STACKED CAPSULE AUTOENCODER BASED GENERATIVE ADVERSARIAL NETWORK

Galagama Arachchige Chatura. Madhusanka

(189386G)

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

October 2020

STACKED CAPSULE AUTOENCODER BASED GENERATIVE ADVERSARIAL NETWORK

Galagama Arachchige Chatura. Madhusanka

(189386G)

Thesis/Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

October 2020

Declaration

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

Name of Student	Signature of Student
G.A.C. Madhusanka	Date:

Supervised by	Signature of Supervisor
Dr. K.S.D. Fernando	Date:

Dedication

I dedicate this thesis to my parents and all my teachers.

Acknowledgement

I acknowledge my supervisor Dr. K.S.D. Fernando for her insight to make my research successful. Without the support from supervisor it would be hard to complete this project. I thank for her advices and techniques which helped me a lot in the process of developing this research.

Also I should thank to all the staff members of Department of Computational Mathematics including lecturers and non-academic staff.

Again I must thank to family members including my mother and farther for their huge support.

Abstract

Convolutional neural network based generative adversarial networks have become the dominant generative model in the field of generative deep learning. But limitations of convolutional neural networks affect generative adversarial networks also, since most of the current generative adversarial networks are based on convolutional neural networks. The main limitation of convolutional neural networks is that they are invariant. In other words, convolutional neural networks can't preserve spatial information of features in an image. In contrast, capsule networks gained attention in recent years due to their equivariant architecture which preserves spatial information.

Stacked capsule autoencoder is a type of capsule networks that is able to overcome the limitations that convolutional neural networks suffer from. Stacked capsule autoencoder is an equivariant model which preserves spatial, relational, geometrical information between parts and objects in an image. So in this research we implemented a generative adversarial network which uses stacked capsule autoencoder as the discriminator of it, by replacing the conventional convolutional neural network discriminator.

Then we evaluated the implementation of stacked capsule autoencoder based generative adversarial network using MNIST images. As the qualitative evaluation we observed the visual quality of generated images. Quality and diversity of the generated images are acceptable. Then we evaluated our model quantitatively using inception score for MNIST. Findings of this research show that, the stacked capsule autoencoder can be used as the discriminator of a generative adversarial network instead a convolutional neural network and its performances are plausible.

Table of Contents

List of Figures

List of Tables

Abbreviation

Convolutional Neural Network	CNN
Generative Adversarial Network	GAN
Deep Convolutional Generative Adversarial Network	DCGAN
Stacked Capsule Autoencoders	SCAE

Chapter 1

Introduction

1.1 Prolegomena

In 2014 at NIPS conference, Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow. Since then GANs play a significant role in generative deep learning. Initial version of GAN was based on the concept of game theory. Two artificial neural networks called Generator and Discriminator are competing with each other while improving themselves. This is similar to the fake currency printer and the police. Thief tries to fool the police and police try to discriminate fake currency. Due to the success of GANs, they have become the main approach for generative modeling. Even though in this paper we focus only on image generation, GANs are not limited to computer vision.

During the training phase, GAN tries to learn the probabilistic distribution of its training images. Then the GAN can generate new data using that learnt probabilistic model which represents the provided images. Currently there are huge number of GAN models and most of them are based on Convolutional neural networks (CNNs). But CNN based GANs only learn the presence probability of the parts in provided images. CNN based GANs usually ignore spatial information, special features of the parts and geometrical relationships between them.

In contrast, Capsule networks based GANs preserve spatial information such as orientation, rotation, scale etc. Capsule networks were introduced by Geoffrey Hinton in 2011, as an alternative to CNN. Capsule networks mimic the function of brain neurons better than CNNs. There are few versions of capsule networks and they have been used as discriminators in GANs instead CNN discriminators.

The latest version of capsule networks is called Stacked Capsule Autoencoders (SCAE). SCAE model achieved state of the art performance in unsupervised classification on MNIST data in 2019. Even though capsule networks based GANs can’t solve inner GAN issues, those may solve the issues which occur due to the limitations of CNNs. That’s why more research on combination of GANs and Capsule networks, is important for the growth of the field of generative modeling.

1.2 Aim and Objectives

The aim of this project is to see whether the Stacked Capsule Auto-encoder (SCAE) can be used as the discriminator of a Generative adversarial network instead a CNN discriminator and evaluate its performances with respect to an existing CNN based GAN.

Following are the objectives of this research

1. Critically review the literature in current researches for Capsule network based Generative adversarial networks.
2. Study in depth of Capsule networks, Generative adversarial networks and Capsule network based Generative Adversarial Networks.
3. Design and develop SCAE capsule network based Generative adversarial network.
4. Evaluate the developed SCAE capsule network based Generative adversarial network.

1.3 Background and Motivation

Success of the recent applications of Generative adversarial networks motivated us for this research. We also inspired by the theories behind Capsule networks which try to overcome from the limitations of conventional Convolutional neural networks. Using these powerful technologies together may address the limitations of CNNs and finally leads to solving the limitations of CNN based GANs which dominate the field of generative models at present. Among the existing capsule architectures, Stacked capsule encoders have shown state of the art performance on unsupervised classification of MNIST data. Note that the process and structure of the stacked capsule encoders are very much similar to neurons in brain than CNNs.

Solving the issues of CNN based GANs by addressing the issues of CNNs using capsule networks is important for GANs and its applications. More importantly the GAN applications in 3D domain and video generation will be heavily benefited by capsule network based GANs due to the main focus of those fields are on orientation changes of objects in an image which address effectively by capsule networks.

1.4 Problem in Brief

In short, the problem with CNNs is that they are able to detect and predict only the existence of features in an image. In other words, main problem with CNNs are that they are invariant. Invariance means relational information between features (orientation, pose, location etc.) of the image will be lost. This problem exists on every implementation which uses CNNs. As a result, CNN based GAN discriminators also have this problem.

1.5 Stacked Capsule Autoencoder based Generative Adversarial Network

Solution to the existing limitations of CNN based GANs, can be a capsule network based discriminator which preserves spatial information of the parts and objects in an image. SCAE is the newest type of capsule networks with many improvements than the previous capsule networks. So in this research we build a Generative adversarial network which use Stacked capsule auto-encoder (SCAE) as the discriminator of it, by replacing the conventional CNN discriminator.

1.6 Structure of the thesis

Rest of the thesis is structure as follows. Next two chapters describe others work. Chapter 2 critically reviews the domain of Capsule network based Generative Adversarial Networks including introduction to generative models, generative adversarial networks, capsule networks. We will highlight limitations of CNN based GANs which leads to our problem definition. In chapter 3, we study in depth of adapted technologies including generative adversarial networks, deep convolutional GAN, capsule networks and stacked capsule autoencoder.

From Chapter 4 we describe our work. Chapter 4 introduces our approach for design and implement the SCAE based GAN. Chapter 5 presents design of the SCAE based GAN in detail by explaining how the system works. Chapter 6 discusses about implementation of SCAE based GAN. Chapter 7 explains our evaluation strategy and displays the performances of the system. In chapter 8 we present our conclusions, limitations and future work.

1.7 Summary

This chapter described the big picture of the research project by introducing objectives, background, motivation, research problem in brief. In next chapter we critically review the domain of Capsule network based Generative Adversarial Networks including introduction to generative models, generative adversarial networks, capsule networks. We will highlight limitations of CNN based GANs which leads to our problem definition.

Chapter 2

Literature Review

2.1 Introduction

In previous chapter we discussed about the introduction. This chapter we do the critical review of related research in detail. Here we give an introduction about the domain of Capsule network based Generative Adversarial Networks, its early developments, its recent developments and the achievements up to now including introduction to generative models, generative adversarial networks, capsule networks. We will highlight limitations of CNN based GANs which leads to our define our research problem.

2.2 Deep Learning

The early developments of deep leaning were mainly dominated by discriminative models in terms of research, interest and usage. Discriminative models try to predict, based on the features of a dataset while generative models try to learn the probabilistic distribution of a dataset and search how the data has been generated. Main reason for the lack of research in the field of Generative models might be the benefits of the generative models were not visible to the stakeholders at that time. So in contrast to discriminative models, generative models were developed slowly. However at present the generative models have become a field of heavily being researched due to its interesting usages and promising future.

2.3 Generative models

Generative models are used to generate data with a probabilistic model which built by learning the structure of a dataset. Our research is mainly focused on Generative adversarial networks (GANs), which is a branch of generative models. Prior to GAN there have been various Generative models such as Autoencoders. Naive Bayes, Restricted Boltsman machines. These early generative models had drawbacks when the complexity of the data rises. It was hard for those early models to learn the features and structure of the data which they try to represent. Also those were not good at generating new samples which were not in the dataset.

To address the challenges which were not be able to solve by early generative models; two different generative models had been introduced. Those were Variational Autoencoders [1] and Generative adversarial networks [2] which were better alternatives to low performing early generative models.

Variational autoencoders solved the problems of vanilla autoencoder by adding randomness to the autoencoder and giving ability to manipulate the latent space. Variational autoencoders however based on the maximum likelihood technique which was similar to previous generative models. [3]. Maximum likelihood based generative models were not good at modeling high dimensional datasets and those models showed poor results when representing probabilistic distribution model of the training data. In contrast to the variational autoencoders; generative adversarial networks were more advanced generative models which were able to represent the probabilistic distribution model of the given dataset and generate better quality samples from it.

2.4 Generative Adversarial Networks

Introduction of generative adversarial networks by Ian Goodfellow has revolutionized the field of generative modeling and changed the motivation of deep learning research bit more towards the generative modeling. Even though generative adversarial network can be used with many data types such as audio, video, text, here we only discuss about images.

Even though, GANs were better than previous generative models, the initial version of GAN (vanilla GAN) had several issues such as unstable training, non-convergence and mode collapsing. Vanilla GAN was not performing well with complex datasets such as CIFAR-10. To tackle these issues, lots of research has been done with related to generator, discriminator and loss functions. As a result various types of GANs have been invented later.

As an example the Wasserstein-GAN introduced a new loss function by removing the cross entropy loss of vanilla GAN [4]. As a result, training of the GAN was more stable and the convergence of the generator was improved. Further versions such as Deep convolutional GAN (DCGAN) [5], Unrolled-GAN [6], Conditional-GAN [7] and improved techniques for training GANs [8] have been introduced.

The paper called “improved techniques for training GANs” introduced new architectural suggestions and training methods while Unrolled-GAN introduced a new training method which solved mode collapsing problem. Conditional-GAN was able to generate images under conditions. Note that we will only be discussing the versions of GANs which are related to our research such as DCGAN.

GANs are typically used for image generation. But those are also used in other applications such as image implanting, data augmentation, music and video generation, image translation and neural style transfer.

Even though current CNN based GANs are advance enough to generate quality images there are still challenges such as unstable training, mode collapsing, invariance and low performance on 3D domain. So Capsule networks based GANs might be better alternative to tackle these problems.

2.5 Capsule Networks

Capsule networks (CapsNets) were introduced as a transformation to the autoencoders by Geoffrey Hinton [9]. After the invention, it took some time to get the attention for capsule networks. In 2017 the paper called "Dynamic routing between capsule" achieved state of the art accuracy on MNIST dataset by outperforming Convolutional Neural Networks (CNNs) [10], [11]. Then in the next year, same research team introduced a new routing method called EM Routing [12]. They evaluated the new version and achieved state of the art results on smallNORB dataset [13].

The newest version of capsule networks named as Stacked Capsule Autoencoders (SCAE). It came up with radical changes to the architecture [14]. The main reason, why this is different from previous capsule architectures is that; this is an unsupervised model. SCAE achieved state of the art accuracy of 98.7% for unsupervised classification on MNIST dataset.

Even though capsule networks are still not performing well on complex image datasets compared to CNNs; capsule networks show a promising future due to the theory and architecture behind them. Capsule networks address many current drawbacks with CNNs such as invariance. Limitations of CNNs compared to capsule networks are described below.

2.6 Limitations of CNNs

Currently, CNNs are the most dominant artificial neural network architecture in the field of deep learning. Even though CNNs have achieved many things, those still have several drawbacks and limitations. Major drawback of a CNN is that it is unable to identify the pose of an image [15]. Here pose means the orientation (translational, rotational) and relationships between the parts in an image. Also the texture and distortion of an image can't be recognized by CNNs.

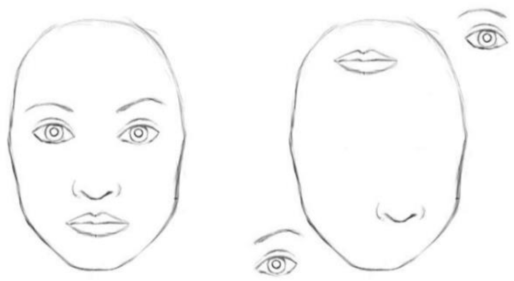


Figure 2.1: CNN can't differentiate these two images. But Capsule networks can.
(Source: Max Pechyonkin; understanding Hintons capsule networks)

In other words, CNNs are invariant. If the input for a CNN changed little bit, then the output will be same as the previous image. This means, CNNs are not sensitive to the position or orientation changes of objects in an image. This limitation of CNNs is shown in figure 2.1.

In contrast to CNNs; capsule networks are equivariant. This means, spatial position and orientation in an image are not ignored by capsule networks. Not only that but also they are sensitive to the spatial relationships between objects in an image. Capsule networks are able to preserve spatial information due to the architecture of those.

Due to these kind of drawbacks of CNNs; researchers are trying to apply capsule networks to different deep learning areas and problems which are currently implemented with CNNs. As an example, capsule networks based 3D shape processing [16] has achieved state of the art results in that category. Likewise researchers are being researching on capsule networks based GANs with the expectation of achieving better results than existing CNN based GANs.

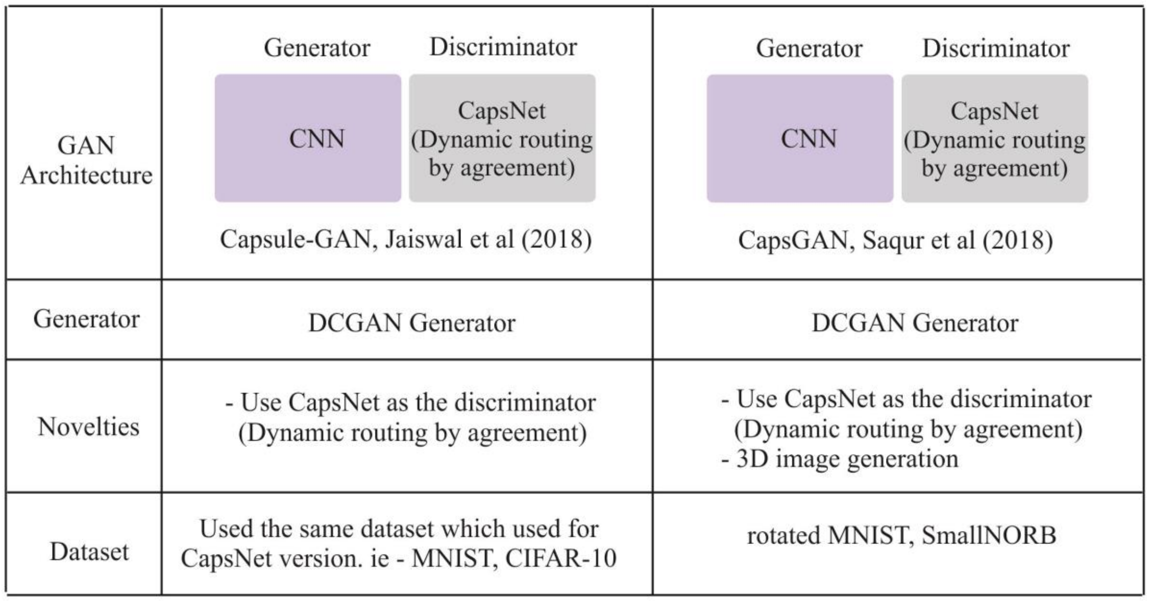
2.7 Capsule Generative Adversarial Networks

Still capsule networks are mainly being used as a discriminative model in the deep learning applications. So in the domain of GAN also, researchers have used capsule networks mainly as the discriminator of the GAN. First implementations of capsule network based GANs were also like that. They named these models as CapsuleGAN [17] and CapsGAN [18].

CapsuleGAN was introduced as a GAN which use a capsule network for its discriminator by replacing the CNN. They came up with an updated optimization function for training of the CapsuleGAN. They showed that the convolutional-GANs are outperformed by CapsuleGANs at semi-supervised classification with a large percentage of unlabeled generated images and rest with real labeled MNIST images. They used generative adversarial metric for their evaluation. They concluded that, the capsule network can be a better alternative to CNN, to use as discriminators in GANs.

CapsGAN also proposed a new method specially for 3D domain image generation which has geometrical transformations with high degree. They stated that capsule network based GAN show better results in 3D domain than CNN based GANs. For the evaluation they used rotated MNIST data to achieve geometric transformations first and then tested with SmallNORB dataset. Also they were able to balance stability and performance during the training of CapsGAN. For this they have used Wasserstein parameters such as penalty, clipping and spectral normalization. Finally they suggested that the capsule network based GANs will be a game changer in the field of 3D image generation and video generation.

Table 2.1 shows a comparison of capsule network based GANs which discussed above. As per the comparison we can observe that the both versions have used DCGAN generator as the generator of their capsule GAN. Both have used the same version of capsule networks called “Dynamic routing by agreement”. Novelties of the both papers are bit similar except the CapsGAN paper has extended the evaluation for 3D image generation.



GAN Architecture	Generator: CNN; Discriminator: CapsNet (Dynamic routing by agreement) Capsule-GAN, Jaiswal et al (2018)	Generator: CNN; Discriminator: CapsNet (Dynamic routing by agreement) CapsGAN, Saqur et al (2018)
Generator	DCGAN Generator	DCGAN Generator
Novelties	- Use CapsNet as the discriminator (Dynamic routing by agreement)	- Use CapsNet as the discriminator (Dynamic routing by agreement) - 3D image generation
Dataset	Used the same dataset which used for CapsNet version. ie - MNIST, CIFAR-10	rotated MNIST, SmallNORB

Table 2.1: Comparison of capsule network based GANs.

However still there are no any capsule network based GAN implementation, which use new capsule versions such as EM Routing and Stacked Capsule Autoencoders (SCAE). From those two, the SCAE would be a better choice, since it is highly advanced when compared to the previous capsule architectures.

2.8 Problem Definition

In summary, the problem with CNNs is that they are able to detect and predict only the existence of features in an image. In other words, the main problem with CNNs is that they are invariant. Invariance means relational information between features (orientation, pose, location etc.) of the image will be lost. This problem exists on every implementation which uses CNNs. As a result, CNN based GAN discriminators also have this problem.

Solution to this can be a capsule network based discriminator which preserves spatial information. Already there are capsule networks based GANs, but all of them are based on the architecture called “Dynamic routing by agreement”. This version of capsule network uses iterative routing. But later it was found that iterative routing is inefficient [19], [20], [21]. Also in that version, only the parts predict the parent object but not vice versa. Also it was a form of supervised learning.

SCAE is the newest type of capsule networks with many improvements than the previous capsule networks. In contrast to “Dynamic routing by agreement”; in Stacked Capsule Autoencoders; it uses objects in an image to predict parts which is more efficient and therefore it can get rid of iterative routing at inference time which is computationally intensive. Also since the SCAE is trained using unsupervised learning, it does not need labeled data.

Many of the previous capsule networks have been tested with GANs but not with SCAE. If we can use SCAE as the discriminator of a GAN instead a CNN it would be come under different type of research such as, integrate some known solutions to see whether those work well than existing solutions. Or it could be a type of research which doing something differently.

So our main research question is that; “Can Stacked Capsule Auto-encoder (SCAE) be used as the discriminator of a Generative adversarial network instead a CNN discriminator and what will be the performance of it?” Our research is unique because particular technologies have not been use together before.

2.9 Summary

This chapter analyzed the past researches and current trends in the field of GANs, Capsule networks and capsule network based GANs. Also we discussed about the limitations of CNNs and problems of CNN based GANs which motivated us for capsule network based GANs. So we defined our research problem as Can we build a new capsule architecture based GAN. Also we discussed the approaches which are taken to build the similar capsule based GANs which gave an insight to build our version of GAN. In the next chapter, we will discuss about the adapted technologies that we are going to use for our new type of capsule architecture based GAN.

Chapter 3

Technology Adapted

3.1 Introduction

In previous chapter we presented the major technologies associated with the research in detail. This chapter presents the major technologies associated with the research in detail such as Deep Convolutional GAN and Stacked Capsule Autoencoder capsule network.

3.2 Architecture of Generative Adversarial Networks

Architecture of the generative adversarial network is as follows. It has two artificial neural networks called Generator and Discriminator. Generator generates fake images staring from noise input by learning the probability distribution of the provided training data. Discriminator classifies fake images from generator and real images. So generator's goal is to fool the discriminator by generating high quality fake images which look similar to real images. By penalizing the generator when the discriminator identifies a generated image as a fake image; it gradually learns the probability distribution of the given dataset and improves the quality of the generated images. In contrast the discriminator will be penalized if it classify generated images as real images or real images from the training data as fake images. So this is similar to a min-max game between generator and discriminator.

3.3 Deep Convolutional Generative Adversarial Network (DCGAN)

DCGAN consists with convolution neural networks for both discriminator and generator. Architecture of DCGAN is shown in figure 3.1. It doesn't have max pooling layers and those layers have been replaced by strides. Also upsampling is done with transpose convolution and it has no fully-connected layers. This architecture showed more stable training of GANs even for the complex datasets such as human faces.

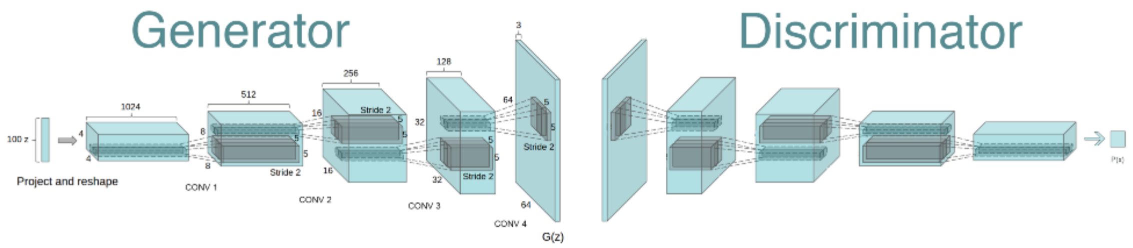


Figure 3.1: DCGAN architecture. This architecture has no max pooling layers and replaced by strides. Also upsampling is done with transpose convolution and it has no fully-connected layers. (Source: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks – A. Radford et al, 2016)

3.4 Architecture of Capsule Networks

Architecture of Capsule Networks is as follows. A capsule contains several neurons. Some parameters of features in an image and likelihood of those features are captured by these capsules. These captured information are then encapsulated into a vector. But in CNNs features captured from an image encapsulated into a scalar. Likelihood of an object existence is invariant while the instantiation parameters (spatial information) are equivariant. This means that the likelihood doesn't change with view point, position, orientation or rotation, but instantiation parameters do. As a result the output feature vector is changed when the spatial information changes [15]. Architecture of capsule networks is shown in figure 3.2.

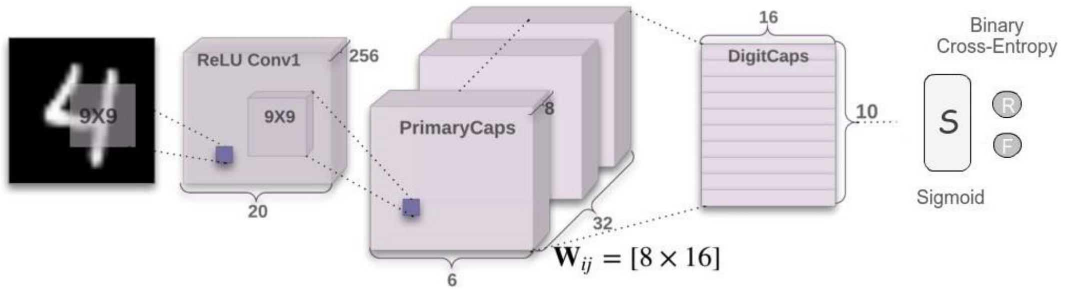


Figure 3.2: Architecture of capsule networks. (Source : Dynamic routing between capsules: Sabour et al, 2017)

As we stated before a capsule consist with probability of a feature as the length of the output feature vector and the spatial state of the feature as the direction of that vector. So if an object has changes its orientation, rotation or position on the image or its spatial state changes, the vector length which represents the probability of presence of the object does not change, but its direction which represents the spatial state, changes.

Then these capsules pass their vectors to the next level capsules and relevant weight matrices are used to multiply with it. This way it can preserve spatial information and relationships between objects in the image. This operation is similar to forward pass of a multilayer perceptron. Even though in conventional artificial neural networks weights are learned using backpropagation, in capsule networks weights are learned using a routing algorithm such as dynamic routing.

3.5 Stacked Capsule Autoencoders (SCAE)

Stacked Capsule Autoencoder is the latest architecture of capsule networks. It is an unsupervised model which can be trained on unlabeled data. SCAE achieved state of the art accuracy for unsupervised classification on MNIST dataset. SCAE works well for viewpoint changing scenarios since it preserves geometric relationships among parts and spatial relationships among objects. Object capsule vectors which represent presence likelihood; build clusters (Figure 3.3) in an unsupervised way by constructing images again.

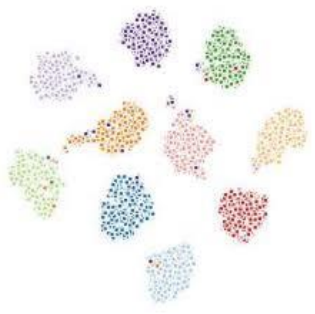


Figure 3.3: SCAEs can learn object classes with their respective object capsules which represent their likelihood in an unsupervised manner. This is the TSNE plot of presence probabilities of object capsule for each digit of MNIST. (Source: Kosiorek et al 2019)

Stacked Capsule Autoencoder has two parts which are Part Capsule Autoencoder (PCAE) and Object Capsule Autoencoder (OCAE). PCAE breakdowns an image into parts and learns their presence and poses. Then it construct the image again using part templates which undergoes through affine transformation. On the other hand, OCAE construct objects using the parts and poses from PCAE. After constructing objects, it tries to predict pose of parts in each object. OCAE has object capsules with parameters which are used to reconstruct the object. In short, at PCAE stage it learns parts and poses and at OCAE stage it uses same parts and poses to construct the object again. Summary of this process is shown in figure 3.4.

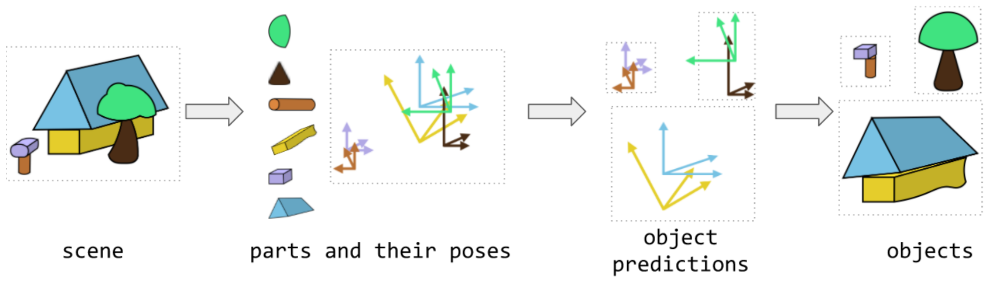


Figure 3.4: At PCAE stage it learns parts and poses and at OCAE stage it uses same parts and poses to construct the object again. (Source : http://akosiorek.github.io)

PCAE basically do two things. First it identifies parts in the image and then find out spatial relationships among them. This can be referred as autoencoding. A part capsule store presence of the part, spatial information such as translations, scale, rotation, shear and a unique identity. Encoder which is a CNN with attention; learns above mentioned features while decoder learns the template which represents the identity of a part capsule. Then this template is gone through affine transformation with pose information, if the part is present. Finally, the image is reconstructed using those templates based on Gaussian mixing probabilities. Summary of the process of PCAE is given by following equations.

$$\begin{aligned}
&\mathbf{x}_{1:M}, d_{1:M}, \mathbf{z}_{1:M} = \mathrm{h}^{\mathrm{enc}}(\mathbf{y}) && \text{predict part capsule parameters,}\\
&\mathbf{c}_m = \mathrm{MLP}(\mathbf{z}_m) && \text{predict the color of the m}^{\text{th}}\text{ template,}\\
&\widehat{T}_m = \mathrm{TransformImage}(T_m, \mathbf{x}_m) && \text{apply affine transforms to image templates,}\\
&p^y_{m,i,j} \propto d_m \widehat{T}^a_{m,i,j} && \text{compute mixing probabilities,}\\
&p(\mathbf{y}) = \prod_{i,j} \sum_{m=1}^{M} p^y_{m,i,j} \mathcal{N}\left(y_{i,j} \mid \mathbf{c}_m \cdot \widehat{T}^c_{m,i,j}; \sigma^2_y\right) && \text{calculate the image likelihood.}
\end{aligned}$$

Then to identify objects; OCAE use part capsules parameters such as pose, features and templates which output from PCAE. Encoder of the OCAE is a attention based Set Transformer [22] which get rid of absent points. For stability reasons, OCAE is trained using only the features learnt at PCAE stage. During the training, predictions about part pose combinations are given by object capsules. So by maximizing the likelihood of part pose, the OCAE infers about parts again. Part pose likelihood function is given by following equation.

$$p(\mathbf{x}_{1:M}, d_{1:M}) = \prod_{m=1}^{M} \left[\sum_{k=1}^{K} \frac{a_k a_{k,m}}{\sum_i a_i \sum_j a_{i,j}} p(\mathbf{x}_m \mid k, m) \right]^{d_m}$$

3.6 Summary

In this chapter we discussed about adapted technologies for our research. We are using Generative adversarial networks and Capsule networks. Specially we focused on Deep Convolutional GAN and Stacked Capsule Autoencoder. In the next chapter we discuss about approach for our research.

Chapter 4

Approach

4.1 Introduction

In previous chapter we presented the major technologies associated with the research in detail. In this chapter we present approach for our research. We will discuss about our hypothesis, inputs, outputs, process, features and users.

4.2 Hypothesis

The hypothesis of our research is that the capsule network architecture called stacked capsule autoencoder can be used as the discriminator of a generative adversarial network instead a CNN to generate images.

4.3 Inputs to the system

There are two inputs to the system. Those are MNIST images and uniform noise. We use MNIST data since the original SCAE paper also used MNIST to benchmark it. Uniform noise input has a length of 100 and it is the input for the Generator. Outputs of generator is the input for the discriminator. In other words, fake images generated by generator are inputs to the discriminator. There is another input for the discriminator. That is MNIST images.

MNIST data were taken from the database located at http://yann.lecun.com/exdb/mnist/. Dataset has 60000 training images and 10000 testing images. These are 28x28 pixel greyscale handwritten images. MNIST is the standard dataset for benchmarking a new computer vision model. This has 10 classes which are digits from 0 to 9. A sample of MNIST data is shown in figure 4.1.

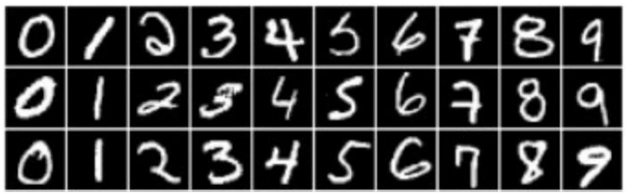


Figure 4.1: Sample of MNIST data

4.4 Outputs from the system

Output from the system is newly generated images. Quality of fake images generated by the generator will improve during the training and will look similar to the MNIST data.

4.5 Process

Image generation using the stacked capsule auto-encoder capsule architecture based generative adversarial network is the main process of the system.. Combined model will take MNIST images and random noise as inputs and outputs the generate images from the system.

4.6 Features

Following are the features of this system.

- The discriminator preserves not only the presence but also the spatial information of the image during its classification.

- Discriminator also uses objects in an image to predict its parts which is more efficient.
- Therefore SCAE discriminator can get rid of iterative routing at inference time which computationally intensive.
- Since discriminator of this GAN is trained using Unsupervised learning, it doesn't need labeled data.

4.7 Users

AI researchers, data scientists and AI students will be the uses of the system. They will use this system as a reference to study, use and improve capsule networks and generative adversarial networks in future.

4.8 Summary

In this chapter we discussed about our approach for our research. We also discussed about our hypothesis, inputs, outputs, process, features and users. In the next chapter we will discuss about Design.

.

Chapter 5

Design

5.1 Introduction

In previous chapter we discussed about our approach for our research. We also discussed about our hypothesis, inputs, outputs, process, features and users. In this chapter we present our design for the research in detail.

5.2 Architecture of proposed GAN

Most of the GANs invented up to now including previous capsule GANs, are based on a basic GAN architecture such as DCGAN but with some modifications. So our proposed GAN is also mainly based on the Deep Convolutional GAN. In our design we use Stacked Capsule Autoencoder network as the discriminator instead conventional CNN discriminator. We will use the same generator in DCGAN for our model which is a CNN. Also we use MNIST data since the original SCAE paper also used MNIST for evaluation.

Uniform noise is the input for the Generator. Fake images generated by generator are inputs to the discriminator. MNIST images are also input for the discriminator. SCAE discriminator of the proposed GAN will classify incoming images as real or fake. This is the main task of the discriminator of a GAN.

Goal of the generator is to fool the discriminator or increase the likelihood of misclassifications done by the discriminator by making its output similar to real MNIST images. Goal of the discriminator is to reach its accuracy around 50%. This means that the discriminator classifies output from the generator as real images. High level design diagram of proposed GAN is shown in figure 5.1.

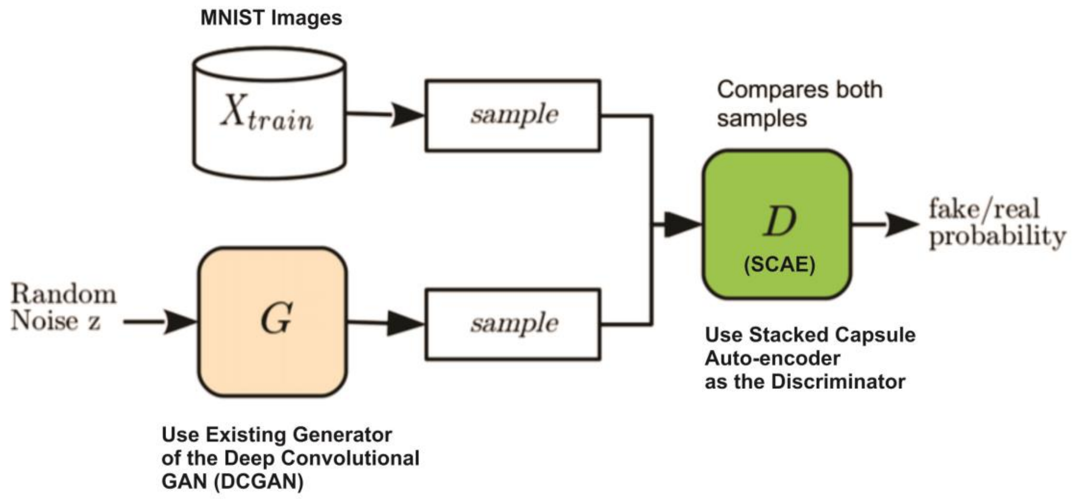

MNIST Images
X_{train}
sample
Compares both samples
D
(SCAE)
fake/real probability
Random Noise z
G
sample
Use Stacked Capsule Auto-encoder as the Discriminator
Use Existing Generator of the Deep Convolutional GAN (DCGAN)

Figure 5.1: High level design diagram of proposed GAN

A comparison of the architectures of previous capsule based GANs and proposed GAN is shown in table 5.1.

	Previous Capsule Network based GANs		**Proposed GAN**
GAN Architecture	Generator: CNN; Discriminator: CapsNet (Routing by agreement) Capsule-GAN, Jaiswal et al (2018)	Generator: CNN; Discriminator: CapsNet (Routing by agreement) CapsGAN, Saqur et al (2018)	Generator: CNN; Discriminator: SCAE
Generator	DCGAN Generator	DCGAN Generator	DCGAN Generator
Novelties	- Use CapsNet as the discriminator (Dynamic routing by agreement)	- Use CapsNet as the discriminator (Dynamic routing by agreement) - 3D image generation	- Use Stacked Capsule Autoencoder as the discriminator
Dataset	Used the same dataset which used for CapsNet version. ie - MNIST, CIFAR-10	rotated MNIST, SmallNORB	Used the same dataset which used for CapsNet version. ie - MNIST

Table 5.1: Comparison of the architectures of previous capsule based GANs and proposed GAN

5.3 Architecture of Generator

Here we use the same generator used in DCGAN. As mentioned earlier uniform noise Z which has length of 100 is the input for the Generator. Architecture of the generator is as follows. Generator has strided convolutional or deconvolutional layers. First layer is a fully connected layer which does matrix multiplication. This is the only fully connected layer which is used and no any fully connected layers in deeper levels. The output is reshaped and use as the input to the deconvolutional layers. So the second layer is a deconvolutional layer. Except the output RELU activation function is used for all layers.

Batch normalization has been used between layers except the output layer. Stable training of the generator can be achieved by applying batch normalization with mean equal to 0 and variance equal to 1 [23]. Batch normalization also a solution for mode collapsing which is a common problem with GANs during training. However batch normalization is not applied for output layer to avoid instability.

2D Upsamplings have also been used in subsequent layers. These increase the size of the image by duplicating rows and columns. Also no any pooling layers in the generator. For the final output layer a Tanh activation function is used. Since discriminator takes 28x28 pixel input, the generator outputs 28x28 tensor. In that way the uniform noise given to the generator with the length of 100, is transformed into a 28x28 tensor. Architecture of the generator is shown in figure 5.2.

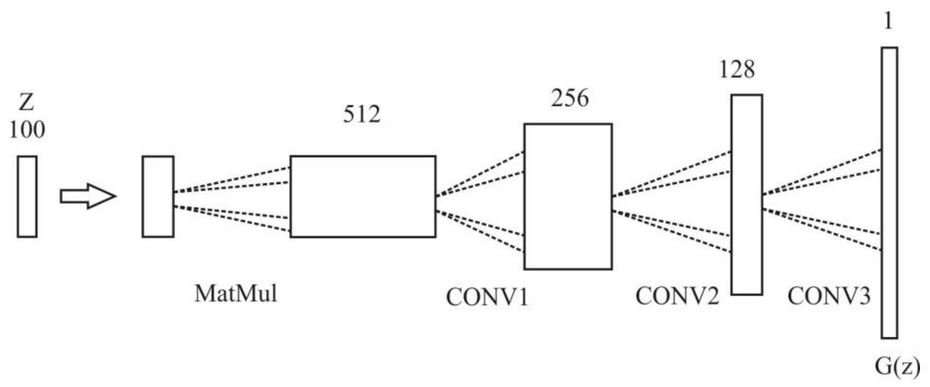


Figure 5.2: Architecture of the Generator

5.4 Architecture of Discriminator

For the discriminator we use the capsule network called Stacked Capsule Autoencoders, as stated in previous chapters. Greyscale 28x28 pixel images are the inputs for the discriminator. These inputs include both real MNIST images from the database and also generated fake images from generator. Architecture of the discriminator is as follows. As we mentioned in Technology Adapted chapter, SCAE has two stages called Part

Capsule Autoencoder (PCAE) and Object Capsule Autoencoder (OCAE). Both stages have encoder and decoder.

5.5 Part Capsule Autoencoder (PCAE)

At first stage, image is passed through the PCAEs encoder. PCAE encoder is basically an attention based CNN which we called as part (primary) capsules. This breakdowns an image into parts and learns their presence, poses and special features. As the first step, image is passed through a CNN encoder and it returns image embedding. Then attention base pooling applied to the image embedding. Attention base pooling is done by a convolution layer followed by reshaping, softmax activation and reshaping again. This returns pose, feature and presence logits. By passing through a softmax activation it gives presence of the part. After applying geometric transformation for pose which includes translations, scale, rotation, shear; PCAE encoder returns pose, feature and presence of parts.

Then comes the template based primary capsule decoder for images. This reconstructs the image again using learned part templates. Decoder makes templates which represents the identity of part capsules based on pose and features returned by part capsule encoder. Then these templates are gone through affine transformation with pose information, if the part is present. Finally, the image is reconstructed using those templates based on computed Gaussian mixing probabilities.

Summary of the process of PCAE is given by following steps.

- Predict parameters of parts such as presence, pose and special features.
- Make image templates.
- Affine transformation of image templates.
- Compute Gaussian mixing probabilities.
- Calculate image likelihood.

5.6 Object Capsule Autoencoder (OCAE)

pose, presence of the parts and image templates which made by part decoder are then passed to the OCAE encoder. OCAE encoder is an attention based permutation invariant Set Transformer which get rid of absent points. This is a neural network based encoder and decoder which reduce computations by model interactions between pose, presence and image templates.

Output tensor of the Set Transformer is then passed to OCAE decoder. OCAE decoder has object capsules which use part capsule parameters (pose, features and templates) to reconstruct the object. During the training, predictions about part pose combinations are given by object capsules. So by maximizing the part pose likelihood, OCAE predicts pose of parts in each object and reconstruct the object again.

Finally there is a two class classification task to determine whether the image is real or fake. It outputs the cross entropy loss and the loss will be added to SCAE loss function. SCAE loss adds up mixture probability, sparsity loss, cross entropy loss, dynamic weights and deducts the log probability of the model after multiplied by predetermined weights. Architecture of SCAE discriminator is shown in Figure 5.3.

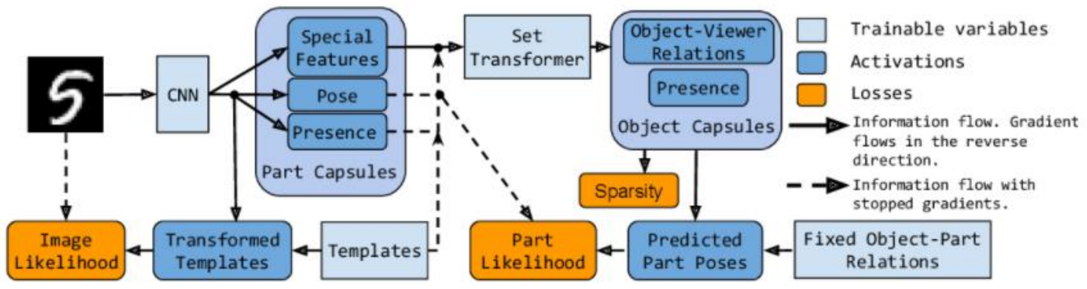


Figure 5.3: SCAE discriminator architecture. (Source: Kosiorek et al 2019)

As we mentioned it above, the discriminator is built using SCAE capsule network with some additional modifications such as changing the number of classes for classification and changing the optimization function.

5.7 Summary

In this chapter we discussed about our methodology and design. As mentioned above we use DCGAN’s Generator and SCAE as discriminator for our GAN. In the next chapter we discuss about Implementation.

Chapter 6

Implementation

6.1 Introduction

In previous chapter we presented the design of our research. In this chapter we present how do we implement our design in detail. We will discuss about data preparation, generator and discriminator implementation and training of the model.

6.2 Dataset preparation

As the first implementation step what we do is load and prepare dataset. We load the 28x28 MNIST dataset from tensorflow dataset repository and take only the training images. Then images converts to type float and then divided by 255.0 to normalize the MNIST images. Then using a iterator we provide batches with the size of 64 for the training of our GAN.

```
dataset = tfds.load(name='mnist').repeat().batch(batch_size)
…
data['image'] = tf.to_float(data['image']) / 255.
…
  input_batch = dataset.make_one_shot_iterator().get_next()
```

Then we generate the uniform noise (z) which is 100 in length (z_shape) using numpy library. Nose values are randomly vary between -1 to 1 and their type will be float32. Similar to MNIST using a iterator again we provide batches of noise with the size of 64.

```
yield np.random.uniform(-1, 1, (batch_size, z_shape))
dataset = tf.data.Dataset.from_generator(noise_generator)
…
input_noise_batch = dataset.make_one_shot_iterator().get_next()
```

6.3 Generator implementation

Generator is built using tensorflow framework. As mentioned in design chapter, the uniform noise Z which has a shape of 100 is the input for the Generator. First layer is a fully connected layer which does matrix multiplication. Then we use a RELU activation for the outputs of first layer Then the output is reshaped and gone through 2D Upsamplings. Then comes the first deconvolutional layer followed by batch normalization with momentum.

These steps repeat once again by replacing RELU activation with Leaky RELU and without reshaping. After this Leaky RELU applies again and goes through the final deconvolutional layer followed by Tanh activation function. In that way the uniform noise input for the generator, is transformed into a 28x28 tensor.

```
z = tf.matmul(X, self.W1)
z = tf.nn.relu(z)
z = tf.reshape(z, [-1, 7, 7, 128])
z = UpSampling2D()(z)
z = tf.nn.conv2d(z, self.W2, [1, 1, 1, 1], padding="SAME")
z = batch_normalization(z, momentum=momentum)
…
z = tf.nn.conv2d(z, self.W4, [1, 1, 1, 1], padding="SAME")
z = tf.nn.tanh(z)
```

Initialization of generator weights is done using random normal distribution, with 0.02 standard deviation. Biases are initialized as zeros.

```
W1=tf.Variable(tf.random_normal(shape=[100, 7*7*128], stddev=0.02))
…
W4=tf.Variable(tf.random_normal(shape=[3, 3, 32, 1], stddev=0.02))
```

6.4 Discriminator implementation

SCAE discriminator is built using tensorflow and library called Sonnet. SCAE discriminator takes greyscale 28x28 pixel images as the input. As we mentioned in Technology Adapted chapter, SCAE has two stages called PCAE and OCAE.

6.5 Part Capsule Autoencoder (PCAE) implementation

First, unlabeled images are passed through the PCAEs encoder which is an attention based CNN. Inside this there is a CNN encoder and it returns image embedding. Then attention base pooling applied to the image embedding. Attention base pooling is done by a convolution layer followed by reshaping, softmax activation and reshaping again. By passing through a softmax activation it gives presence of the part. After applying geometric transformation, it outputs primary capsules (ie - pose, feature, presence).

```
cnn_encoder = snt.nets.ConvNet2D(output_channels=[128] * 4,
kernel_shapes=[3], strides=[2, 2, 1, 1], paddings=[snt.VALID],
activate_final=True)
…
img_embedding = self.cnn_encoder (x)
…
h = snt.AddBias(bias_dims=[1, 2, 3])(img_embedding)
…
h = snt.Conv2D(n_dims * self._n_caps + self._n_caps, 1, 1)(h)
…
pose, feature, pres_logit = tf.split(h, splits, -1)
…
pres = tf.nn.sigmoid(pres_logit)
pose = utils.geometric_transform(pose, transform_to_matrix=False)
return pose, feature, pres
```

Then comes the template based primary capsule decoder. Decoder makes templates which represents the identity of part capsules.

```
template_shape=([1,n_templates]+list(self.template_size)+[n_dims])
…
template_logits = tf.get_variable('templates', initializer=q)
self._template_logits = template_logits
```

```
…
self._templates = template_nonlin(template_logits)
  …
if template_feature is not None: # primary_caps.feature
  # Whether to infer template color from input.
 …
  template_color = mlp(template_feature)[:,:,tf.newaxis,tf.newaxis]
…
templates = tf.identity(templates) * template_color
```

Then these templates a gone through affine transformation with pose information, if the part is present. Finally, the image is reconstructed using those templates based on computed Gaussian mixing probabilities.

```
transformed_templates = resampler(templates, grid_coords)
…
transformed_templates = tf.concat([transformed_templates,bg_image],1)
…
mixing_logits = template_mixing_logits #Mixture Prob Distribution
…
mixing_log_prob = mixing_logits-tf.reduce_logsumexp(mixing_logits, 1)
…
log_prob = distributions.log_prob(target_x)
  rec_ll_per_pixel = tf.reduce_logsumexp(log_prob + mixing_log_prob, 1)
```

6.6 Object Capsule Autoencoder (OCAE) implementation

Pose, presence, features and templates produced by PCAE are the inputs to the OCAE encoder which is Set Transformer. Set Transformer code is borrowed from the official repository [https://github.com/juho-lee/set_transformer].

```
class SetTransformer(snt.AbstractModule):#Permutation-invariant Trans
 …
class QKVAttention(snt.AbstractModule): #Trans-like self-attention
   …
class MultiHeadQKVAttention(snt.AbstractModule): #Multi-head version
   …
class SelfAttention(snt.AbstractModule):
…
```

Output tensor of the Set Transformer is then passed to OCAE decoder. OCAE decoder has object capsules and by maximizing the part pose likelihood, OCAE predicts pose of parts in each object to reconstruct the object again.

```
capsule = CapsuleLayer(n_caps, n_caps_dims, n_votes, n_caps_params,
n_hiddens)# capsule layer
res = capsule(h) #h=output_tensor of SetTransformer
#res = AttrDict(votes, scale_per_vote, vote_presence)
…
likelihood = CapsuleLikelihood(votes, scale, vote_presence_prob)
ll_res = likelihood(target_pose, target_presence) # ll_res =
mixture_log_prob, vote_presenceposterior_mixing_probs
    …
```

Finally there is a two class classification task to determine whether the image is real or fake. It outputs the cross entropy loss and the loss will be added to SCAE loss function. SCAE loss adds up mixture probability, sparsity loss, cross entropy loss, dynamic weights and deducts the log probability of the model after multiplied by a predetermined weights.

```
linear_model = snt.Linear(self._n_classes)
logits = linear_model(tf.stop_gradient(features))
…
cross_entropy_loss = tf.reduce_mean(
tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=label))
return logits, cross_entropy_loss
…
loss = (- rec_ll - log_prob *1. +dynamic_weights_l2 *10 +
          + prior_cross_entropy_loss + prior_within_sparsity_loss *2.
          - prior_between_sparsity_loss*0.35
          + posterior_cls_cross_entropy_loss)
```

6.7 Training of GAN

Some of the training and model parameters are as follows.

```
batch_size = 64;
canvas_size = 28
lr_dcgan = 0.0001;
beta1 = 0.5;
lr = 3e-5;
epsilon = 1e-2 / float(batch_size) ** 2
```

Discriminator loss calculated by adding scae loss for both real and fake images. We also used RMSPropOptimizer optimizing function with momentum of 0.8, as per the original SCAE paper.

```
scae_loss = tf.add(res_fake.loss, res_real.loss)
opt_scae = tf.train.RMSPropOptimizer(lr, momentum=.8, epsilon=epsilon)
  …
```

We used AdamOptimizer optimizing function for generator with momentum of 0.5. Generator loss is calculated as follows.

```
gen_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
labels=labels, logits=logits))
  …
opt_gen = tf.train.AdamOptimizer(lr_dcgan, momentum=.5,
epsilon=epsilon)
  …
```

5.8 Summary

In this chapter we discussed about our methodology and design. As mentioned above we use DCGAN's Generator and SCAE discriminator. In the next chapter we discuss about results and evaluation.

Chapter 7

Evaluation

7.1 Introduction

In previous chapter we presented about how to implement the proposed GAN. In this chapter we present about how to evaluate our research.

7.2 Evaluation procedure

Implemented GAN was trained on Google Colab GPUs such as Tesla K80, T4. Training time depends on the available GPU type. We evaluate the performance of SCAE based GAN by generating images. Then we compare its results with Deep Convolutional GAN.

7.3 Visual quality evaluation of generated images

Here we visually analyze the quality of generated images. SCAE based GAN generate images similar to real MNIST data. By looking at generated images, most of the times, we can clearly see the digit. Diversity of the generated digits is also acceptable which means the mode collapsing does not occur with the model. Generated MNIST images by SCAE based GAN are shown in figure 7.1.

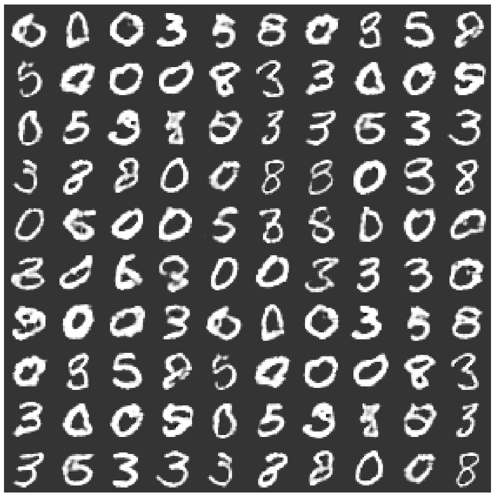


Figure 7.1: Generated MNIST images.

By observing the generated images qualitatively, we can say that the performance of the SCAE based GAN is qualitatively plausible.

7.4 Losses of the GAN

Since discriminator and generator are in a min-max game, when the generator is learning the discriminator's loss is increasing. Usually it is hard to measure the performance of a GAN using its metrics such as accuracies and losses unlike a discriminative model. As we can see, losses of both discriminator and generator are varying, even though those are converging with the number of epochs.

Generator loss is varying between around 0 to 10 if we ignore spikes. This kind of variation of loss can be observed in previous GAN models such as DCGAN. Usually it is hard to infer about the performance of the generator by observing its loss values. However since we can observe that the loss values are varying roughly between fixed values, we can infer that the GAN has achieved equilibrium between the min-max game of discriminator and generator. This means the generator can't improve further or it has learnt the probability distribution of the MNIST data.

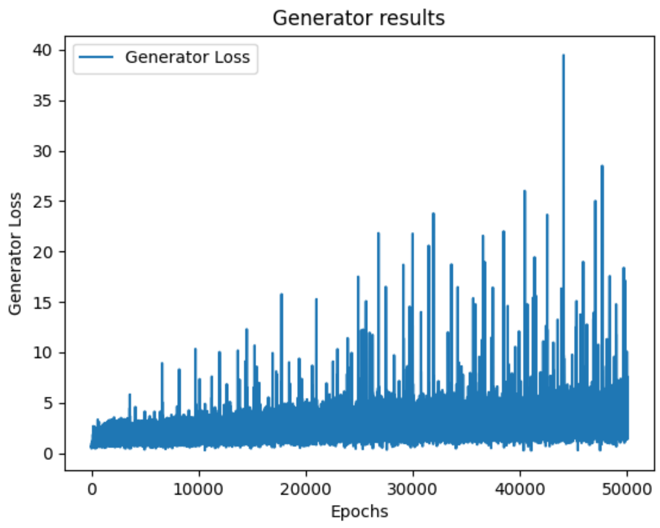


Figure 7.2: Generator loss

Discriminator loss is also varying. Similar to generator it is hard to infer about the performance of the discriminator by observing its loss values. As we can see the discriminator loss is increasing and varying.

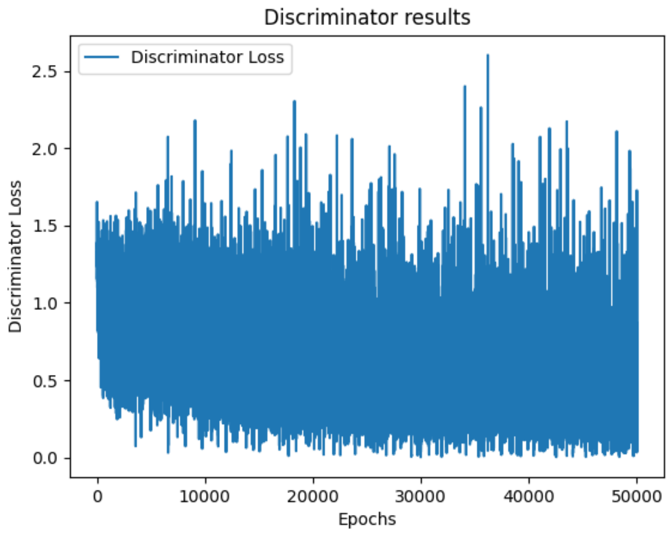


Figure 7.3: Discriminator loss

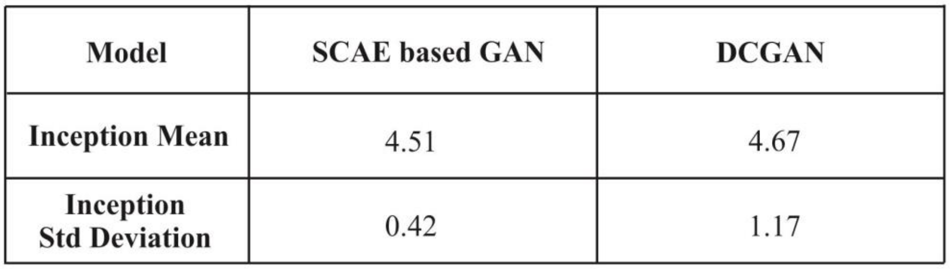
Theoretically at the equilibrium, discriminator accuracy has to be around 50% but in practice that does not happen with GANs. As per the early research, accuracy of a GAN also doesn't reflect the quality of the images in practice, similar to losses.

7.5 Inception Score for MNIST

Usually it is hard to measure the performance of a GAN qualitatively by human. To measure and compare the performances quantitatively we have to use a metric. There are several metrics such as Generative adversarial metric, Fréchet Inception Distance (FID), Inception score etc. Here, to compare the performances quantitatively we use Inception score. Inception score consider two things of the generated images to measure the performance of a GAN [24]. Those are;

- Quality or clearness of generated images,
- Diversity of generated images.

Inception score is calculated using the Inception v3 Network [25] which pre-trained on ImageNet [26]. Usually inception score is used to measure the quality of color images such as CIFAR-10. But here we use the pre-trained Resnet18 [27] which pre-trained on MNIST dataset [28]. Table 7.1 shows the comparison of inception scores for SCAE based GAN and DCGAN.



Model	SCAE based GAN	DCGAN
Inception Mean	4.51	4.67
Inception Std Deviation	0.42	1.17

Table 7.1: Inception means and standard deviations for SCAE based GAN and DCGAN

According to the comparison above, we can see that the SCAE based GANs performances are much similar to the performances of DCGAN since both have similar inception means.

7.6 Summary

In this chapter we discussed about results and evaluation procedure of our model. We also compared our SCAE based GAN with DCGAN. In the next chapter we discuss about conclusions, limitations and future work.

Chapter 8
Conclusion

8.1 Introduction

In the previous chapter we evaluated our research using suitable methods. In this chapter we present our conclusions based on the results of previous chapter. Also we state some limitations and future work of our research.

8.2 Conclusions

In this research we identified that, capsule networks could be better alternative for CNNs. This is because, capsule networks can preserve spatial information and special features of an image. Implementing and evaluating of existing deep learning architectures with capsule networks, which are previously based on CNNs, is a common research trend nowadays. Few researchers have attempted to build capsule based GANs recently, such as Capsule-GAN.

During our literature review, we identified that, even though capsule based GANs do not address current limitations and issues of GANs, they address some limitations of CNNs. So we conclude that, this approach can be used to address the limitations of CNN based GANs too.

With this idea in our mind, we implemented a SCAE capsule network based GAN model and evaluated its performances. For the implementation we used DCGANs generator and SCAE discriminator. During the final stages, training of the GAN was bit hard since capsule networks are powerful but not yet stable when they combined with CNNs. This is due to the non-similarity between two networks. So optimizing both together at the same time was bit difficult.

We used MNIST images for evaluation since the original SCAE version also evaluated using them. We evaluated our model qualitatively and quantitatively. As per the qualitative evaluation, we observed the visual quality of generated images. Quality of the generated images was acceptable. Diversity of the generated digits is also acceptable. This means mode collapsing did not occur with our model and the training was successful. In appendix B, we have done a qualitative comparison between DC GAN and SCAE based GAN generated images.

Despite the fact that, it is hard to measure the performance of a GAN using its metrics such as accuracies and losses unlike a discriminative model, we recorded those values. As we could see, losses of both discriminator and generator were varying, even though those are converging with the number of epochs. As per the results of our research, we could confirm that the accuracies and losses of a GAN don't reflect the quality of the images in practice.

To compare the performances quantitatively we used inception score metric for MNIST. According to the comparison of inception scores for SCAE based GAN and DCGAN, we could observe that, both have similar inception means. Based on this quantitative evaluation, we could see that the SCAE based GANs performances are similar to the performances of DCGAN. So according to both evaluation techniques, the results of the SCAE based GAN is plausible and it is somewhat similar to DCGAN.

We also achieved all the four objectives mentioned in introduction chapter. Because we critically reviewed the literature in current researches for Capsule network based Generative adversarial networks. We studied in depth of Capsule networks, Generative adversarial networks and Capsule network based Generative Adversarial Networks. Also we designed and developed SCAE capsule network based Generative adversarial network. Then we evaluated the developed SCAE based Generative adversarial network.

Finally we can conclude that, Stacked Capsule Auto-encoder (SCAE) can be used as the discriminator of a Generative adversarial network; instead a CNN discriminator. Also we observed that the performance of SCAE capsule network based Generative adversarial network is plausible. So, SCAE capsule network based Generative adversarial networks are better alternative to CNN based GANs since they address the limitations of the latter.

8.3 Limitations and Future Work

Capsule networks based GANs have the same drawbacks which have with capsule networks. Usually capsule networks take more time for the training, in contrast to CNNs. This is a considerable limitation in SCAE, than previous capsule architectures. The reason is that, SCAE does huge computations during its training.

Capsule networks based GANs have potential to become a game changer in the field of generative models. Goal of the capsule based GANs, is that they could address limitations of CNNs which finally leads to address limitations of CNN based GANs. In this research, we built our model as a benchmarking model. So we tested our model only with MNIST data.

Improving capsule based GANs to model complex datasets is an open area for future research. Especially our model could be extended for 3D image datasets such as rotated MNIST or smallNORB; since capsule networks are good with orientation changes.

Even though, here we experimented using only the MNIST data as a benchmarking dataset, we can suggest that our model could also be generalized to use with more diversified datasets such as CIFAR and fashion MNIST. Furthermore, it is possible to extend our model to work with very complex and diversified datasets such as human faces, fashion images and medical images etc.

Also improving the techniques for stable training of capsule based GANs would be a future research area. Implementation of SCAE based generator is also another research opportunity. Different GAN applications such as image implanting, video generation, image translation and neural style transfer can also be implemented with capsule based GANs in future.

8.4 Summary

In this chapter we discussed about our conclusion, limitations and future work. We concluded that our hypothesis is correct and we achieved all of our objectives. Also we discussed about limitations of SCAE based GAN and possible further improvements for the research. From the next page onwards we have mentioned references and appendices.

References

[1] Diederik P. Kingma , Danilo J. Rezende , Shakir Mohamed , Semi-supervised Learning with Deep Generative Models. Max Welling Machine Learning Group, Univ. of Amsterdam, Google Deepmind. 2014.

[2] 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.

[3] Diederik P. Kingma and Max Welling, An Introduction to Variational Autoencoders, Foundations and Trends R in Machine Learning: Vol. xx, No. xx, pp 1–18. 2019.

[4] Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN. Courant Institute of Mathematical Sciences, Facebook AI Research. Dec. 2017.

[5] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. indico Research, Facebook AI Research. Jan. 2016.

[6] Luke Metz el at, Unrolled Generative Adversarial Networks, Google Brain. 2017.

[7] Mehdi Mirza, Simon Osindero. Conditional Generative Adversarial Nets. Departement d'informatique et de recherche op erationnelle Universite de Montreal,, Flickr/Yahoo Inc. 2014.

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

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

[10] Yann LeCun, L eon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document. Recognition. IEEE. 1998.

[11] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, University of Toronto. 2012

[12] GeoffreyHinton, SaraSabour, NicholasFrosst. Matrix capsules with EM**,** Google Brain. 2018.

[13] Fu Jie Huang, Yann LeCun, The small NORB dataset, v1.0, Courant Institute, New York University. October, 2005.

[14] Adam R. Kosiorek, Sara Sabour, Yee Whye Teh, Geoffrey E. Hinton. Stacked Capsule Autoencoders. University of Oxford. 2019

[15] Sara Sabour, Nicholas Frosst, Geoffrey E. Hinton. Dynamic Routing Between Capsules. Google Brain. Nov. 2017.

[16] Yongheng Zhao, Tolga Birdal,Haowen Deng, Federico Tombari, 3D Point Capsule Networks. Technische Universitat Munchen, University of Padova, Siemens AG. 2019.

[17] Ayush Jaiswal, Wael AbdAlmageed, Yue Wu, Premkumar Natarajan. CapsuleGAN: Generative Adversarial Capsule Network. USC Information Sciences Institute. Mar. 2018.

[18] Raeid Saqur, Sal Vivona. CapsGAN: Using Dynamic Routing for Generative Adversarial Networks. Department of Computer Science, University of Toronto 2018.

[19] D. Wang and Q. Liu . An Optimization View on Dynamic Routing Between Capsules. International Conference on Learning Representations Workshop. 2018.

[20] S. Zhang, Q. Zhou, and X. Wu. Fast Dynamic Routing Based on Weighted Kernel Density. 2018

[21] H. Li, X. Guo, B. Dai, W. Ouyang, and X. Wang. Neural Network Encapsulation. In: *CoRR*. 2018.

[22] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. The. Set Transformer. International Conference on Machine Learning. 2019.

[23] Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift. Christian Szegedy Google Inc. 2015.

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

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

[26] ImageNet. Visual database designed for use in visual object recognition software research. Stanford Vision Lab, Stanford University, Princeton University. 2012.

[27] Kaiming, He Xiangyu, Zhang Shaoqing, Ren Jian Sun. Deep Residual Learning for Image Recognition, Microsoft Research. 2015.

[28] Inception score for MNIST. https://github.com/sundyCoder/IS_MS_SS. 2019.

Appendix A

Codes

Here we present only the Generator and Discriminator codes. Discriminator code mentioned here is just an abstract code. Sub components codes of the discriminator are not mentioned here.

A.1 Generator code

```
class Generator:
    def __init__(self, img_shape, batch_size):
        self.img_rows, self.img_cols, self.channels = img_shape
        self.batch_size = batch_size
        with tf.variable_scope('g'):
            print("Initializing generator weights")
            self.W1 = init_weights([100, 7*7*512])
            self.W2 = init_weights([3, 3, 512, 256])
            self.W3 = init_weights([3, 3, 256, 128])
            self.W4 = init_weights([3, 3, 128, 1])


    def forward(self, X, momentum=0.5):
        z = tf.matmul(X, self.W1)
        z = tf.nn.relu(z)
        z = tf.reshape(z, [-1, 7, 7, 512])

        z = UpSampling2D()(z)
        z = conv2d(z, self.W2, [1, 1, 1, 1], padding="SAME")
        z = batch_normalization(z, momentum=momentum)
        z = tf.nn.leaky_relu(z)

        z = UpSampling2D()(z)
        z = conv2d(z, self.W3, [1, 1, 1, 1], padding="SAME")
        z = batch_normalization(z, momentum=momentum)
        z = tf.nn.leaky_relu(z)

        z = conv2d(z, self.W4, [1, 1, 1, 1], padding="SAME")
        return tf.nn.tanh(z)
```

A.2 Discriminator code

```
def make_scae(canvas_size):

  template_size= 11;
  n_part_caps= 40;
  n_part_caps_dims= 6;
  n_part_special_features= 16;
  n_channels= 1;
  n_obj_caps= 32;
  n_obj_caps_params= 32;
  img_size = [canvas_size] * 2;
  template_size = [template_size] * 2;

  cnn_encoder = snt.nets.ConvNet2D(output_channels=[128] * 4,
                      kernel_shapes=[3], strides=[2, 2, 1, 1],
                      paddings=[snt.VALID], activate_final=True)

  part_encoder = primary.CapsuleImageEncoder(cnn_encoder,
                             n_part_caps, n_part_caps_dims,
                             n_features=n_part_special_features)

  part_decoder = primary.TemplateBasedImageDecoder(
              output_size=img_size, template_size=template_size,
              n_channels=n_channels)

  obj_encoder = SetTransformer(n_layers=3, n_dims=16,
                      n_output_dims=256, n_outputs=n_obj_caps)

  obj_decoder = capsule.ImageCapsule(n_obj_caps, 2, n_part_caps,
                                    n_obj_caps_params, 128)

  model = ImageAutoencoder(n_obj_caps,
              primary_encoder=part_encoder,
              primary_decoder=part_decoder, encoder=obj_encoder,
              decoder=obj_decoder, n_classes=1)

  return model


class ImageAutoencoder(snt.AbstractModule):

  def __init__(self, n_caps, primary_encoder, primary_decoder,
                             encoder, decoder, n_classes=None):
    super(ImageAutoencoder, self).__init__()
    self._primary_encoder = primary_encoder;
    self._primary_decoder = primary_decoder;
    self._encoder = encoder;
    self._decoder = decoder;
    self._n_classes = n_classes;
    self._n_caps = n_caps

  def _build(self, data, fake):

    input_x = data
    target_x = data
    batch_size = int(input_x.shape[0])
    primary_caps = self._primary_encoder(input_x)
    pres = primary_caps.presence
    pose = primary_caps.pose
    expanded_pres = tf.expand_dims(pres, -1)
    input_pose = tf.stop_gradient(tf.concat([pose, 1. -
                                        expanded_pres], -1))
    input_pres = tf.stop_gradient(pres)
    target_pose = tf.stop_gradient(pose)
    target_pres = tf.stop_gradient(pres)
    if primary_caps.feature is not None:
      input_pose = tf.concat([input_pose, primary_caps.feature],-1)

    n_templates = int(primary_caps.pose.shape[1])
    templates = self._primary_decoder.make_templates(n_templates,
                                        primary_caps.feature)
    inpt_templates = tf.stop_gradient(templates)
    if inpt_templates.shape[0] == 1:
      inpt_templates = snt.TileByDim([0],[batch_size])
                                               (inpt_templates)
    inpt_templates = snt.BatchFlatten(2)(inpt_templates)
    pose_with_templates = tf.concat([input_pose,inpt_templates],-1)

    h = self._encoder(pose_with_templates, input_pres)

    res = self._decoder(h, target_pose, target_pres)

    primary_dec_vote = primary_caps.pose
    primary_dec_pres = pres
    res.rec_ll_per_pixel = self._primary_decoder(target_x,
                         primary_dec_vote, primary_dec_pres,
                         template_feature=primary_caps.feature)
    rec_ll_per_pixel = snt.BatchFlatten()(res.rec_ll_per_pixel)
    res.rec_ll = tf.reduce_mean(tf.reduce_sum(rec_ll_per_pixel,-1))
    mass_explained_by_capsule = tf.reduce_sum(
                                  res.posterior_mixing_probs, 1)
    batch_size, num_caps = res.caps_presence_prob.shape.as_list()
    within_example_constant = float(num_caps) / self._n_classes
    res.prior_within_sparsity_loss = tf.nn.l2_loss(tf.reduce_sum(
                    res.caps_presence_prob, 1) -
                    within_example_constant) / batch_size * 2.
    between_example_constant = float(batch_size) / self._n_classes
    res.prior_between_sparsity_loss = -tf.nn.l2_loss(tf.reduce_sum(
                    res.caps_presence_prob, 0) -
                    between_example_constant) / num_caps * 2.

    def _classification_probe(features):
      linear_model = snt.Linear(self._n_classes)
      logits = linear_model(tf.stop_gradient(features))
      if fake:
        label = tf.zeros_like(logits)
      else:
        label = tf.ones_like(logits)
      cross_entropy_loss = tf.reduce_mean(
                    tf.nn.sigmoid_cross_entropy_with_logits(
                    logits=logits, labels=label))
      return logits, cross_entropy_loss

    model = snt.Module(_classification_probe)

    res.logits, res.posterior_cls_cross_entropy_loss =
                         model(mass_explained_by_capsule)
    _, res.prior_cls_cross_entropy_loss=
                         model(res.caps_presence_prob)

    res.loss = (- res.rec_ll - res.log_prob * 1. +
                    res.dynamic_weights_l2 * 10 +
                    res.prior_cls_cross_entropy_loss +
                    res.prior_within_sparsity_loss * 2. -
                    res.prior_between_sparsity_loss * 0.35 +
                    res.posterior_cls_cross_entropy_loss)

    return res
```

Appendix B

Results

B.1 Qualitative comparison between DC GAN and SCAE based GAN

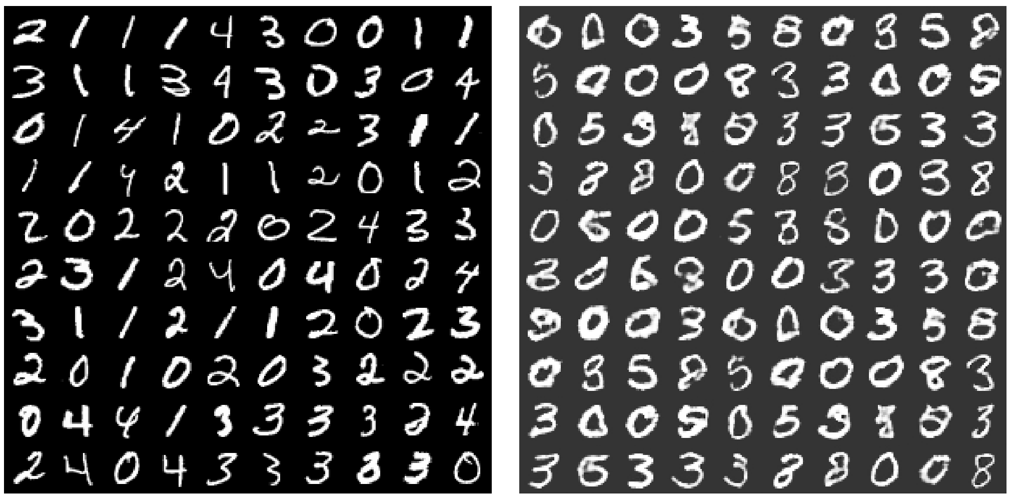

Deep Convolutional GAN SCAE based GAN

Figure B.1: Qualitative comparison of generated images

Figure B.1 shows the comparison of the visual quality of generated images between DCGAN and SCAE based GAN. Quality and diversity of the generated images of our model are acceptable when compared to DCGAN.