Capsule Network GAN vs. DCGAN vs. Vanilla GAN for Apparel Image Generation

Kruthi N Raj

Pre-Final Year(B.E.) Student, Department of Computer Science and Engineering, Bangalore Institute of Technology, Bangalore

---

Abstract - *Generative Adversarial Networks have produced considerable enthusiasm in the field of Artificial Intelligence owing to their ability to generate new data. Among the plethora of applications, GANs are commonly utilised for apparel designing in the fashion sector. GANs can be seen in artistic circumstances working as creative helpers to designers, creating distinctive items with realistic, attractive, and/or considerate traits. GANs are generative models that use 2 neural networks, a generator and a discriminator that compete with each other. While the generator is responsible for generating realistic data, the discriminator tries to identify whether the data is made artificially. They use existing data, in this case, images of clothing to generate new plausible images of apparels. The generators pretend to be designers while the discriminators act as design judges. Three distinct GANs are implemented and compared in this study. GANs (Generative Adversarial Networks), DCGAN (Deep Convolutional Generative Adversarial Network), and Capsule Network GAN (CapsNet GAN) with capsule network in discriminator are the three types.*

***Key Words*: Generative Adversarial Networks, Deep Convolutional Generative Adversarial Network, Capsule Network, Capsule Network Generative Adversarial Networks, Fashion Industry, Fashion-MNIST**

1. INTRODUCTION

In machine learning, generative modelling is an unsupervised learning job that entails automatically detecting and learning regularities or patterns in input data so that the model may be used to produce or output new examples that could have been drawn from the original dataset. Generative Adversarial Networks are part of the generative model family. It implies that they have the ability to create/generate new stuff. GANs, in general, are a model architecture for training a generative model, and deep learning models are most commonly used in

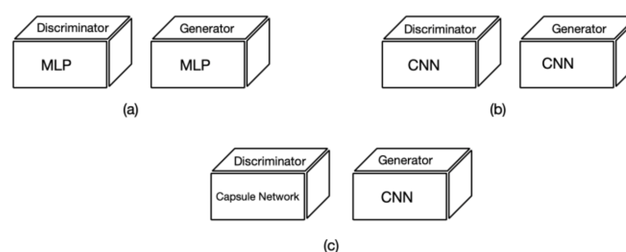


Fig 1: The structures of GANs used in this paper. (a) is GAN, using MLP (b) is DCGAN, using CNN and (c) is CapNetGAN, using capsule network in discriminator and CNN in generator

this architecture. They are an ingenious way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model, which we train to generate new examples, and the discriminator model, which attempts to classify examples as either real (from the domain) or fake (generated). The two models are trained in an adversarial zero-sum game until the discriminator model is tricked around half of the time, indicating that the generator model is creating credible examples. GAN's generator/discriminator models usually use Convolutional Neural Networks or CNNs with the picture data. This can be attributed both to the first description of the technique in computer vision and the use of CNNs and

image data and to the remarkable progress made with CNNs more generally over the last years in achieving state-of-the-art results in a number of computer vision tasks, such as object detection and facial recognition.

GANs are an exciting and rapidly changing field that fulfils the promise of generative models by generating realistic examples across a wide range of problem domains, most notably in image-to-image translation tasks, and in generating realistic photos of objects, scenes, and people that even humans cannot tell are fake. In particular to the fashion industry, Artificial intelligence has recently been incorporated to provide new services, and research into combining fashion design and AI is ongoing. In the fashion business, generative adversarial networks that synthesis realistic-looking images have been widely used. The increasing diversity of customer wants, strong worldwide competition, and decreasing time-to-market (a.k.a. "fast fashion") all add to the designers' struggle. Recent advancements in deep generative models have opened up new avenues for designers to overcome cognitive barriers by automating the development and/or editing of design concepts.

Ian Goodfellow et al.[1] first proposed GANs in 2014, and this issue has since opened up a new area of research. Within a few years, the scientific community produced a plethora of publications on this subject, some of which had quite intriguing titles. This paper compares three such GAN networks namely Generative Adversarial Networks or GANs(Goodfellow et al., 2014)[1] , Deep Convolutional Generative Adversarial Network or DCGAN(Radford et al., 2015)[2] and Capsule Network GAN(CapsNet GAN)(Jaiswal et al., 2018)[32] for apparel image production using the Fashion-MNIST dataset.

2. GENERATIVE ADVERSARIAL NETWORK

As proposed by Goodfellow et al.[1], GAN is made up of two separate networks: a Generator and a Discriminator. This generator creates false data samples and tries to mislead the discriminator. On the other hand, the Discriminator seeks to differentiate between true and fraudulent specimens. The generator and the discriminator are multilayer perceptrons, both of which compete in the training phase. The stages are performed multiple times and, after each repetition, the generator and the discrimination officer improve and improve in their respective roles.

While the Generator is generates synthetic samples given random noise [selected from a latent space], the Discriminator is a binary classifier that distinguishes between whether the input sample is real [output a scalar value 1] or fake [output a scalar value 0]. The Discriminator aspires to be the best at what it does. When the Discriminator is given a synthetic sample [created by the Generator], it wants to call it out as fake, while the Generator wants to create samples in such a way that the Discriminator calls it out as real. In a way, the Generator is attempting to deceive the Discriminator. The Generator and the Discriminator are antagonistic in this formulation, which adds to its beauty.

2.1. Training Process

Detailed GAN training as proposed by Goodfellow et al.[1] is as follows,

1. We take some noise from a random distribution and feed it to the Generator G to generate the false x
2. Discriminator D is fed with both fake and real data alternatively
3. The discriminator D calculates loss for both real and fake data and adds them up to compute final loss Dloss
4. Since each network has a different objective function, the generator G calculates the loss from it's noise as Gloss
5. These two losses are back-propagated to their respective networks to learn from and adjust the parameters according to the loss
6. An optimization algorithm is applied and this process is repeated for a number of epochs until a satisfactory result is obtained.

Before calculating the final loss, the discriminator network runs twice (once for real, once for fake), whereas the generator simply runs once. We calculate the gradients based on their parameters and back propagate across their networks independently after we have these two losses. The generator G becomes better and better at producing realistic results, while the discriminator D gets better and better at determining which ones are real and which are counterfeit. The model should produce high real data probabilities and low fake data probability (generator's output).

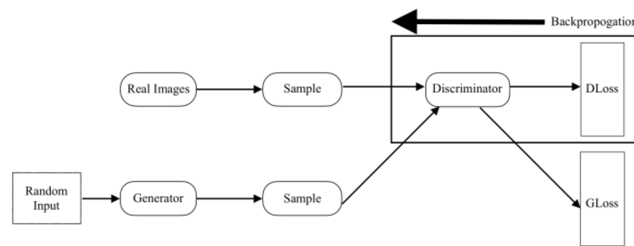


Fig 2: Overview of GAN Architecture

2.2. Objective Function

The Discriminator function is denoted by the letter D, while the Generator function is denoted by the letter G. The following are the functions and variables defined by Goodfellow et al.[1] :

1. z - Noise vector
2. G(z) - Generator's output
3. x - training sample
4. D(x) - Discriminator's output for real data
5. D(G(z)) - Discriminator's output for generated fake data
6. Pz - probability distribution of the latent space(usually a random Gaussian distribution)
7. Pdata - the probability distribution of the training dataset

The functions D(x) and D(G(x)) produce score between 0 and 1. G(z) produces data of the same shape as that of the real input data but with noise. The aim is to build a discriminator model that maximises the real data and minimises the fake data. The generator model built should maximise the fake data. Thus, at the Discriminator D, D(x) should be maximised and D(G(z)) should be minimised and at the Generator G, D(G(z)) should be

maximised. The Discriminator wishes to reduce the probability of D(G(z)) to zero. As a result, it seeks to maximise (1-D(G(z))), whereas the Generator seeks to force the probability of D(G(z)) to 1 such that the Discriminator makes an error in identifying a generated sample as real. As a result, Generator wishes to minimise (1-D(G(z)). Figure gives the objective function of GAN.

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{log prob of D predicting that real-world data is genuine}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{log prob of D predicting that G's generated data is not genuine}}$$

Fig 3: Objective Function of GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$$

  end for
  • Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Fig 4: A snippet from Ian et al.'s 2014 paper[1]

3. DEEP CONVOLUTIONAL GAN

DCGAN was proposed by Radford and other co-authors in 2015[2]. This was not the first time a group of academics attempted to employ Convolutional Networks in GANs, but it was the first time they were successful in combining them into a full-scale GAN model. DCGAN is a GAN architecture extension that uses deep convolutional neural networks for both the generator and discriminator models, as well as model and training configurations that result in robust training of a generator model. DCGAN is significant because it proposed the model restrictions needed to effectively create high-quality generator models in practise. In turn, this architecture served as the foundation for the quick creation of a huge number of GAN extensions and applications.

3.1. Architecture

The key component of the DCGAN model is to replace the fully connected layers in the generator with convolutional layers. Radford et al.[2] established a family of architectures in his paper after significant model exploration that resulted in robust training across a range of datasets and enabled for training better resolution and deeper generative models.

The writers of the original paper[2] mention three sources of inspiration in building this architecture.
1. Spatial downsampling convolutions can be used to replace pooling operations
2. Fully connected layers after convolutions can be eliminated
3. Normalisation activations can help gradient flow

The following are the architecture guidelines for stable Deep Convolutional GANs:

1. Strided convolutions (discriminator) and fractional-strided convolutions (generator) should be used in place of any other pooling layers
2. Batch normalisation should be used in both the generator and the discriminator
3. For deeper architecture, fully connected hidden layers should be removed.
4. In the generator, employ ReLU activation for all layers except for Tanh in output (Since images are normalised between [-1, 1] rather than [0, 1], Tanh is preferred to sigmoid.)
5. For all layers, use LeakyReLU activation in the discriminator

Many modern GAN literature has now been elaborated on these architectural standards. The structure's other fundamental is, the first layer of DCGAN, which accepts a uniform noise distribution Z as input, might be referred to as fully connected network because it is simply a matrix multiplication, and the result is then

formed into a 4-dimensional tensor and utilised as the start of the convolution stack. The final convolution layer is flattened and fed into a single sigmoid output for the discriminator.

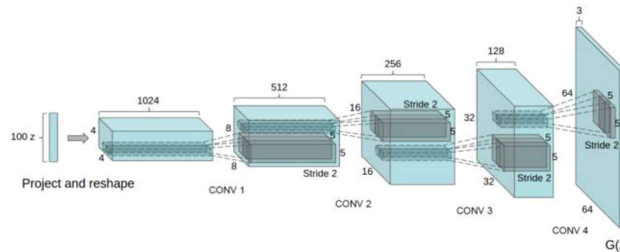


Fig 5: DCGAN architecture used by Radford et al.[2] to generate 64x64 RGB bedroom images from the LSUN dataset

More information on the network's parameters and training are as follows:

1. Other than scaling to the range of the tanh activation function [-1,1], no pre-processing is required for the training images

2. All models were trained using 128-batch mini-batch stochastic gradient descent (SGD)

3. All weights were drawn from a zero-centered Normal distribution with a standard deviation of 0.02

4. In all models of the LeakyReLU, the slope of the leak was adjusted at 0.2

5. Previously, momentum was employed to accelerate training; however, this paper used the Adam optimiser with customised hyperparameters. The learning rate has been set to 0.0002

6. Set beta1 to 0.5 for the momentum term

3.2. Unsupervised Representation and Feature Extraction

Many applications for GANs have been investigated, and much of the research is focused on improving image synthesis quality. Because they require class labels for conditioning, many of the methods for attaining high-quality picture synthesis are truly supervised learning techniques.

The fundamental idea is to use the discriminator's characteristics as a feature extractor for a classification model. Radford et al.[2], in particular, investigated the use of unsupervised GAN feature extractors in conjunction with an L2 + SVM classification model. The SVM model employs a high-dimensional hyperplane and a loss function that seeks to optimise inter-class distance based on the margin between nearest points in each class. The SVM model is an excellent classifier, but it's not a feature extractor, and using it on images as they are will result in a huge number of local minima, effectively rendering the problem intractable. As a result, the DCGAN acts as a feature extractor, reducing the dimensionality of the images while maintaining semantics, allowing an SVM to develop a discriminative model.

3.3. Overfitting

The paper[2] discusses about GAN overfitting. In the context of supervised learning, overfitting is extremely intuitive. The highly parametric model adapts itself to match the training data accurately and without error. When we take a step back from the statistics of the bias-variance tradeoff, we may think of overfitting as the model's generalisability, or how well it performs on training data versus testing data. Looking at this idea in the context of DCGANs, the generator's objective is to generate data that the discriminator predicts as 'real', that is, data that closely resembles the training dataset. The generator appears to be the most successful if it rejects

any attempt to add stochastic variations to data points and simply duplicates the training data exactly. The paper[2] proposed three intriguing techniques for demonstrating that the DCGAN model is not performing this function.

1. Models that learn quickly and generalise effectively, referred to as heuristic approximations

2. Collisions between auto-encoder hashes(To examine how similar the low-dimensional representations are after a trained auto-encoder has encoded both generated and original data)

3. Latent Space Smoothness (sharp transitions = overfitting)

3.4. Visualisation of Features

Previous researches has shown that supervised CNN training on big image datasets yields extremely powerful learned features. Object detectors are also learned by supervised CNNs trained on scene classification. The researchers, in the paper[2], showed that an unsupervised DCGAN trained on a large image dataset(LSUN bedroom dataset) may learn a hierarchy of useful features. It is demonstrated through guided back-propagation that the discriminator's learnt features activate on typical aspects of a bedroom, such as beds and windows.

3.5. Latent Space Interpolation

Latent Space interpolation allows for control of the generator. With their generated photos, Radford et al.[2] investigates this interpolation. One intriguing aspect of the latent space interpolation addressed in this study is that the Z vectors of individual points are not used. They instead demonstrated that averaging the Z vector created for three cases produced consistently reliable generations that obeyed linear arithmetic conceptually. Including object manipulation and facial expressions.

To create a smiling male image, for example, they do not just take the Z vector of one smiling lady, subtract the Z vector of one neutral woman, and then add the Z vector of one neutral guy, as shown in Figure 7. Rather, they pick the average Z vector of a set of created images with exterior attributes such as ‘smiling woman’, as shown in Figure 8.

This is one of the first instances that fully unsupervised models can learn to model object properties such as scale, rotation, and position convincingly.

Fig 6: The following is how feature visualisation in CNNs is accomplished. Gradient descent is used to train a generator network to produce a picture with the highest activation from a given feature.

Radford et al.[2] put this to the test with their discriminator model on the LSUN bedroom dataset and presented the image above

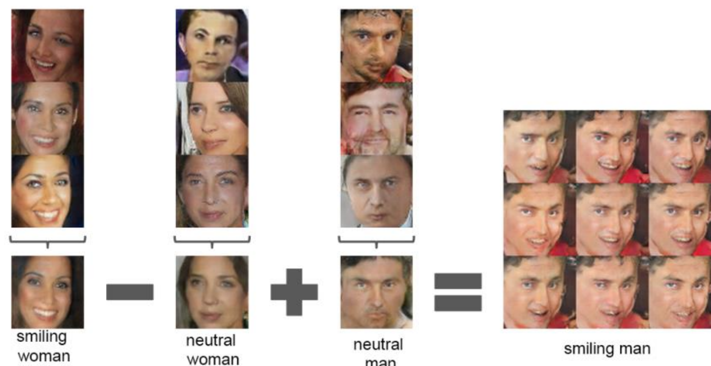


Fig 7: Image from the original paper[2] where the Z vector of one smiling woman is subtracted from the Z vector of one neutral woman and then is added to the Z vector of one neutral man to achieve a smiling man image

International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 08 | Aug 2021 www.irjet.net p-ISSN: 2395-0072

Fig 8: Image from the original paper[2] where they take the average Z vector of a series of generated images that display the external characteristics such as 'smiling woman'

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

Fig 9: Classification results of DCGAN implemented by Radford et al. [2] that is pre-trained on Imagenet-1k and the features are used to classify CIFAR-10 images

4. CAPSULE NETWORKS

Geoffrey Hinton and his research team invented capsule networks (CapsNets)[8]. It is a machine learning approach that attempts to more precisely simulate biological neuron architecture using artificial neural networks. It was released in 2017. The research offered a completely new notion, one that did not follow the typical layered structure of neural networks, but instead took a different approach to the problem. Sabour et al.[8] proposed that a typical CNN be enhanced by adding capsule structures and reusing the output from several of these capsules to generate more stable representations for higher capsules in the network.

4.1. Capsule

Hinton introduced capsules in his paper on Transforming Auto-encoders[7]. A capsule is a group of neurons that each activate for different aspects of an item, such as position, colour, and size. The length of the vector obtained from the capsule's output can be used to measure the entity's presence in a certain input, while the vector's orientation can be used to quantify the capsule's attributes. Capsules, unlike artificial neurons, are self-contained in nature. This means that when numerous capsules agree, the chances of a correct identification increase dramatically. Capsules are a type of artificial neural layer that allows you to store all of the essential and required information about the state of the features as a vector. Each capsule, according to Hinton and his team, performs complex internal computations on its input before encapsulating the result into a small vector of highly informative output. This is far superior to artificial neurons, which display the end results of computations using a single scalar output.

4.2. Drawbacks of CNN

CNNs are credited with giving computers the ability to decipher and analyse images. CNNs, however, have some restrictions. The convolutional layer is the most important aspect of a CNN. It looks for essential features in an image's pixels. Deeper layers in the network would detect simpler characteristics (such as edges and contours)

than higher layers, which would combine these simpler features to detect more complex features and produce classification predictions.Conventional CNNs have a key flaw, they don't usually take into account spatial distances between features when predicting features and classifying images. For example, in the case of a digitally manipulated image in which the components of a face (eyes, nose, mouth) are jumbled together(Figure 10), the presence of these elements provides sufficient proof that there is a face present in the image for a CNN. It makes no difference whether the mouth is above or below the eyes. Faces would be detected because the higher layers would detect the traits related to the faces on which they were trained.The cause of this issue is Max pooling. While max pooling is an approach that helps CNNs perform amazingly effectively, it also causes the loss of essential information such as the pose link between higher-level features.In short, the layers in a CNN do not account for the key spatial hierarchies between image detected features. In the above example, the simple presence of eyes and a mouth should not be sufficient to detect a face. The neural network should also be able to determine whether and how these features are orientated in relation to one another.

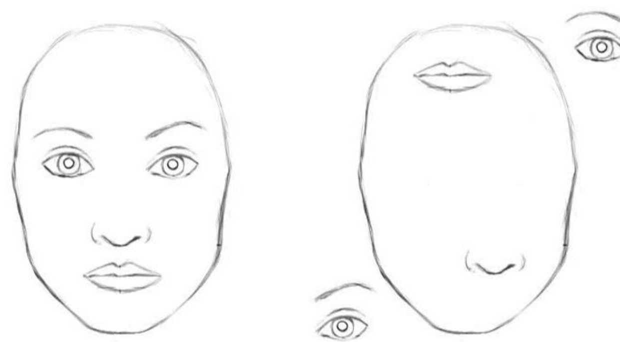


Fig 10: In both of the above images, a CNN would detect faces

4.3. Capsule Network Architecture

The capsule network defined by the authors in the paper[8] is separated into two major components: Encoder and Decoder. The encoder is associated with three major layers. They are summarized below:

1. Convolutional Layer: The primary function of this layer is to detect basic features in a two-dimensional image. The convolutional layer in CapsNet has 256 kernels of size 9x9x1 and a stride of 1.

2. PrimaryCaps Layer: This layer contains the primary capsules (which give the network its name) whose function it is to take the outputs of the convolutional layer's detected features and produce combinations of the input features. The layer contains 32 'primary capsules,' each of which generates an output tensor with the shape 6x6x8. This layer's ultimate output has the dimensions 6x6x8x32 (since there are 32 capsules).

3. DigitCaps Layer: There are ten capsules in this layer, one for each digit. (This is unique to this network because it was trained on the MNIST

4. dataset, which requires an output probability for ten classes.) Each capsule receives 6x6x8x32 shape vector and delivers a 16x10 matrix to the decoder.

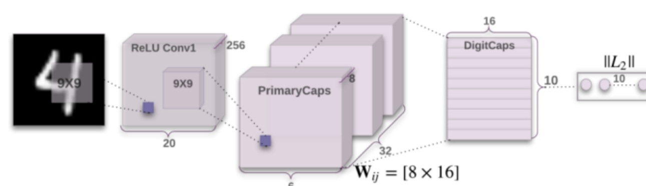


Fig 11: CapsNet Encoder Architecture by Sabour et al. [8]

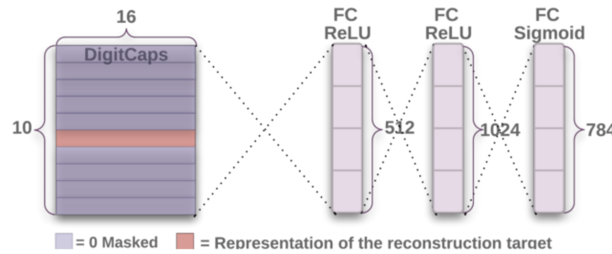


Fig 12: CapsNet Decoder Architecture by Sabour et al.[8]

The CapsNet decoder consists of three fully-connected layers (FCs) that train to decode a 16-dimensional input from the DigitCaps layer. The image replicated in the paper is 28x28 pixels in size.

4.4. Dynamic Routing Algorithm

It is a unique algorithm that is used to train a capsule network to perform the way it does. The primary idea behind the algorithm is best expressed as it's given in the paper[8]: Lower level capsule will send its input to the higher level capsule that "agrees" with its input. This is the essence of the dynamic routing algorithm.
Because the output of a capsule is a vector, a strong dynamic routing technique may be used to ensure that the output of each capsule is sent to a specified and appropriate parent in the layer above it. The process is comparable to classic training approaches in that higher-level capsules cover bigger areas of the image. However, it ignores the max pooling phase, which, as previously stated, is linked with CNN training. Dynamic routing does not discard information about the entity's precise location within the region. The location information for capsules in the bottom level of the hierarchy is place-coded based on which capsule is active at any given time. As we progress up the network, more and more information begins to be rate-coded in the magnitude component of the output vector of the capsule. Figure 13 is a line-by-line description of the algorithm from the paper[8]. In a lower-level layer, a capsule *i* must determine how to send its output vector to higher-level capsules *j*. It makes a decision by adjusting the scalar weight *c[ij]*, which multiplies its output vector and is then treated as input to a capsule higher up in the hierarchy.

1. The procedure's first line takes all existing capsules at a lower level *l* and their outputs *u*, as well as the number of routing iterations *r* specified by the user. The last line implies that the operation would yield the output of a capsule located above in the network's hierarchy, *v[j]*. This approach is similar to the forward pass of a traditional neural network.

2. The second line declares a coefficient *b[ij]*, which is a temporary value that is updated iteratively and saved in *c[ij]* after the method is completed. At the start of the procedure, it is set to zero.

3. The third line indicates that lines 4–7 will be iterated for *r* times.

4. The fourth line computes the vector *c[i]*, which contains all of the weights for the lower-level capsule *i*. This is then repeated for all lower level capsules. Softmax is employed to ensure that each captured weight *c[ij]* is non-negative and that the sum of all weights finally equals one.

5. Step five creates a linearly connected combination of all obtained input vectors after the weights for all lower level capsules have been determined. The routing coefficient *c[ij]* computed in the previous phase is used to weight each input vector.

6. The sixth line explains the vectors that are passed via the squash non-linearity, which ensures that the vector's direction component is preserved but that its length is limited to one.

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\mathbf{u}}_{j|i}$, r, l)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ softmax computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ squash computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Fig 13: Dynamic Routing Algorithm by Sabour et al.[8]

5. CAPSULE NETWORK GAN

GANs have mostly been used to simulate the distribution of picture data and related characteristics, as well as for other image-based applications such as image-to-image translation and image synthesis from textual descriptions. Conventionally, the generator and discriminator are built using deep CNNs following the DCGAN criteria. Jaiswal et al.[32] have designed the CapsNet GAN generator as a deep CNN using this concept. CapsNet discriminator, on the other hand, is designed using capsule-layers instead of convolutional layers because of the stronger intuition behind and higher performance of CapsNet networks compared to CNNs. The model is essentially made up of a DCGAN with a built-in Capsule Architecture in the Discriminator, which houses a classification function for identifying real and false labels for supplied images, and an unaltered Generator component(DCGAN generator architecture). The CapsNet GAN discriminator is architecturally similar to the CapsNet model proposed by Sabour et al.

CapsNets contain a high number of parameters in general because, first, each capsule generates a vector output rather than a single scalar and, second, each capsule includes extra parameters connected with all the capsules in the layer above it that are used to forecast their outputs. The number of parameters in the CapsuleGAN discriminator must be kept low for two reasons: (1) CapsNets are powerful models that can easily begin harshly penalising the generator early in the training process, causing the generator to fail completely or suffer from mode collapse, and (2) current implementations of the dynamic routing algorithm are slow. It is worth noting that the first rationale for keeping the number of parameters of the CapsNet modest is consistent with the typical design of convolutional discriminators as relatively shallow neural networks with a few number of relatively large-sized filters in their convolutional layers. The CapsuleGAN discriminator's final layer comprises a single capsule, the length of which reflects the likelihood that the discriminator's input is real or artificial. The Discriminator network is built on the capsule network architecture presented by Sabour et al.[8] in Figure 11.

Jaiswal et al.[32] trained the CapsuleGAN model using margin loss LM rather of the traditional binary cross-entropy loss since LM performs better for training CapsNets. As a result, the goal of CapsuleGAN is formulated as illustrated in Figure 14. Jaiswal et al.[32] trained the generator to minimise LM(D(G(z)), T = 1) rather than —LM (D(G(z)), T = 0). This effectively removed the down-weighting influence λ in LM while training the generator, which contains no capsules.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[-L_M(D(\mathbf{x}),\mathbf{T}=\mathbf{1})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[-L_M(D(G(\mathbf{z})),\mathbf{T}=\mathbf{0})]$$

Fig 14: Formulated objective of CapNetGAN

6. DATASET

For the experiments, the Fashion-MNIST dataset was utilized, which is a new standard dataset used in computer vision and deep learning. The Fashion-MNIST dataset includes 60,000 training (and 10,000 test) 28×28 pixel images of fashion and apparel items from 10 classes. The images are grayscale with a black background(0 pixel value) and the apparel images in white(pixel values near 255). This means if the images

were plotted, they would be mostly black with a white digit in the middle. Zalando developed Fashion-MNIST as a suitable alternative for the MNIST collection of handwritten digits.

6.1. Loading and Preparing the Dataset

Keras provides access to the Fashion-MNIST dataset. The images are 2D arrays of pixels, whereas convolutional neural networks require 3D arrays of images with one or more channels as input. The images are updated to have an additional dimension for the grayscale channel. Finally, the input values between [0, 255] will be normalised between -1 and 1. This implies that the value 0 will be mapped to -1, the value 255 to 1, and all values in between will be assigned a value in the range [-1, 1].

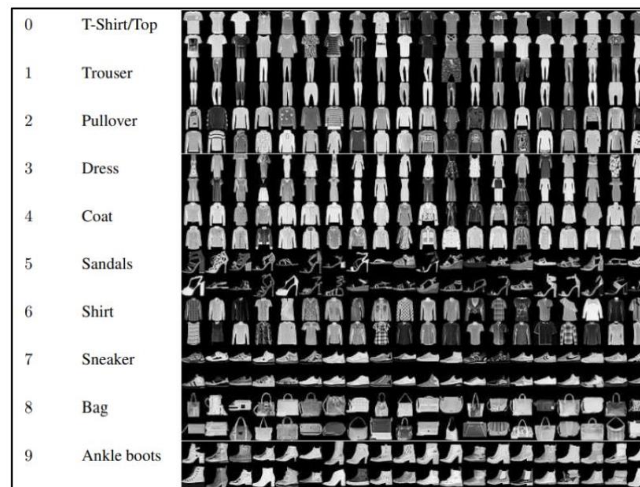


Fig 15: Class Labels and Images from the Fashion-MNIST Dataset

7. METHODOLOGY

To create the generated or fake samples of apparels, a batch of random points from the latent space is chosen to be used as input to the generator model. Then, as the real samples, a batch of samples from the training dataset is chosen to be fed into the discriminator. The discriminator model then generates predictions(probabilities) for the actual and fake samples, and the discriminator's weights are updated according to how accurate or inaccurate the predictions are. The model will be updated in batches, specifically with a collection of real examples and a collection of generated samples. In this study, I have implemented three distinct forms of GAN: Vanilla GAN, DCGAN, and CapNet GAN, utilising the Keras library on top of TensorFlow, and I have thoroughly explained my experimental settings here.

7.1. GAN Structure

The discriminator network will accept a flattened image as input and return the probability of it being part of the actual or fake dataset. Each image will be 28X28=784 pixels in size. There are three hidden layers in this network, each followed by a Leaky-ReLU nonlinearity and a Dropout layer to prevent overfitting.

The Generative Network takes a point from the latent space as input. The latent space is an arbitrarily specified vector space with Gaussian-distributed values. It has no meaning, but by randomly drawing points from this space and supplying them to the generator model during training, the generator model will assign meaning to the latent points and, thus, the latent space, until, at the end of training, the latent vector space represents a compressed representation of the output space, Fashion-MNIST images, that only the generator knows how to turn into plausible Fashion-MNIST images. The generator network's goal is to learn how to make indistinguishable pictures of apparel, which is why its output is a new image. The generator network outputs a

International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 08 | Aug 2021 www.irjet.net p-ISSN: 2395-0072

784-valued vector that represents a flattened 28X28 picture. There are three hidden layers in this network, each with a Leaky-ReLU nonlinearity. A TanH activation function is included in the output layer. A normal distribution with mean 0 and variance 1 is used to sample the random noise which is given as the input to the generator model.

For both neural networks, I used Adam as the optimisation algorithm, with a learning rate of 1e-4, or 0.0001. The loss function used for this task is Binary Cross Entopy Loss (BCE Loss), and it is used for this scenario as it

resembles the log-loss for both the Generator and Discriminator. The average of the loss calculated for each mini-batch is considered.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	25856
leaky_re_lu (LeakyReLU)	(None, 256)	0
dense_1 (Dense)	(None, 512)	131584
leaky_re_lu_1 (LeakyReLU)	(None, 512)	0
dense_2 (Dense)	(None, 1024)	525312
leaky_re_lu_2 (LeakyReLU)	(None, 1024)	0
dense_3 (Dense)	(None, 784)	803600

Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0

Fig 17: GAN Generator Architecture

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 1024)	803840
leaky_re_lu_3 (LeakyReLU)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524800
leaky_re_lu_4 (LeakyReLU)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 256)	131328
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 1)	257

Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0

Fig 18: GAN Discriminator Architecture

In the formula for Binary Cross Entropy Log(Figure $), the values y are named targets, v are the inputs, and w are the weights. Since weights are not needed, it'll be set to w_i=1 for all i. If we replace $v_i = D(x_i)$ and $y_i=1\ \forall\ i$ (for all i) in the BCE-Loss definition, we obtain the loss related to the real-images. Conversely if we set $v_i = D(G(z_i))$ and $y_i=0\ \forall$ i, we obtain the loss related to the fake-images. In the mathematical model of a GAN, the gradient of this is ascended, but here the function is minimised instead. But there is no need for concern about the sign because maximising a function is the same as minimising it's negative, and the BCE-Loss term has a minus sign. The real-images 'targets are always ones, while the fake-images 'targets are zero. By summing up these two discriminator losses we obtain the total mini-batch loss for the Discriminator.

Training the Generator to maximise log D(G(z)) rather than minimising log(1- D(G(z)) results in significantly higher gradients early in training. Both losses result in the same dynamics for the Generator and Discriminator, thus they may be changed around. Maximising log D(G(z)) is identical to minimising it's negative, and because the BCE-Loss definition has a minus sign, there is no need for concern about the sign. Similar to the Discriminator, if we set $v_i = D(G(z_i))$ and $y_i=1\ \forall$ i, we get the required loss to be minimised.

The weights of the generator model are updated based on the discriminator model's performance. When the discriminator is strong at identifying false samples, the generator model is updated more frequently, but when the discriminator model is relatively weak or confused at detecting fake samples, the generator model is updated less frequently. This establishes the two models' zero-sum or adversarial relationship.

$$L = \{l_1, ..., l_N\}^T, l_i = -w_i[y_i \cdot log(v_i) + (1-y) \cdot log(1-v_i)]$$

Fig 16: Binary Cross Entropy Loss

7.2. DCGAN Structure

The input of the discriminator model is image with one channel and 28X28 pixels in size. The output is binary classification, probability that a sample is real or fake. The discriminator model has two convolutional layers with 64 filters for the first and 128 filters for the second, a small kernel size of 5X5, and larger than normal stride of 2X2. Each layer is followed by a Leaky-ReLU nonlinearity and a Dropout layer.

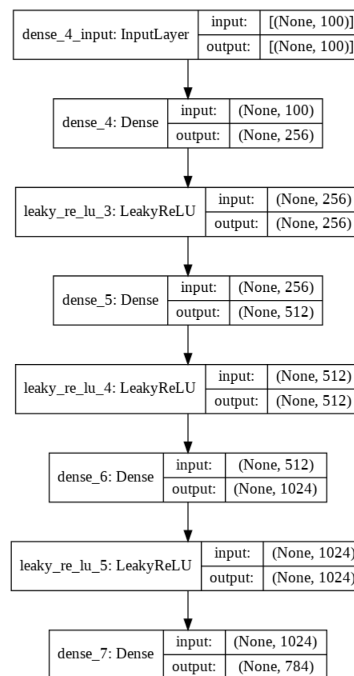


Fig 17: Generator and Discriminator Model Plot in GAN

The model has no pooling layers and only one node in the output layer to predict if the input sample is real or false. The model is trained to minimise the binary cross entropy loss function, appropriate for binary classification. Before the model generates an output prediction, the aggressive 2X2 stride downsamples the input picture, first from 28X28 to 14X14, then to 7X7. This pattern is intentional since I don't utilise pooling layers and instead use a big stride to create a comparable downsampling effect. As seen in the plot(Fig 22),the model expects two inputs and will predict a single output.

The generator model exhibits a similar pattern, but in reverse. The generator accepts a point from the latent space as input and produces a two-dimensional square grayscale image of 28×28 pixels with pixel values in [0,1] as output. A vector from the latent space with 100 dimensions is transformed into a 2D array with 28×28 or 784 values. The first hidden layer is a Dense layer with enough nodes to depict a low-resolution version of the output image. An image half the size (one quarter the area) of the output image would have 14X14 or 196 nodes, while an image one eighth the size (one eighth the area) would have 7X7 or 49 nodes.

Simply said, only one low-resolution image is not required; instead, many copies or interpretations of the input are desired. This is a phenomenon in convolutional neural networks where several parallel filters result in multiple parallel activation maps, termed feature maps, with diverse interpretations of the input. At reverse, we want multiple concurrent copies of our output with various learnt characteristics that can be collapsed into a final image in the output layer. Space is required for the model to invent, develop, or generate. As a result, the first hidden layer, the Dense, requires a sufficient number of nodes to support several low-resolution variants

of our output picture, such as 256. The activations from these nodes may then be moulded into something like an image to be sent into a convolutional layer, such as 256 distinct 7X7 feature maps.

The *Conv2DTranspose* layer is used in the upsampling process, which includes upsampling the low-quality image to a higher resolution version of the image. The *Conv2DTranspose* layer is configured with a stride of 2×2 and a kernel size of 5X5. This is repeated to arrive at our 28×28 output image. The second *Conv2DTranspose* layer has a filter size of 64, kernel size of 5X5 and a stride of 2X2, the output size of which is upscaled to 14X14 pixels. The output layer of the model is a *Conv2DTranspose* with one filter and a kernel size of 5×5, stride of 2X2 and 'same 'padding, designed to create a single upscaled feature map and preserve its dimensions at 28×28 pixels. A *tanh* activation is used to ensure output values are in the desired range of [-1,1].

Leaky-ReLU non linearity function is used for each layer in the discriminator and the generator of the model. Batch Normalisation is provided for each layer in the generator. For optimisation, Adam version of stochastic gradient descent with a learning rate of 1e-4 i.e 0.0001 is used. The loss function used for this task is Binary Cross Entopy Loss (BCE Loss).

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12544)	1254400
batch_normalization_v2 (Batc	(None, 12544)	50176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTran	(None, 7, 7, 128)	819200
batch_normalization_v2_1 (Ba	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	204800
batch_normalization_v2_2 (Ba	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 1)	1600

Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472

Fig 19: DCGAN Generator Architecture

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273

Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0

Fig 20: DCGAN Discriminator Architecture

7.3. CapsNet GAN Structure

The discriminator consists of a Capsule network adapted from the original paper[8] with some changes. As an input, the model is given a 28x28 pixel with a single colour channel from the Fashion-MNIST dataset. The first layer is a basic convolutional layer with 256 filters, a 9 pixel kernel size, and a stride value of 1. This layer transforms pixel intensities to local feature detector activities, which are subsequently utilised as inputs to the PrimaryCaps. This initial layer in many contemporary papers has a high filter size. The rationale for this is because it enables us to automatically expand the receptive field of all subsequent layers from now on. This layer, according to the original Dynamic Routing paper[8], takes low-level information from the image and allows us to transfer it to the Capsule network.

Unlike the study[8], however, I added extra Leaky ReLU activation and a batch normalisation on top of the first convolutional layer. The purpose of adding an activation function to the initial convolution is to make the Discriminator perform as well as possible, whereas an activation function such as a Leaky ReLU will not only introduce a non-linearity but will also take care of the vanishing gradient by allowing a small negative slope.

The core Capsule architecture begins here, since it specifies the PrimaryCaps layers. PrimaryCaps are a collection of convolution, reshape, and squashing functions. The lowest layer of the PrimaryCaps is a convolution with the same amount of filters as the preceding layer, 256, which implies an 8D vector with 32 feature maps. Following that, a reshape function is used to divide all output neurons into an 8D vector. At this level, primary capsules include collections of activations that indicate the orientation of the apparel, as well as the strength of the vector, which indicates the presence of the apparel. Finally, a special squashing function and a Batch Normalisation (not present in the original paper[8]) are applied to PrimaryCaps . Batch Normalisation showed to stabilise learning by normalising the input to each unit to have zero mean and unit variance. This helps to tackle training issues that arise due to poor initialisation and helps gradient flow in deeper models. Directly applying Batch Normalisation to internal Primary- and Digit layers however, resulted in sample oscillation and model instability. This was avoided by not applying Batch Normalisation to internals of the Capsules but rather to outputs as in the case above.

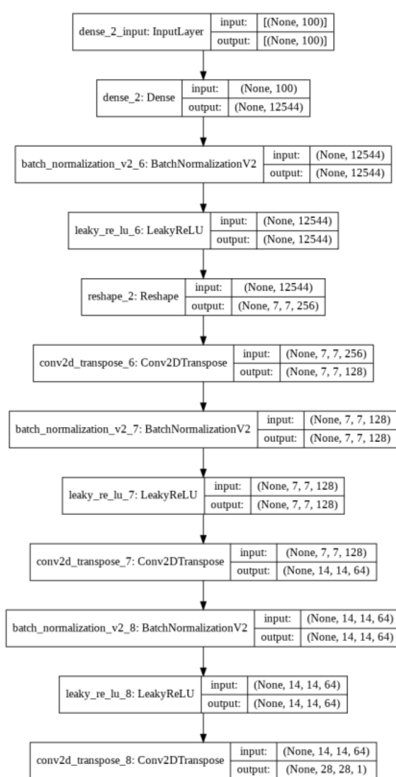


Fig 21: Generator Model Plot in DCGAN

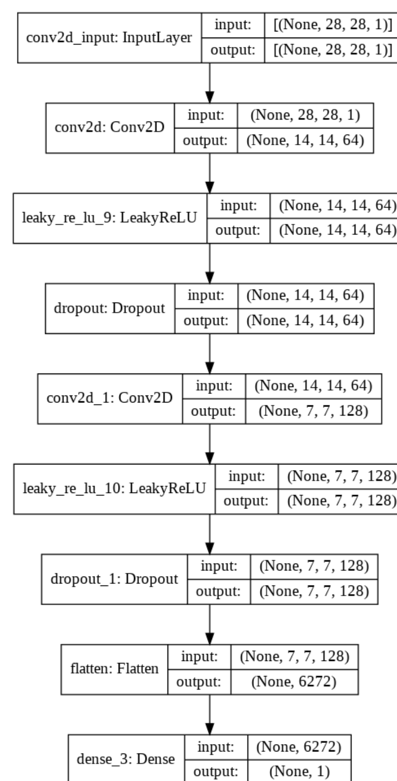


Fig 22: Discriminator Model Plot in DCGAN

DigitCaps are the architecture's second and most essential component, since they execute the "routing by agreement" method. DigitCaps, in comparison to PrimaryCaps, has been changed and works slightly differently from the original paper[8]. The key change is that at the end of each routing cycle, I replaced the squashing function with another Leaky ReLU activation, which otherwise causes artefacts on produced images. Although this method is the same in terms of computing, it is feasible that the implementation will result in a weaker equivariance.

Values are flattened before being sent from PrimaryCaps to DigitCaps. The reason for this is that Keras Dense layers used as value/weight holders can only accept flattened neurons. DigitCaps are not vectorised, but rather represented as a collection of flattened neurons, which differs the most from the original study. However, all math operations are performed in the same manner as if they were performed on vectors.

Flattened neurone that get passed into Keras Dense layer acts as a prediction vector û. The prediction vector is made up of 160 neurons, which are represented by a multiplication of 10 capsules and 16 vectors. We also changed Keras Dense layer settings like the kernel initialiser to he_normal and the bias initialiser to zeros. The prediction vector ʌû is then transferred to a three-length loop that includes a C coupling coefficient that is a softmax over the bias weights of the previous layer (which ensures that all capsules in the layer below total to one) and û. Following each multiplication, LeakyReLU is activated. The discriminator's final layer, which determines the realness score of each image, is a Keras Dense layer with a single neuron and a sigmoid.

The CapsNet GAN generator model is identical to the above explained DCGAN generator model. The Adam version of stochastic gradient descent, is utilised for optimisation. Binary Cross Entopy Loss is the loss function utilised for this work (BCE Loss).

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 28, 28, 1)]	0	
conv1 (Conv2D)	(None, 20, 20, 256)	20992	input_2[0][0]
leaky_re_lu_3 (LeakyReLU)	(None, 20, 20, 256)	0	conv1[0][0]
batch_normalization_3 (BatchNor	(None, 20, 20, 256)	1024	leaky_re_lu_3[0][0]
primarycap_conv2 (Conv2D)	(None, 6, 6, 256)	5308672	batch_normalization_3[0][0]
primarycap_reshape (Reshape)	(None, 1152, 8)	0	primarycap_conv2[0][0]
primarycap_squash (Lambda)	(None, 1152, 8)	0	primarycap_reshape[0][0]
batch_normalization_4 (BatchNor	(None, 1152, 8)	32	primarycap_squash[0][0]
flatten (Flatten)	(None, 9216)	0	batch_normalization_4[0][0]
uhat_digitcaps (Dense)	(None, 160)	1474720	flatten[0][0]
softmax_digitcaps1 (Activation)	(None, 160)	0	uhat_digitcaps[0][0]
dense_1 (Dense)	(None, 160)	25760	softmax_digitcaps1[0][0]
multiply (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_1[0][0]
leaky_re_lu_4 (LeakyReLU)	(None, 160)	0	multiply[0][0]
softmax_digitcaps2 (Activation)	(None, 160)	0	leaky_re_lu_4[0][0]
dense_2 (Dense)	(None, 160)	25760	softmax_digitcaps2[0][0]
multiply_1 (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_2[0][0]
leaky_re_lu_5 (LeakyReLU)	(None, 160)	0	multiply_1[0][0]
softmax_digitcaps3 (Activation)	(None, 160)	0	leaky_re_lu_5[0][0]
dense_3 (Dense)	(None, 160)	25760	softmax_digitcaps3[0][0]
multiply_2 (Multiply)	(None, 160)	0	uhat_digitcaps[0][0] dense_3[0][0]
leaky_re_lu_6 (LeakyReLU)	(None, 160)	0	multiply_2[0][0]
dense_4 (Dense)	(None, 1)	161	leaky_re_lu_6[0][0]

Total params: 6,882,881
Trainable params: 6,882,353
Non-trainable params: 528

Fig 23: CapsNet GAN Discriminator Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 100)]	0
dense (Dense)	(None, 12544)	1266944
batch_normalization (BatchNo	(None, 12544)	50176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTran	(None, 7, 7, 128)	819200
batch_normalization_1 (Batch	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	204800
batch_normalization_2 (Batch	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 1)	1600

Total params: 2,343,488
Trainable params: 2,318,016
Non-trainable params: 25,472

Fig 24: CapsNet GAN Generator Architecture

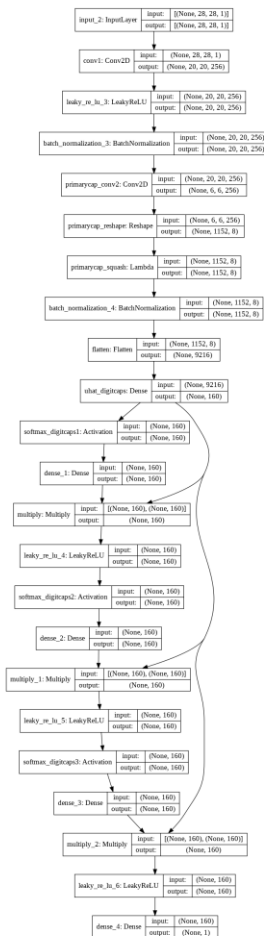


Fig 25: Discriminator Model Plot in CapsNet GAN.

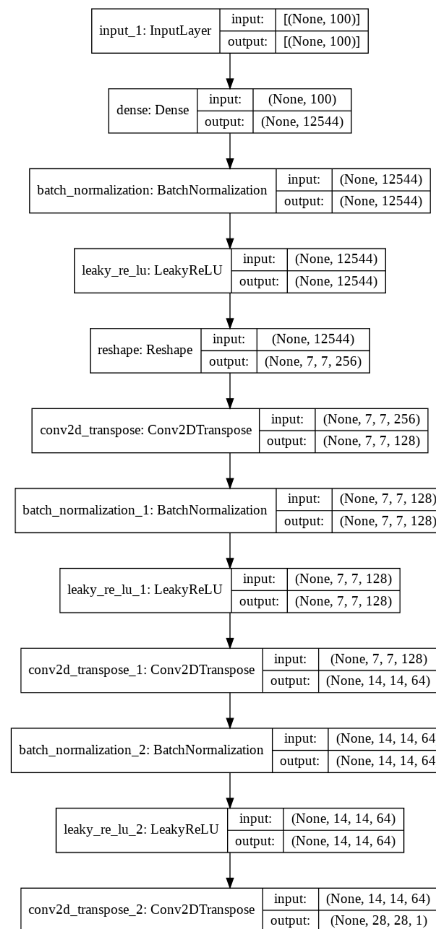


Fig 26: Generator Model Plot in CapsNet GAN

8. RESULTS

8. 1. Training Time

The NVIDIA Tesla T4 GPU, which is accessible in Google Colab, was used to train the GAN and DCGAN models. On a 1.4 GHz quad-core Intel Core i5 CPU, the CapsNet GAN was trained. Each model was trained for about 4000 epochs on the Fashion-MNIST dataset.

MODEL	AVG. TIME/EPOCH	TOTAL TIME/4000 EPOCHS
GAN	2.29916886479s	3 hrs (Approx.)
DCGAN	11.8152600765s	13 hrs (Approx.)
CapsNet GAN	2.48371303081s	2.8 hrs (Approx.)

Table 1: Training Time

8.2. Qualitative Evaluation

I compared the quality of images created at random using GAN (denoted by (a)), DCGAN (denoted by (b)), and CapsNet GAN (denoted by (c)).

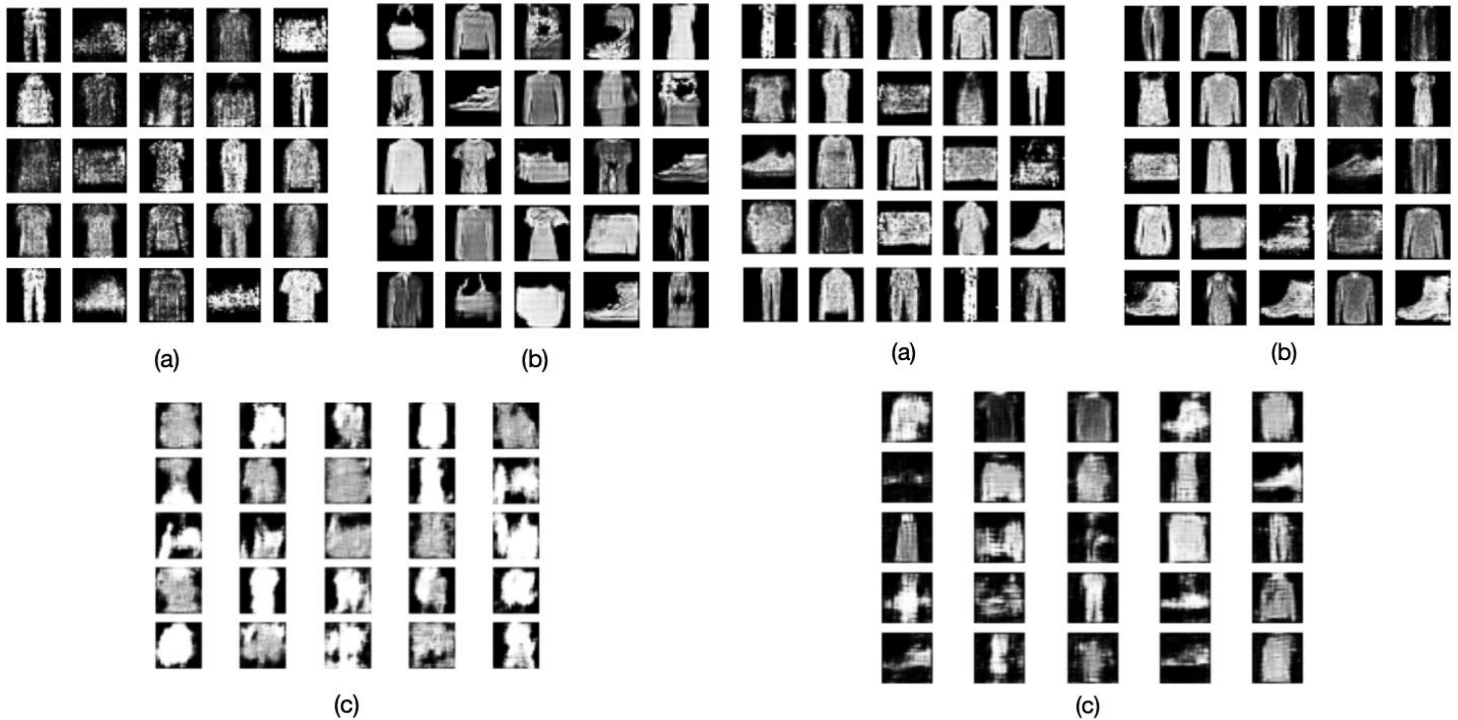


Fig 27: Epoch 200 **Fig 28:** Epoch 1000

Qualitatively, DCGAN produces better quality images during the initial epochs followed by GAN and the images produced by CapsNet GAN are comparatively not very accurate.

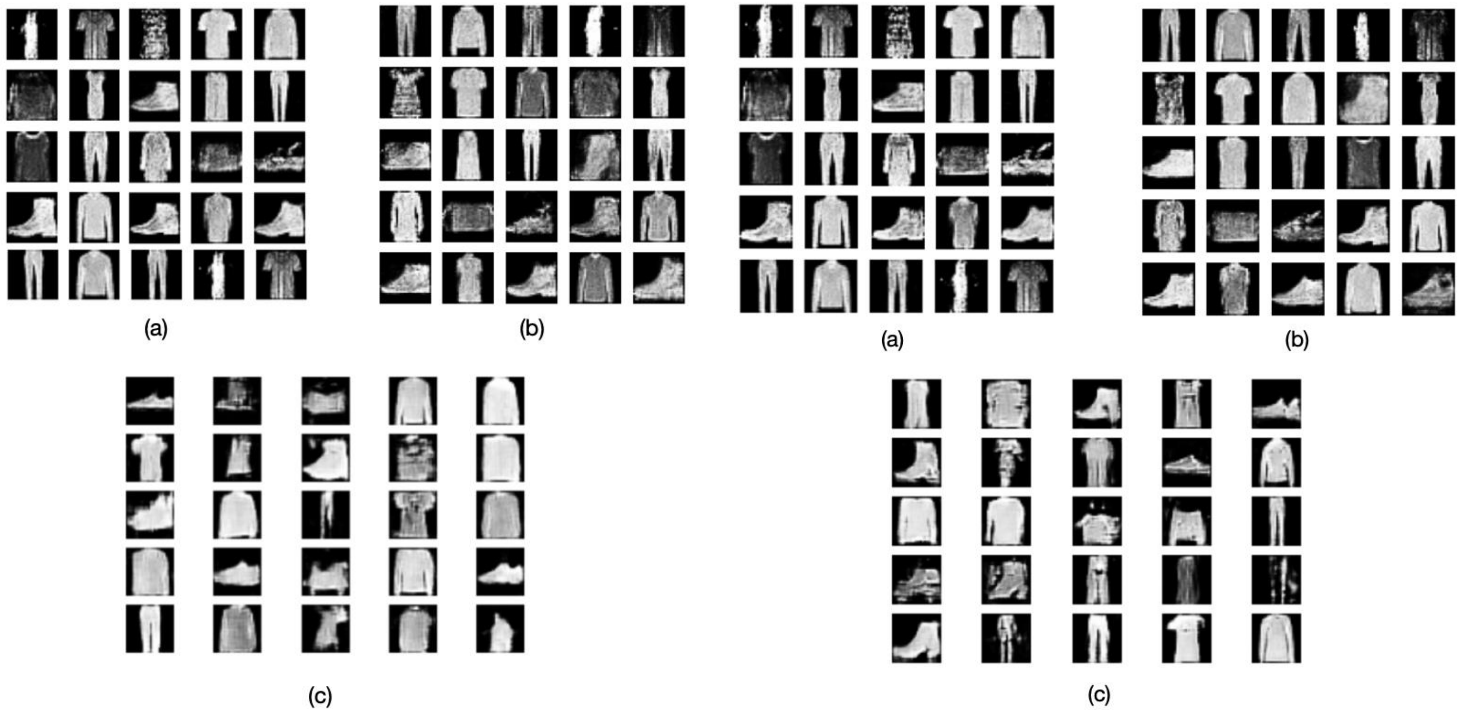


Fig 29: Epoch 2000

Fig 30: Epoch 3000

At the end of 4000 epochs, the images produced by DCGAN and CapsNet GAN are almost of similar quality. The images generated by CapsuleGAN are cleaner and sharper than those generated by DCGAN but DCGAN's images are, on average, more accurate when compared to the ones produced by CapsNet GAN. As a result, in the subsection below, I've included the findings of a quantitative comparison between DCGAN and CapsNet GAN for further analysis of their image generation performance.

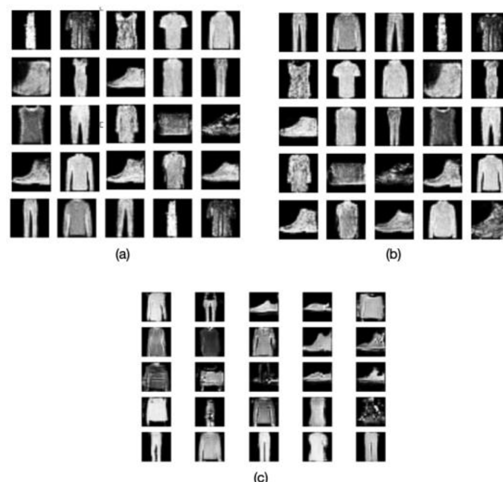


Fig 31: Epoch 4000

GAN is making good progress in image generation, but the quality of the images produced is comparatively low. At this stage of training, the diversity of the models in terms of generated clothing classes cannot be discussed because the images generated have to be more refined by further training.

8.2. QUANTITATIVE EVALUATION OF DCGAN AND CapsNet GAN

8.2.1. Inception Score

One issue with generative models is that there is no way to objectively assess the quality of the images created. As a result, it's usual to generate and save images at regular intervals during the model training process, and then utilise subjective human evaluation of the generated images to assess the quality of the images. There have been several attempts to provide an objective measure of generated image quality. The Inception Score, or IS[11], is an early and frequently used example of an objective assessment technique for produced pictures.
The Inception Score, or IS[11] for short, is a measure for assessing the quality of generated pictures, especially synthetic images produced by generative adversarial network models. Tim Salimans et al.[11] suggested the inception score. They developed the inception score as an attempt to remove the subjective human evaluation of images.

To categorise the produced images, the inception score employs a pre-trained deep learning neural network model. Specifically t he Inception v3 model, as defined by Christian Szegedy et al. The inception score gets its name from its dependence on the inception model. The model is used to classify a huge number of generated images. The likelihood of an image class labels is forecasted in detail. The inception score is created by combining these predictions. The goal of the score is to capture two characteristics of a set of produced images: Image Quality and Image Diversity.

The inception score has a lowest value of 1.0 and a highest value of the number of classes supported by the classification model; in this case, the Inception v3 model supports the 10 classes of the Fashion-MNIST dataset, and as such, the highest inception score on this dataset is 10.

I have computed the inception score for DCGAN and CapsNet GAN models for the Fashion-MNIST dataset.

MODEL	INCEPTION SCORE
DCGAN	4.37
CapsNet GAN	4.32

Table 2: Inception Score

Higher the inception score, better the quality of images generated. According to the results obtained, DCGAN produces better quality images when compared to CapsNet GAN for Fashion-MNIST dataset.

8.2.2. Generative Adversarial Metric

The generative adversarial metric (GAM) was introduced by Im et al. [33] as a pairwise comparison measure across GAN models by putting each generator against the opponent's discriminator, i.e., given two GAN models M1 = (G1,D1) and M2 = (G2,D2), G1 fights D2 while G2 fights D1. r_{test} and $r_{samples}$ are the ratios of their classification mistakes on a genuine test dataset and on generated samples. To prevent numerical difficulties, the ratios of classification accuracies are determined instead of errors following their implementation[15], as indicated in equations in Figure 32. Both $r_{samples} < 1$ and $r_{test} \simeq 1$ must be met for CapsuleGAN to win over DCGAN. On the Fashion-MNIST dataset, we achieve $r_{samples} = 1.0$ and $r_{test} = 0.89$ in our trials. Thus, on

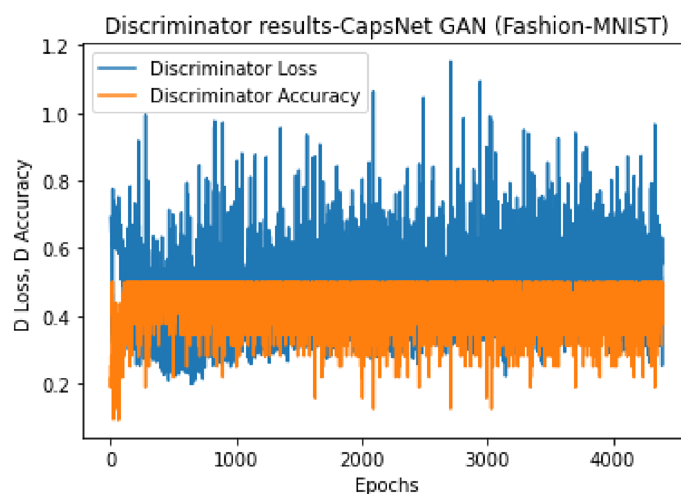
the Fashion-MNIST dataset, CapsuleGAN does not outperform DCGAN on this metric; rather, the two models tie on the dataset.

$$r_{samples} = \frac{A(D_{GAN}(G_{CapsuleGAN}(\mathbf{z})))}{A(D_{CapsuleGAN}(G_{GAN}(\mathbf{z})))}$$

$$r_{test} = \frac{A(D_{GAN}(\mathbf{x_{test}}))}{A(D_{CapsuleGAN}(\mathbf{x_{test}}))}$$

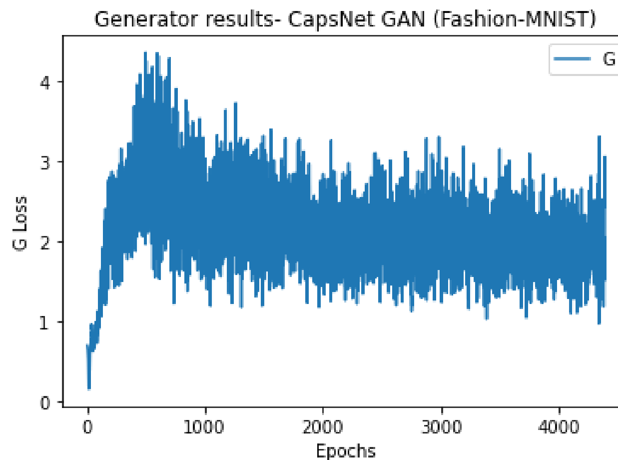
Fig 32: Equations for sample ratio and test ratio

8.2.3. Visualisation of loss and accuracy during CapsNet GAN training



Graph 1: CapsNet GAN model discriminator loss and accuracy values across 4000 epochs on Fashion-MNIST dataset

Figure 33 depicts the relationship between discriminator loss and accuracy. As its numbers do not reflect real progress, accuracy is shown inverted in relation to loss. The discriminator loss does not appear to be stable and continues to fluctuate. Accuracy does not reveal much since, despite study on the subject[12], there is no obvious ideal in practice. The discriminator's accuracy value simply cannot be translated to the output image quality. Throughout the training process, however, it is common for the accuracy score to stay around 40-75 percent.



Graph 2: CapsNet GAN model generator loss values across 4000 epochs on Fashion-MNIST dataset

Figure 34 depicts the generator's losses during training on the Fashion-MNIST dataset. The model's loss is fluctuating over all 4000 epochs, however it clearly lowers with time from 4.0. Although the pattern does not appear to depart the bounds of 3.0 and 1.5, which may be considered flattening, it is difficult to identify the flattening because it still varies a lot. These variations can be considered as evidence that the model is attempting to improve. Loss flattening would typically indicate that the model has reached a point where it can no longer improve, implying that it has learnt to translate the noise vector Z to a representation that is sufficient to challenge the discriminator.

9. DISCUSSION AND FUTURE WORK

GANs are famously difficult to train. As we have two different networks fighting each other, we need to optimise them both so that the fight lasts as long as possible by not prioritising one of them. Even with an alternative Discriminator architecture, such as Capsules, the inner GAN difficulties are not resolved; instead, issues relating to Convolutions are addressed. Research is being actively conducted towards further improving and making the training easier and more stable.

GANs have a wide range of applications, and there is still much to learn about them. Recently, GANs have achieved significant advances in Computer Vision, including picture inpainting, style transfer, image improvement, and so on. Despite this, new studies and enhancements to current models are published on a weekly basis. In this research, I used the Fashion-MNIST dataset to compare the original vanilla GAN with such enhancements as the powerful DCGAN and the CapsNet GAN for apparel image generation.

Using a more complicated dataset, integrating capsule network just in the generator, and using capsule network in both the generator and discriminator are some of the proposals for further study on this topic.

10. CONCLUSION

In conclusion, I demonstrated and described the features of the vanilla GAN, DCGAN, and the CapsNet GAN model, which was created by integrating Capsules into a DCGAN. I also went through the present bottlenecks of CNNs for modern Computer Vision applications, as well as a thorough discussion of Capsule Networks, their many layers, and the fundamental Routing algorithm that underpins them. I also presented a qualitative and quantitative comparison of the models which included GAM and IS which were used to explain the results of experiments. Although CapsNet GAN produced issues comparable to DCGAN, it was unable to outperform it in image generation for the Fashion-MNIST dataset.Vanilla GAN generates relatively less image quality. Although the vanilla GAN brings us to conclusion that DCGAN and CapsNet GAN are more powerful, DCGAN outperforming CapsNet GAN does highlight that the Capsule Network and DCGAN combination may not yet be fully robust.

REFERENCES

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. Departement d'informatique et de recherche op´ erationnelle. June 2014.

[2] Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434.

[3] Chapagain, Ashutosh. (2019). DCGAN--Image Generation. 10.13140/RG.2.2.23087.79523.

[4] Max Pechyonkin. Understanding Hinton's Capsule Networks. Part I: Intuition. Medium blog. *Nov. 2017.* Web link: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuiti on-b4b559d1159b

[5] Jonathan Hui. Understanding Dynamic Routing

between Capsules (Capsule Networks). Blog. Nov. 2017. Web link: https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules

[6] Max Pechyonkin. Understanding Hinton's Capsule Networks. Understanding Hinton's Capsule Networks. Part II: How Capsules Work. Nov. 201 . Web link: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-ii-how-c apsules-work-153b6ade9f66

[7] G. E. Hinton, A. Krizhevsky, S. D. Wang. Transforming Auto-encoders. Department of Computer Science, University of Toronto. 2011.

[8] S. Sabour, N. Frosst and G. E. Hinton, "Dynamic routing between capsules", Neural Information Processing Systems (NIPS), pp. 3859-3869, Dec. 2017.

[9] H. Gadirov, M. Tamošiūnaité and D. Vitkute-Adzgauskiene, "Capsule architecture as a discriminator in generative adversarial networks", Vytautas Magnus University, M. D. thesis, Feb. 2018.

[10] H. Xiao, K. Rasul amd R. Vollgraf, "Fashion-MNIST: A novel image dataset ", Computer Vision and Pattern Recognition (CVPR), Aug. 2017.

[11]Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved Techniques for Training GANs. OpenAI. June 2016 .

[12] Martin Arjovsky, Leon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. Courant Institute of Mathematical Sciences, Facebook AI Research. Jan 2017.

[13] Daniel Jiwoong Im, Roland Memisevic, Chris Dongjoo Kim, Hui Jiang. Generative Adversarial Metric. Montreal Institute for Learning Algorithms, Department of Engineering and Computer Science. 2016.

[14] Zhiming Zhou, Weinan Zhang, Jun Wang. Inception Score, Label Smoothing, Gradient Vanishing and -log(D(x)) Alternative. Shanghai Jiao Tong University. Aug. 2017.

[15] GAM iplementation-https://github.com/jiwoongim/GRAN/battle.py

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. Google Inc., University College London. Dec. 2015.

[17] Andrej Karpaty. CS231n Convolutional Neural Networks for Visual Recognition course. Stanford University. 2017.

[18] Barsim, K. S., Yang, L. and Yang, B. (2018). Selective sampling and mixture models in gener- ative adversarial networks, arXiv preprint arXiv:1802.01568 .

[19] Eshwar, S., Rishikesh, A., Charan, N., Umadevi, V. et al. (2016). Apparel classification using convolutional neural networks, 2016 International Conference on ICT in Business Industry & Government (ICTBIG), IEEE, pp. 1–5.

[20] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye," A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications," arXiv preprint arXiv: 2001.06937, 2020.

[21] Z. Wang, Q. She, and T.E. Ward, "Generative Adversarial Networks: A Survey and Taxonomy," arXiv preprint arXiv: 1906.01529, 2019.

[22] Z. Pan, W. Yu, X. Yi1, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (GAN): A survey," IEEE Access, vol. 7, pp. 36322–36333, 2019.

[23] S. Hitawala, "Comparative study on generative adversarial networks,"arXiv preprint arXiv: 1801.04271, 2018.

[24] Im, D.J., Kim, C.D., Jiang, H., Memisevic, R.: Generating images with recurrent adversarial networks. CoRR abs/1602.05110 (2016).

[25] Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167 (2015).

[26] Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: ICML (2013).

[27]Kingma, D., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014).

[28] Yadav, A., Shah, S., Xu, Z., Jacobs, D., & Goldstein, T. (2018). Stabilizing adversarial nets with prediction methods. In 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings.

[29] A Krizhevsky, I Sutskever, GE Hinton, "Imagenet classification with deep convolutional neural networks", Advances in neural information

processing systems. 2012.

[30] Shane Barratt and Rishi Sharma. A note on the inception score. arXiv preprint arXiv:1801.01973, 2018.

[31] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work, 2016.

[32] A. Jaiswal, W. AbdALmageed, Y. Wu and P. Natarajan, "CapsuleGAN: Generative adversarial capsule network", Brain Driven Computer Vision (BDCV), Oct. 2018.

[33] Im, D.J., Kim, C.D., Jiang, H., Memisevic, R.: Generative adversarial metric (2016)