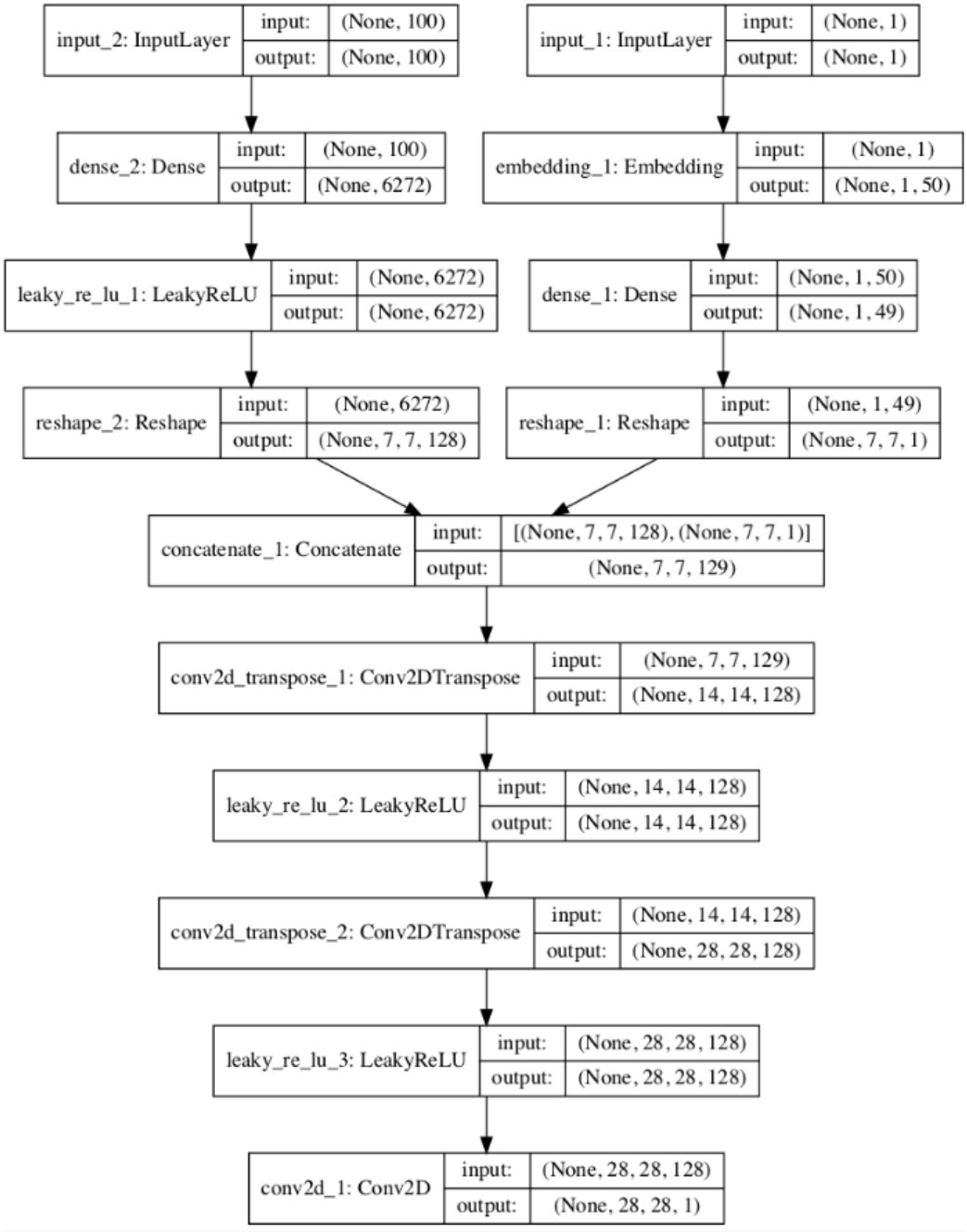
Below is an architecture for the CGAN discriminator we have followed here.


input_1: InputLayer
input: (None, 1)
output: (None, 1)
embedding_1: Embedding
input: (None, 1)
output: (None, 1, 50)
dense_1: Dense
input: (None, 1, 50)
output: (None, 1, 784)
reshape_1: Reshape
input: (None, 1, 784)
output: (None, 28, 28, 1)
input_2: InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
concatenate_1: Concatenate
input: [(None, 28, 28, 1), (None, 28, 28, 1)]
output: (None, 28, 28, 2)
conv2d_1: Conv2D
input: (None, 28, 28, 2)
output: (None, 14, 14, 128)
leaky_re_lu_1: LeakyReLU
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
conv2d_2: Conv2D
input: (None, 14, 14, 128)
output: (None, 7, 7, 128)
leaky_re_lu_2: LeakyReLU
input: (None, 7, 7, 128)
output: (None, 7, 7, 128)
flatten_1: Flatten
input: (None, 7, 7, 128)
output: (None, 6272)
dropout_1: Dropout
input: (None, 6272)
output: (None, 6272)
dense_2: Dense
input: (None, 6272)
output: (None, 1)

4. Define generator:

The generator again takes two inputs, one the noise which in this case is the latent noise of 100 dimension, and the other input is the class for the image to be generated.
As an output it gives an image of dimensions 28x28x1.

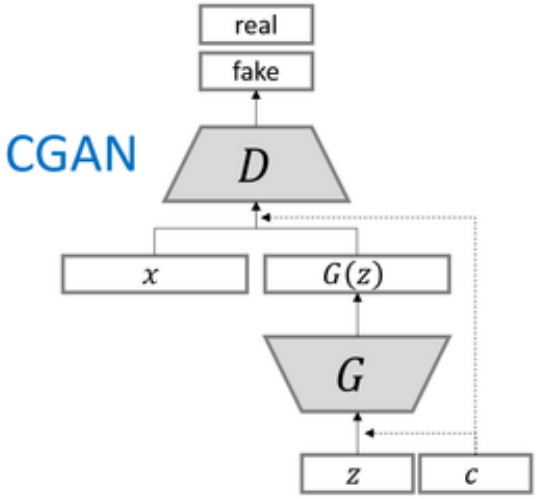
The architecture used for the generator model that we have used over here is as below:



5. Define GAN model:

Once the discriminator and the generator are defined, we need to connect the two models to create a GAN model.

Referring to the earlier image,



After combining the discriminator and the generator, we get the GAN model. The inputs to this model is a label class and a latent noise of dimension 100 (in this case).

The output will be an image of size (28x28x1) generated by the generator belonging to the specified class.

6. Train GAN model:

For each epoch we divide the dataset into batches (47 batches in our case). Each batch will have the same number of data (64 in our case).

There are 2 steps while training the GAN model

Step 1: Train the discriminator to differentiate between real and fake images.

For this we divide the batch into half (32 for our case),

We generate an equal number of noise samples (32) for the “False” examples and then use the data to train the discriminator to train and detect if the input image belongs to the concerned class or not.

Step 2: Train the Generator to generate correct images

Once the discriminator is trained, we set,

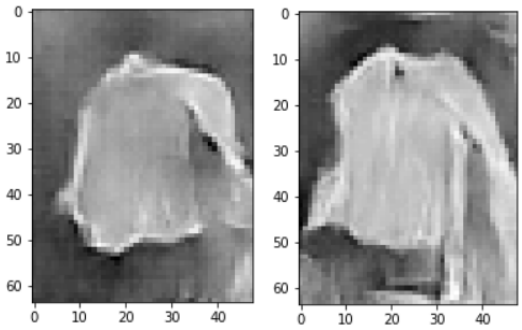
```
d_model.trainable = "False"
```

Now generate 64 random points in the latent dimension. And label all of them as true images with some random label class for each noise. Give this as an input to the GAN model.

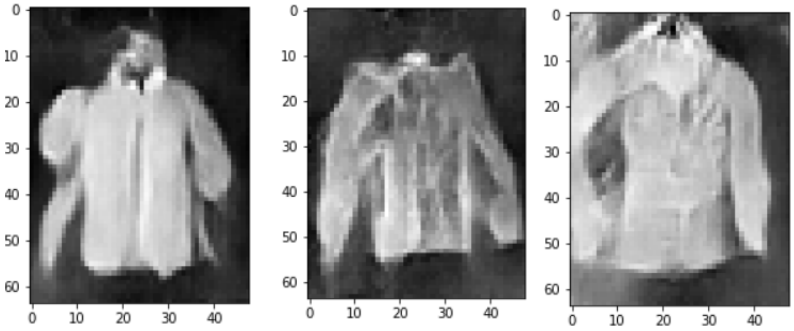
Results:

The results are as follows

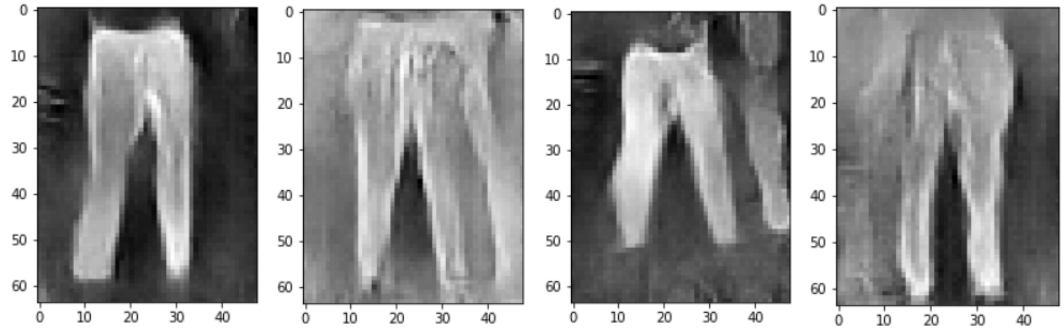
1. Long Sleeve



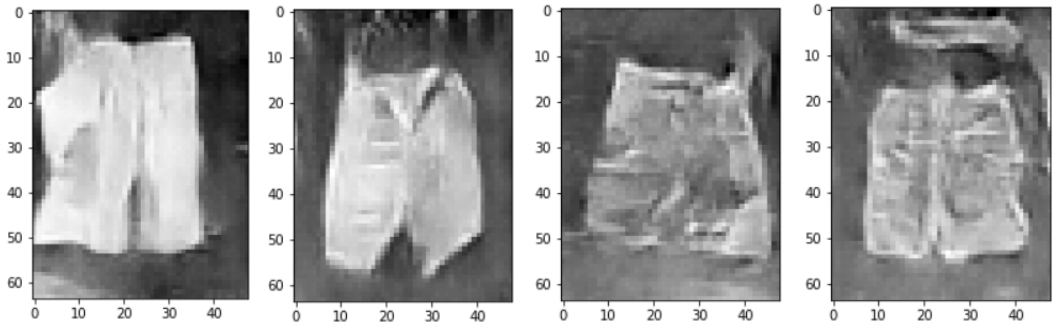
2. Outwear



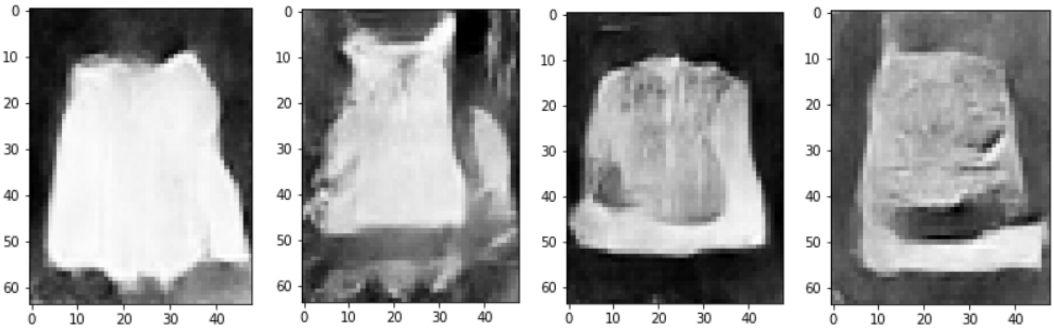
3. Pants



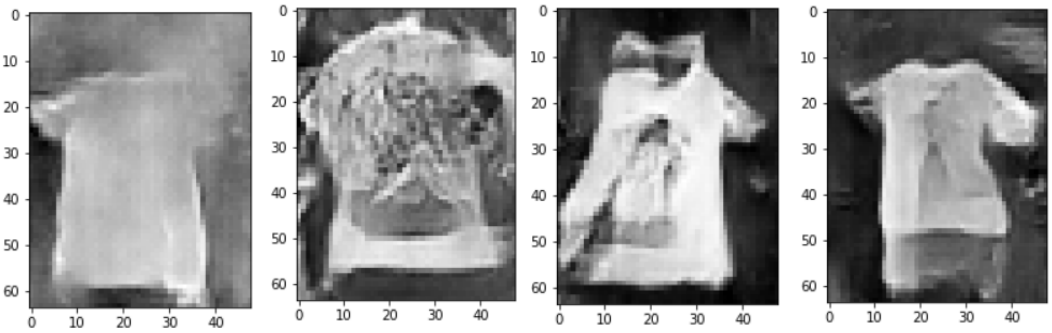
4. Shorts



5. Skirt



6. T-shirt



Improvements:

The model can further be improved if we are able to **preprocess the input images** better. Completely extract the foreground from the background so that we can get a better image dataset to train on.

Also an **increase in the batchsize or the training dataset** can result in a significant improvement in the final results.

References:

1. https://www.kaggle.com/agrigorev/clothing-dataset-full
2. https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/
3. https://research.aimultiple.com/synthetic-data/
4. https://towardsdatascience.com/create-a-synthetic-image-dataset-the-what-the-why-and-the-how-f820e6b6f718
5. https://medium.com/featurepreneur/generate-synthetic-image-data-for-your-next-machine-learning-project-74cf71b65a8f
6. https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/
7. https://medium.datadriveninvestor.com/an-introduction-to-conditional-gans-cgans-727d1f5bb011

8. https://medium.com/analytics-vidhya/step-by-step-implementation-of-conditional-generative-adversarial-networks-54e4b47497d6
9. https://research.aimultiple.com/synthetic-data/