Deep Learning for Business
MBA G514

End-Sem Project

3-D Image Generation using GAN

SUBMITTED TO

Dr. Revendranath Tirumalsety

SUBMITTED BY

NAME	BITS ID
SHOYEB KHAN	2021H1540806P
VISHVA BHALODIYA	2021H1540833P
FRANCIS XAVIER	2021H1540823P
MAANVENDRA SINGH SONGIRA	2021H150814P
SOWGANDH JYOTHULA	2021H1540848P

Problem Statement

The traditional method of developing a 3-D model is very complicated. This affects the ordinary user’s enthusiasm for creative design. According to almost everyone in the tech community, the metaverse is not a hot topic — it's the hot topic. Probably the closest thing we've seen to this mania was the promise of the internet in the '90s, so it's only fair that this concept has been labeled the next generation of the internet.

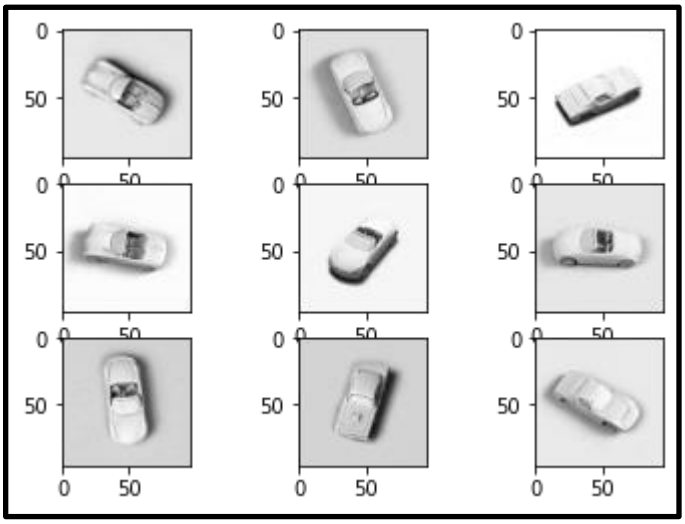
However, there's still one phase of the journey that needs to be addressed before brands can even start to conceive this whole idea of entering the metaverse: 3D modeling. So, if the metaverse really is to become the new iteration of the internet, the production of 3D models of real-world objects needs to be possible at scale, as this will lay the foundation for a virtual nirvana.

The McKinsey’s “Next in Personalization 2021” report says that customers' demand for improved experiences in the past two years has been such that three-quarters decided to switch to a new store, product or buying method during the pandemic as they looked for augmented experiences elsewhere. Moreover, in a world where one-to-one personalization is already possible at scale, the study also found that 71% of consumers still expect brands to deliver , and 76% get frustrated when this doesn't happen.

The modern way of designing and constructing 3D models involves usage of some of the popular 3D modeling software like NX, CATIA, SolidWorks or 3D scanners to obtain digital 3D models. However, it is generally a very exhaustive task to quickly develop some creative or innovative design for an existing 3D model. Therefore, exploring effective 3D image generation methods is an essential aspect in the domain of computer graphics and computer vision.

Data Preprocessing :

1. Loaded the dataset from the tensorflow dataset. We gave the batch size = -1 to get the entire dataset.
2. Uses tfds.as_numpy to convert -> np.array
3. Split into train x values and y train labels by setting the as_supervised = True.
4. We retrieved the unique count values from the y_train.
5. For each level in the unique level(classes) , we summed up the counts where y_train == level or class.
6. Using these indices we subset the x_train dataset.
7. The image shape is 96 X 96 X 1 .
8. The plot of images in the train dataset :



9. The length of the dataset is 4860 images (3-D)
10. For the capsule layer , we have implemented a new activation function called squash function , which is a nonlinear function . It drives the large vector to 1 and small vector to 0. The reason for implementing this is to overcome the limitation with a CNN for the object recognition task which is that the activation of its neurons is based on the chances of detecting specific image features.
11. Neurons do not consider the properties or features of an image, such as pose, texture, and deformation of the objects in the image.
12. While a typical neuron outputs a single scalar value, a capsule outputs a vector representing a generalized set of related object properties.
13. Transforming autoencoders make use of this complex feature detectors called a capsule, to explicitly capture the exact pose of each feature in the image and this is how they try to learn the overall transformation matrix.
14. The goal was not to recognize objects in images but to accept an image and its pose as input and output the same image in the original pose.

METHODOLOGY :

Typically GANs are usually used for modeling the distribution of image data and/or associated attributes for other image-based applications like style transfer, fake but similar data generation,image-to-image translation. All these applications use somewhat modified versions of the GAN architecture. Therefore, our generator and discriminator have conventionally been modeled as deep CNNs following the DCGAN guidelines. Motivated by the stronger intuition behind and the superior performance of CapsNets with respect to CNNs, we design our CapsuleGAN framework to incorporate capsule-layers instead of convolutional layers in the GAN discriminator, which fundamentally performs a two-class classification task.
The model primarily consists of a DCGAN with a built in Capsule Architecture in the Discriminator.

Working of Discriminator :

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.
The Discriminator Model takes an image as an input (generated and real) and classifies it as real or fake.

Generated images come from the Generator and the real images come from the training data.

The discriminator model is the simple binary classification model.

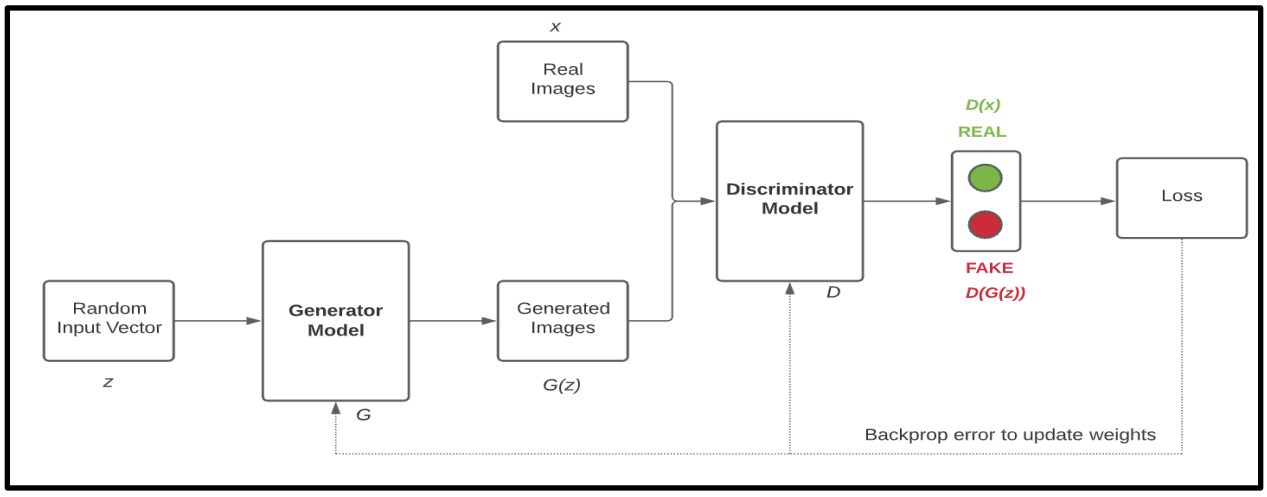
The discriminator's training data comes from two sources:

- Real data instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- Fake data instances created by the generator. The discriminator uses these instances as negative examples during training.

The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss. We use the generator loss during generator training.

During discriminator training:

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



The Generator Model G takes a random input vector z as an input and generates the images G(z). These generated images along with the real images x from training data are then fed to the Discriminator Model D. The Discriminator Model then classifies the images as real or fake. Then, we have to measure the loss and this loss has to be back propagated to update the weights of the Generator and the Discriminator.

When we are training the Discriminator, we have to freeze the Generator and back propagate errors to only update the Discriminator.

When we are training the Generator, we have to freeze the Discriminator and back propagate errors to only update the Generator.

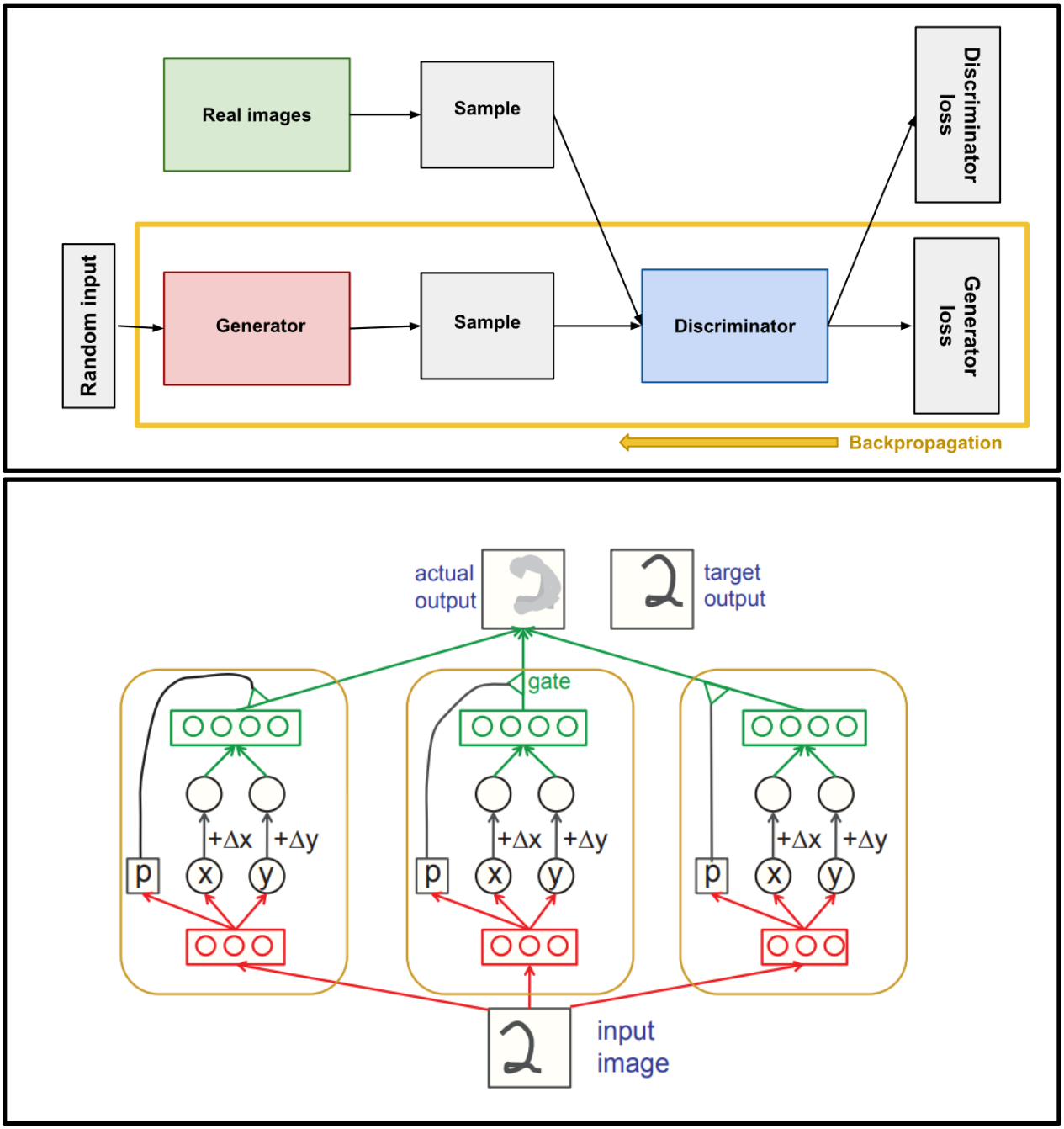
Thus the Generator Model and the Discriminator Model getting better and better at each epoch.

The limitation with a CNN for the object recognition task is that the activation of its neurons is based on the chances of detecting specific image features. Neurons do not consider the properties or features of an image, such as pose, texture, and deformation of the objects in the image. In other words, CNNs are incapable because of their invariance as a result of the pooling operation. Basically, a capsule is a group of neural layers. While a typical neuron outputs a single scalar value, a capsule outputs a vector representing a generalized set of related object properties. A capsule attempts to capture many object properties like pose (position, angle of view, size), deformation, the texture inside an image to define the probability of some object existence.

Transforming autoencoders make use of this complex feature detectors called a capsule, to explicitly capture the exact pose of each feature in the image and this is how they try to learn the overall transformation matrix. Capsules allow the autoencoder to maintain translational invariance without throwing away important positional information. They are not only capable of recognizing features in different poses and lighting conditions but are also capable of outputting pose-specific variables to be used by higher visual layers rather than discarding them. The goal was not to recognize objects in images but to accept an image and its pose as input and output the same image in the original pose.

WORKING OF GENERATOR:

DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively.The Generator creates fake images attempting to fool the Discriminator into believing they are real. The first convolution layer has 128 filters with a kernel size of 5x5 and a stride length of 1. LeakyReLU has been used as the activation function. The second convolution layer has 64 filters with a kernel size of 5x5 and a stride length of 2.

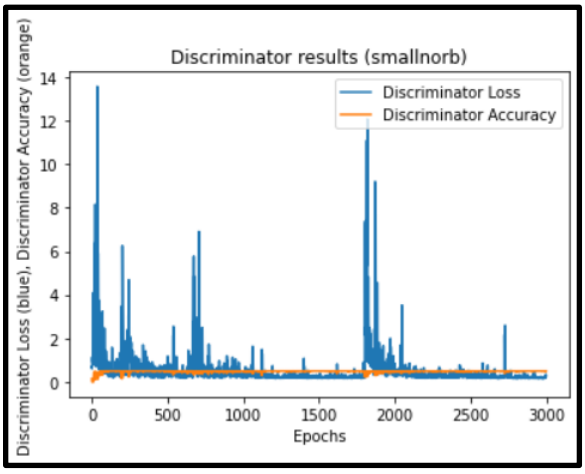


Three capsules of a transforming auto-encoder that models translations. Each capsule in the figure has 3 recognition units and 4 generation units. The weights on the connections are learned by backpropagating the discrepancy between the actual and target outputs.

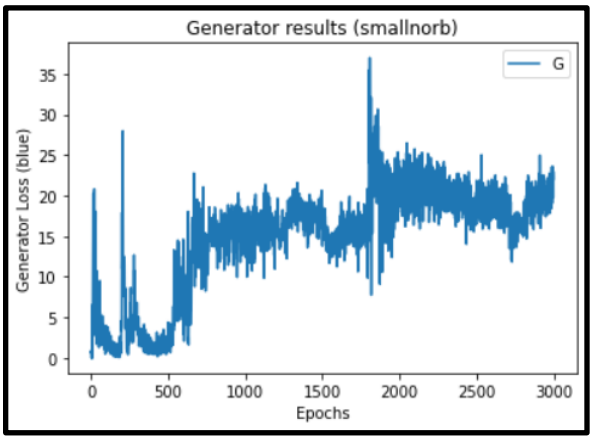
One of the major advantages of capsules that output explicit instantiation parameters is that they provide a simple way to recognize wholes by recognizing their parts. If a capsule can learn to output the pose of its visual entity in a vector that is linearly related to the “natural” representations of pose used in computer graphics, there is a simple and highly selective test for whether the visual entities represented by two active capsules, A and B, have the right spatial relationship to activate a higher-level capsule C.

Result :

Discriminator Loss Plot :



Generator Loss Plot :



Using transformers to analyze the Depth of the objects in the image :

Here , the white part represents the depth of the objects/persons. The varying intensity of the color represents the value of the depth.

The black color represents the absence of the objects/persons. The main reason for implementing the depth representation using the transformers is that CapsNet failed to get the depth sense in the output. In order to fill the gap or compensate , we used hugging faces and Intel collaborated transformer models to find the depth of the objects/images.

CapsNet Result :

At 600 epoch :